Fiorano

Enabling change at the speed of thought

www.fiorano.com

Fiorano SOA Platform®

User Guide

Fiorano

AMERICA'S
Fiorano Software, Inc.
230 S. California Avenue,
Suite 103, Palo Alto,
CA 94306 USA
Tel: +1 650 326 1136
Fax: +1 646 607 5875
Toll-Free: +1 800 663 3621
Email: info@fiorano.com

EMEA
Fiorano Software Ltd.
3000 Hillswood Drive Hillswood
Business Park Chertsey Surrey
KT16 0RS UK
Tel: +44 (0) 1932 895005
Fax: +44 (0) 1932 325413
Email: info_uk@fiorano.com

APAC
Fiorano Software Pte. Ltd.
Level 42, Suntec Tower Three 8
Temasek Boulevard 038988
Singapore
Tel: +65 68292234
Fax: +65 68292235
Email: info_asiapac@fiorano.com

Entire contents © Fiorano Software and Affiliates. All rights reserved. Reproduction of this document in any form without prior written permission is forbidden. The information contained herein has been obtained from sources believed to be reliable. Fiorano disclaims all warranties as to the accuracy, completeness or adequacy of such information. Fiorano shall have no liability for errors, omissions or inadequacies in the information contained herein or for interpretations thereof. The opinions expressed herein are subject to change without prior notice.

FIORANO END-USER LICENSE AGREEMENT

This Fiorano end-user license agreement (the "Agreement") is a legal agreement between you (hereinafter "Customer"), either an individual or a corporate entity, and Fiorano Software, Inc., having a place of business at 718 University Ave, Suite 212 Los Gatos, CA 95032, USA, or its affiliated companies (hereinafter "Fiorano") for certain software developed and marketed by Fiorano as defined in greater detail below. By opening this package, installing, copying, downloading, extracting and/or otherwise using the software, you are consenting to be bound by and are becoming party to this agreement on the date of installation, copying, download or extraction of the software (the "Effective Date"). If you do not agree with any of the terms of this Agreement, please stop installing and/or using the software and promptly return the unused software to the place of purchase. By default, the Software is made available to Customers in online, downloadable form. The terms of this Agreement shall apply to each Software license granted by Fiorano under this Agreement.

1. **Definitions**.

"**Affiliate**" means, in relation to Fiorano, another person firm or company which directly or indirectly controls, is controlled by or is under common control with Fiorano and the expression 'control' shall mean the power to direct or cause the direction of the general management and policies of the person firm or company in question.

"**Commencement Date**" means the date on which Fiorano delivers the Software to Customer, or if no delivery is necessary, the Effective Date set forth in this Agreement or on the relevant Order Form.

"Designated Center" means the computer hardware, operating system, customer-specific application and Customer Geographic Location at which the Software is deployed as designated on the corresponding Order Form.

"Designated Contact" shall mean the contact person or group designated by Customer and agreed to by Fiorano who will coordinate all Support requests to Fiorano.

"Documentation" means the user guides and manuals for installation and use of the Software. Documentation is provided in CD-ROM or bound form, whichever is generally available.

"Error" shall mean a reproducible defect in the Supported Program or Documentation when operated on a Supported Environment which causes the Supported Program not to operate substantially in accordance with the Documentation.

"**Excluded Components**" shall mean such components as are listed in Exhibit B. Such Excluded Components do not constitute Software under this Agreement and are third party components supplied subject to the corresponding license agreements specified in Exhibit B.

"**Excluded License**" shall mean and include any license that requires any portion of any materials or software supplied under such license to be disclosed or made available to any party either in source code or object code form. In particular, all versions and derivatives of the GNU GPL and LGPL shall be considered Excluded Licenses for the purposes of this Agreement.

"Resolution" shall mean a modification or workaround to the Supported Program and/or Documentation and/or other information provided by Fiorano to Customer intended to resolve an Error.

"**Residuals**" shall mean information in non-tangible form which may be retained by persons who have had access to the Confidential Information, including ideas, concepts, know-how or techniques contained therein.

"Order Form" means the document in hard copy form by which Customer orders Software licenses and services, and which is agreed to in writing by the parties. The Order Form shall reference the Effective Date and be governed by the terms of this Agreement. Customer understands that any document in the nature of a purchase order originating from Customer shall not constitute a contractual offer and that the terms thereof shall not govern any contract to be entered into between Fiorano and Customer. The Order Form herein shall constitute an offer to purchase made by the Customer under the terms of the said Order Form and this Agreement.

"Software" means each of the individual Products, as further outlined in Exhibit-A, in object code form distributed by Fiorano for which Customer is granted a license pursuant to this Agreement, and the media, Documentation and any Updates thereto.

"Support" shall mean ongoing support provided by Fiorano pursuant to the terms of this Agreement and Fiorano's current support policies. **"Supported Program" or "Supported Software"** shall mean the then current version of the Software in use at the Designated Center for which the Customer has paid the then-current support fee **("Support Fee")**.

"Support Hours" shall mean 9 AM to 5 PM, Pacific Standard Time, Monday through Friday, for Standard Support.

"Support Period" shall mean the period during which Customer is entitled to receive Support on a particular Supported Program, which shall be a period of twelve (12) months beginning from the Commencement Date, or if applicable, twelve (12) months from the expiration of the preceding Support Period. Should Fiorano withdraw support pursuant to section 1 (q), the Support Period shall be automatically reduced to the expiration date of the appropriate Software.

"Supported Environment" shall mean any hardware and operating system platform which Fiorano provides Support for use with the Supported Program.

"Update" means a subsequent release of the Software that Fiorano generally makes available for Supported Software licensees at no additional license fee other than shipping and handling charges. Update shall not include any release, option, feature or future product that Fiorano licenses separately. Fiorano will provide Updates for the Supported Programs as and when developed for general release in Fiorano's sole discretion. Fiorano may withdraw support for any particular version of the Software, including without limitation the most current Update and any preceding release with a notice of three (3) months to Customer.

2. **Software License**.

(a) Rights Granted, subject to the receipt by Fiorano of appropriate license fees.

(i) The Software is Licensed to Customer for use under the terms of this Agreement and **NOT SOLD**. Fiorano grants to Customer a limited, non-exclusive, world wide license to use the Software as specified on an Order Form and subject to the licensing restrictions in Exhibit C under this Agreement, as follows:

(1) to use the Software solely for Customer's operations at the Designated Center consistent with the use limitations specified or referenced in this Agreement, the Documentation for such Software or any Order Form accepted by Fiorano pursuant to this Agreement. Customer may not relicense, rent or lease the Software or use the Software for third party training, commercial timesharing or service bureau use;

(2) to use the Documentation provided with the Software in support of Customer's authorized use of the Software;

(3) to make a single copy for back-up or archival purposes and/or temporarily transfer the Software in the event of a computer malfunction. All titles, trademarks and copyright or other restricted rights notices shall be reproduced in any such copies;

(4) to allow third parties to use the Software for Customer's operations, so long as Customer ensures that use of the Software is in accordance with the terms of this Agreement.

(ii) Customer shall not copy or use the Software (including the Documentation) except as specified in this Agreement and applicable Order Form. Customer shall have no right to use other third party software or Excluded Components that are included within the Software except in connection and within the scope of Customer's use of Fiorano's Software product.

Customer agrees not to cause or permit the reverse engineering, disassembly, decompilation, or any other attempt to derive source code from the Software, except to the extent expressly provided for by applicable law.

Customer hereby warrants that it shall not, by any act or omission, cause or permit the Products or any part thereof to become expressly or impliedly subject to any Excluded License.

(v) Fiorano and its Affiliates shall retain all title, copyright and other proprietary rights in the Software. Customer does not acquire any rights, express or implied, in the Software, other than those specified in this Agreement.

(vi) Customer agrees that it will not publish or cause or permit to be published any results of benchmark tests run on the Software.

(vii) If the Software is licensed for a specific term, as noted on the Order Form, then the license shall expire at the end of the term and the termination conditions in section 4(d) shall automatically become applicable.

(b) **Transfer.** Customer may transfer a Software license within its organization upon notice to Fiorano; transfers are subject to the terms and fees specified in Fiorano's transfer policy in effect at the time of the transfer. If the Software is licensed for a specific term, then it may not be transferred by Customer.

(c) **Verification.** At Fiorano's written request, Customer shall furnish Fiorano with a signed certification verifying that the Software is being used pursuant to the provisions of this Agreement and applicable /Order Form. Fiorano (or Fiorano's designee) may audit Customer's use of the Software. Any such audit shall be conducted during regular business hours at Customer's facilities and shall not unreasonably interfere with Customer's business activities. If an audit reveals that Customer has underpaid fees to Fiorano, Customer shall be invoiced directly for such underpaid fees based on the Fiorano Price List in effect at the time the audit is completed. If the underpaid fees are in excess of five percent (5%) of the aggregate license fees paid to Fiorano pursuant to this Agreement, the Customer shall pay Fiorano's reasonable costs of conducting the audit. Audits shall be conducted no more than once annually.

(d) **Customer Specific Objects**.

(i) The parties agree and acknowledge, subject to Fiorano's underlying proprietary rights, that Customer may create certain software objects applicable to Customer's internal business ("Customer Specific Objects"). Any Customer Specific Object developed solely by Customer shall be the property of Customer. To the extent that Customer desires to have Fiorano incorporate such Customer Specific Objects into Fiorano's Software (and Fiorano agrees, in its sole discretion, to incorporate such Customer Specific Objects), Customer will promptly deliver to Fiorano the source and object code versions (including documentation) of such Customer Specific Objects, and any updates or modifications thereto, and hereby grants Fiorano a perpetual, irrevocable, worldwide, fully-paid, royalty-free, exclusive, transferable license to reproduce, modify, use, perform, display, distribute and sublicense, directly and indirectly, through one or more tiers of sublicensees, such Customer Specific Objects.

(ii) Any objects, including without limitation Customer Specific Objects, developed solely or jointly with Customer by Fiorano shall be the property of Fiorano.

(e) **Additional Restrictions on Use of Source Code**.

Customer acknowledges that the Software, its structure, organization and any human-readable versions of a software program ("Source Code") constitute valuable trade secrets that belong to Fiorano and/or its suppliers Source Code Software, if and when supplied to Customer shall constitute Software licensed under the terms of this Agreement and the Order Form. Customer agrees not to translate the Software into another computer language, in whole or in part.

(i) Customer agrees that it will not disclose all or any portion of the Software's Source Code to any third parties, with the exception of authorized employees ("Authorized Employees") and authorized contractors ("Authorized Contractors") of Customer who (i) require access thereto for a purpose authorized by this Agreement, and (ii) have signed an employee or contractor agreement in which such employee or contractor agrees to protect third party confidential information. Customer agrees that any breach by any Authorized Employees or Authorized Contractors of their obligations under such confidentiality agreements shall also constitute a breach by Customer hereunder.

(ii) Customer shall ensure that the same degree of care is used to prevent the unauthorized use, dissemination, or publication of the Software's Source Code as Customer uses to protect its own confidential information of a like nature, but in no event shall the safeguards for protecting such Source Code be less than a reasonably prudent business would exercise under similar circumstances. Customer shall take prompt and appropriate action to prevent unauthorized use or disclosure of such Source Code, including, without limitation, storing such Source Code only on secure central processing units or networks and requiring passwords and other reasonable physical controls on access to such Source Code.

(iii) Customer shall instruct Authorized Employees and Authorized Contractors not to copy the Software's Source Code on their own, and not to disclose such Source Code to anyone not authorized to receive it.

(iv) Customer shall handle, use and store the Software's Source Code solely at the Customer Designated Center.

(f) **Acceptance tested Software**

Customer acknowledges that it has, prior to the date of this Agreement, carried out adequate acceptance tests in respect of the Software. Customer's acceptance of delivery of the Software under this Agreement shall be conclusive evidence that Customer has examined the Software and found it to be complete, and in accordance with the Documentation, in good order and condition and fit for the purpose for which it is required.

3. **Technical Services**.

(a) **Maintenance and Support Services.** Maintenance and Support services will be provided under the terms of this Agreement and Fiorano's support policies in effect on the date Support is ordered by Customer. Support services shall be provided from Fiorano's principal place of business or at the Designated Center, as determined in Fiorano's sole discretion. If Fiorano sends personnel to the Designated Center to resolve any Error in the Supported Program, Customer shall pay Fiorano's reasonable travel, meals and lodging expenses.

(b) **Consulting and Training Services.** Fiorano will, upon Customer's request, provide consulting and training services agreed to by the parties pursuant to the terms of a separate written agreement.

(c) **Incidental Expenses.** For any on-site services requested by Customer, Customer shall reimburse Fiorano for actual, reasonable travel and out-of-pocket expenses incurred (separate from then current Support Fees).

(d) **Reinstatement.** Once Support has been terminated by Customer or Fiorano for a particular Supported Program, it can be reinstated only by prior approval from Fiorano and then only upon payment of the reinstatement fee applicable at the time of reinstatement.

(e) **Supervision and Management.** Customer is responsible for undertaking the proper supervision, implementation and management of its use of the Supported Programs, including, but not limited to: (i) assuring proper Supported Environment configuration, Supported Programs installation and operating methods; and (ii) following industry standard procedures for the security of data, accuracy of input and output, and back-up plans, including restart and recovery in the event of hardware or software error or malfunction. Fiorano does not warrant (i) the performance of, or combination of, Software with any third party software, (ii) any implementation of the Software that does not follow Fiorano's delivery methodology, or (iii) any components not supplied by Fiorano.

(f) **Training.** Customer is responsible for proper training of all appropriate personnel in the operation and use of the Supported Programs and associated equipment.

(g) **Access to Personnel and Equipment.** Customer shall provide Fiorano with access to Customer's personnel and its equipment during Support Hours. This access must include the ability to dial-in from Fiorano facilities to the equipment on which the Supported Programs are operating and to obtain the same access to the equipment as those of Customer's employees having the highest privilege or clearance level. Fiorano will inform Customer of the specifications of the modem equipment and associated software needed, and Customer will be responsible for the costs and use of said equipment.

(h) **Support Term.** Upon expiration of an existing Support Period for a particular Supported Program, a new Support Period shall automatically begin for a consecutive twelve (12) month term ("Renewal Period") so long as (i) Customer pays the Support Fee within thirty (30) days of invoice by Fiorano; and (ii) Fiorano is still offering Support on such Supported Program.

(i) **Annual Support Fees.** Annual Support Fees shall be at the rates set forth in the applicable Order Form.

4. **Term and Termination**.

(a) **Term.** This Agreement and each Software license granted under this Agreement shall continue unless terminated under this ***Section 4*** ("Term and Termination").

(b) **Termination by Customer.** If the Software is licensed for a specific term as noted on an Order Form, Customer may terminate any Software license at the end of the term; however, any such termination shall not relieve Customer's obligations specified in ***Section 4(d)*** ("Effect of Termination").

(c) **Termination by Fiorano.** Fiorano may terminate this Agreement or any license upon written notice if Customer breaches this Agreement and fails to correct the breach within thirty (30) days of notice from Fiorano.

(d) **Effect of Termination.** Termination of this Agreement or any license shall not limit Fiorano from pursuing other remedies available to it, including injunctive relief, nor shall such termination relieve Customer's obligation to pay all fees that have accrued or are otherwise owed by Customer under any Order Form. Such rights and obligations of the parties' which, by their nature, are intended to survive the termination of this agreement shall survive such termination. Without limitation to the foregoing, these shall include rights and liabilities arising under Sections ***2 (a)(iii), 2(a)(iv)*** ("Rights Granted")***, 2(d)*** ("Customer Specific Objects"), ***4*** ("Term and Termination"), ***5*** ("Indemnity, Warranties, Remedies"), ***6*** ("Limitation of Liability"), ***7*** ("Payment Provisions"), ***8*** ("Confidentiality") and ***9*** ("Miscellaneous") Upon termination, Customer shall cease using, and shall return or at Fiorano's request destroy, all copies of the Software and Documentation and upon Fiorano's request certify the same to Fiorano in writing within thirty (30) days of termination. In case of termination of this Agreement or any license for any reason by either party, Fiorano shall have no obligation to refund any amounts paid to Fiorano by Customer under this Agreement. Further, if Customer terminates the agreement before the expiry of a term for a term-license, then Customer shall be obliged to pay the entire license fee for the entire licensed term.

5. **Indemnity, Warranties, Remedies**.

(a) **Infringement Indemnity.** Fiorano agrees to indemnify Customer against a third party claim that any Product infringes a U.S. copyright or patent and pay any damages finally awarded, provided that: (i) Customer notifies Fiorano in writing within ten (10) days of the claim; (ii) Fiorano has sole control of the defense and all related settlement negotiations; and (iii) Customer provides Fiorano with the assistance, information and authority at no cost to Fiorano, necessary to perform Fiorano's obligations under this ***Section 5*** ("Indemnities, Warranties, Remedies"). Fiorano shall have no liability for any third party claims of infringement based upon (i) use of a version of a Product other than the most current version made available to the Customer, (ii) the use, operation or combination of any Product with programs, data, equipment or documentation if such infringement would have been avoided but for such use, operation or combination; or (iii) any third party software, except as the same may be integrated, incorporated or bundled by Fiorano, or its third party licensors, in the Product licensed to Customer hereunder.

If any Product is held or claimed to infringe, Fiorano shall have the option, at its expense, to (i) modify the Product to be non-infringing or (ii) obtain for Customer a license to continue using the Software. If it is not commercially reasonable to perform either of the above options, then Fiorano may terminate the license for the infringing Product and refund the pro rated amount of license fees paid for the applicable Product using a twelve (12) month straight-line amortization schedule starting on the Commencement Date. This ***Section 5(a)*** ("Infringement Indemnity") states Fiorano's entire liability and Customer's sole and exclusive remedy for infringement.

(B) **WARRANTIES AND DISCLAIMERS**.

(i) Software Warranty. Except FOR EXCLUDED COMPONENTS WHICH ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, For each Supported Software license which Customer acquires hereunder, Fiorano warrants that for a period of thirty (30) days from the Commencement Date the Software, as delivered by Fiorano to Customer, will substantially perform the functions described in the associated Documentation in all material respects when operated on a system which meets the requirements specified by Fiorano in the Documentation. Provided that Customer gives Fiorano written notice of a breach of the foregoing warranty during the warranty period, Fiorano shall, as Customer's sole and exclusive remedy and Fiorano's sole liability, use its reasonable efforts, during the warranty period only, to correct any reproducible Errors that cause the breach of the warranty in accordance with its technical support policies. If Customer does not obtain a Supported Software license, the Software is provided "AS IS." any implied warranty or condition applicable to the software, documentation or any part thereof by operation of any law or regulation shall operate only for defects discovered during the above warranty period of thirty (30) days unless temporal limitation on such warranty or condition is expressly prohibited by applicable law. Any supplements or updates to the Software, including without limitation, bug fixes or error corrections supplied after the expiration of the thirty-day Limited Warranty period SHALL NOT be covered by any warranty or condition, express, implied or statutory.

(ii) Media Warranty. Fiorano warrants the tapes, diskettes or any other media on which the Software is supplied to be free of defects in materials and workmanship under normal use for thirty (30) days from the Commencement Date. Customer's sole and exclusive remedy and Fiorano's sole liability for breach of the media warranty shall be for Fiorano to replace defective media returned within thirty (30) days of the Commencement Date.

(iii) Services Warranty. Fiorano warrants any services provided hereunder shall be performed in a professional and workmanlike manner in accordance with generally accepted industry practices. This warranty shall be valid for a period of thirty (30) days from performance. Fiorano's sole and exclusive liability and Customer's sole and exclusive remedy pursuant to this warranty shall be use by Fiorano of reasonable efforts for re-performance of any services not in compliance with this warranty which are brought to Fiorano's attention by written notice within fifteen (15) days after they are performed.

(IV) DISCLAIMER OF WARRANTIES. SUBJECT TO LIMITED WARRANTIES PROVIDED FOR HEREINABOVE, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THE SOFTWARE, DOCUMENTATION AND SERVICES (IF ANY) ARE PROVIDED *AS IS AND WITH ALL FAULTS*, FIORANO HEREBY DISCLAIMS ALL OTHER WARRANTIES AND CONDITIONS, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY (IF ANY) IMPLIED WARRANTIES, DUTIES OR CONDITIONS OF MERCHANTABILITY, OF FITNESS FOR A PARTICULAR PURPOSE, OF RELIABILITY OR AVAILABILITY, OF ACCURACY OR COMPLETENESS OF RESPONSES, OF RESULTS, OF WORKMANLIKE EFFORT, OF LACK OF VIRUSES, AND OF LACK OF NEGLIGENCE, ALL WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT OR OTHER SERVICES, INFORMATION, SOFTWARE, AND RELATED CONTENT THROUGH THE SOFTWARE OR OTHERWISE ARISING OUT OF THE USE OF THE SOFTWARE. ALSO, THERE IS NO WARRANTY OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT WITH REGARD TO THE SOFTWARE.

6. Limitation of liability. To the maximum extent permitted by applicable law, in no event shall fiorano be liable for any special, incidental, punitive, indirect, or consequential damages whatsoever (including, but not limited to, damages for loss of profits or confidential or other information, for business interruption, for personal injury, for loss of privacy, for failure to meet any duty of good faith or of reasonable care, for negligence, and for any other pecuniary or other loss whatsoever) arising out of or in any way related to the use of or inability to use the software, the provision of or failure to provide support or other services, information, software, and related content through the software, or otherwise under or in connection with any provision of this eula, even in the event of the fault, tort (including negligence), misrepresentation, strict liability, breach of contract or breach of warranty of fiorano, and even if fiorano or any supplier has been advised of the possibility of such damages.

Notwithstanding any damages that may be incurred for any reason and under any circumstances (including, without limitation, all damages and liabilities referenced herein and all direct or general damages in law, contract or anything else), the entire liability of fiorano under any provision of this eula and the exclusive remedy of the customer hereunder (except for any remedy of repair or replacement if so elected by fiorano with respect to any breach of the limited warranty) shall be limited to the pro-rated amount of fees paid by customer under this agreement for the product, using a twelve (12) month straight-line amortization schedule starting on the Commencement Date. Further, if such damages result from customer's use of the software or services, such liability shall be limited to the prorated amount of fees paid for the relevant software or services giving rise to the liability till the date when such liability arose, using a twelve (12) month straight-line amortization schedule starting on the Commencement Date. Notwithstanding anything in this agreement, the foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails its essential purpose.

The provisions of this Agreement allocate the risks between Fiorano and Customer. Fiorano's pricing reflects this allocation of risk and the limitation of liability specified herein.

7. **Payment Provisions**.

(a) Invoicing. All fees shall be due and payable thirty (30) days from receipt of an invoice and shall be made without deductions based on any taxes or withholdings. Any amounts not paid within thirty (30) days will be subject to an immediately due and payable late payment fee equivalent to: the sum of $50.00 plus an interest equal to the lower of (a) the maximum applicable legal interest rate, or (b) one percent (1%) per month.

(b) Payments. All payments made by Customer shall be in United States Dollars for purchases made in all countries except the United Kingdom or the European Union, in which case the payments shall be made in British Pounds Sterling or Euros respectively. Payments shall be directed to:

Fiorano Software, Inc.

718 University Ave.

Suite 212, Los Gatos, CA 95032

Attn: Accounts Receivable.

If the product is purchased outside the United States, payments may have to be made to an Affiliate as directed by Fiorano Software, Inc.

(c) Taxes. The fees listed in this Agreement or the applicable Order Form does not include Taxes. In addition to any other payments due under this Agreement, Customer agrees to pay, indemnify and hold Fiorano harmless from, any sales, use, excise, import or export, value added or similar tax or duty, and any other tax not based on Fiorano's net income, including penalties and interest and all government permit fees, license fees, customs fees and similar fees levied upon the delivery of the Software or other deliverables which Fiorano may incur in respect of this Agreement, and any costs associated with the collection or withholding of any of the foregoing items (the "Taxes").

8. **Confidentiality**.

(a) Confidential Information. "Confidential Information" shall refer to and include, without limitation, (i) the source and binary code of Products, and (ii) the business and technical information of either party, including but not limited to any information relating to product plans, designs, costs, product prices and names, finances, marketing plans, business opportunities, personnel, research, development or know-how;

Exclusions of Confidential Information. Notwithstanding the foregoing, "Confidential Information" shall not include: (i) Information that is not marked confidential or otherwise expressly designated confidential prior to its disclosure, (ii) Information that is or becomes generally known or available by publication, commercial use or otherwise through no fault of the receiving party, (iii) Information that is known to the receiving party at the time of disclosure without violation of any confidentiality restriction and without any restriction on the receiving party's further use or disclosure; (iv) Information that is independently developed by the receiving party without use of the disclosing party's confidential information, or (v) Any Residuals arising out of this Agreement. Notwithstanding, any Residuals belonging to Source Code shall belong exclusively to Fiorano and Customer shall not have any right whatsoever to any Residuals relating to Source Code hereunder.

Use and Disclosure Restrictions. During the term of this Agreement, each party shall refrain from using the other party's Confidential Information except as specifically permitted herein, and from disclosing such Confidential Information to any third party except to its employees and consultants as is reasonably required in connection with the exercise of its rights and obligations under this Agreement (and only subject to binding use and disclosure restrictions at least as protective as those set forth herein executed in writing by such employees).

Continuing Obligation. The confidentiality obligation described in this section shall survive for three (3) years following any termination of this Agreement.Notwithstanding the foregoing, Fiorano shall have the right to disclose Customer's Confidential Information to the extent that it is required to be disclosed pursuant to any statutory or regulatory provision or court order, provided that Fiorano provides notice thereof to Customer, together with the statutory or regulatory provision, or court order, on which such disclosure is based, as soon as practicable prior to such disclosure so that Customer has the opportunity to obtain a protective order or take other protective measures as it may deem necessary with respect to such information.

9. **Miscellaneous**.

(a) Export Administration. Customer agrees to comply fully with all applicable relevant export laws and regulations including without limitation, those of the United States ("Export Laws") to assure that neither the Software nor any direct product thereof are (i) exported, directly or indirectly, in violation of Export Laws; or (ii) are intended to be used for any purposes prohibited by the Export Laws, including, without limitation, nuclear, chemical, or biological weapons proliferation.

(b) **U. S. Government Customers.** The Software is "commercial items," as that term is defined at 48 C.F.R. 2.101 (OCT 1995), consisting of "commercial computer software" and "commercial computer software documentation" as such terms are used in 48 C.F.R. 12.212 (SEPT 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (JUNE 1995), all U.S. Government Customers acquire the Software with only those rights set forth herein.

(c) **Notices.** All notices under this Agreement shall be in writing and shall be deemed to have been given when mailed by first class mail five (5) days after deposit in the mail. Notices shall be sent to the addresses set forth at the beginning of this Agreement or such other address as either party may specify in writing.

(d) **Force Majeure.** Neither party shall be liable hereunder by reason of any failure or delay in the performance of its obligations hereunder (except for the payment of money) on account of strikes, shortages, riots, insurrection, fires, flood, storm, explosions, acts of God, war, governmental action, labor conditions, earthquakes, material shortages or any other cause which is beyond the reasonable control of such party.

(e) **Assignment.** Neither this Agreement nor any rights or obligations of Customer hereunder may be assigned by Customer in whole or in part without the prior written approval of Fiorano. For the avoidance of doubt, any reorganization, change in ownership or a sale of all or substantially all of Customer's assets shall be deemed to trigger an assignment. Fiorano's rights and obligations, in whole or in part, under this Agreement may be assigned by Fiorano.

(f) **Waiver.** The failure of either party to require performance by the other party of any provision hereof shall not affect the right to require such performance at any time thereafter; nor shall the waiver by either party of a breach of any provision hereof be taken or held to be a waiver of the provision itself.

(g) **Severability.** In the event that any provision of this Agreement shall be unenforceable or invalid under any applicable law or court decision, such unenforceability or invalidity shall not render this Agreement unenforceable or invalid as a whole and, in such event, any such provision shall be changed and interpreted so as to best accomplish the objectives of such unenforceable or intended provision within the limits of applicable law or applicable court decisions.

(h) **Injunctive Relief.** Notwithstanding any other provisions of this Agreement, a breach by Customer of the provisions of this Agreement regarding proprietary rights will cause Fiorano irreparable damage for which recovery of money damages would be inadequate, and that, in addition to any and all remedies available at law, Fiorano shall be entitled to seek timely injunctive relief to protect Fiorano's rights under this Agreement.

(i) **Controlling Law and Jurisdiction.** If this Software has been acquired in the United States, this Agreement shall be governed in all respects by the laws of the United States of America and the State of California as such laws are applied to agreements entered into and to be performed entirely within California between California residents. All disputes arising under this Agreement may be brought in Superior Court of the State of California in Santa Clara County or the United States District Court for the Northern District of California as permitted by law. If this Software has been acquired in any other jurisdiction, the laws of the Republic of Singapore shall apply and any disputes arising hereunder shall be subject to the jurisdiction of the courts of Singapore, Singapore. Customer hereby consents to personal jurisdiction of the above courts. The parties agree that the United Nations Convention on Contracts for the International Sale of Goods is specifically excluded from application to this Agreement.

(j) **No Agency.** Nothing contained herein shall be construed as creating any agency, partnership or other form of joint enterprise or liability between the parties.

(k) **Headings.** The section headings appearing in this Agreement are inserted only as a matter of convenience and in no way define, limit, construe or describe the scope or extent of such section or in any way affect such section.

(l) **Counterparts.** This Agreement may be executed simultaneously in two or more counterparts, each of which will be considered an original, but all of which together will constitute one and the same instrument.

(m) **Disclaimer.** The Software is not specifically developed or licensed for use in any nuclear, aviation, mass transit or medical application or in any other inherently dangerous applications. Customer agrees that Fiorano and its suppliers shall not be liable for any claims or damages arising from Customer's use of the Software for such applications. Customer agrees to indemnify and hold Fiorano harmless from any claims for losses, costs, damages or liability arising out of or in connection with the use of the Software in such applications.

(n) **Customer Reference**. Fiorano may refer to Customer as a customer in sales presentations, marketing vehicles and activities. Such activities may include, but are not limited to; a press release, a Customer user story completed by Fiorano upon implementation of the Software, use by Fiorano of Customer's name, logo and other marks, together with a reasonable number of technical or executive level Customer reference calls for Fiorano.

(o) **Entire Agreement.** This Agreement, together with any exhibits, completely and exclusively states the agreement of the parties. In the event of any conflict between the terms of this Agreement and any exhibit hereto, the terms of this Agreement shall control. In the event of any conflict between the terms of this Agreement and any purchase order or Order Form, this Agreement will control, and any pre-printed terms on Customer's purchase order or equivalent document will be of no effect. This Agreement supersedes, and its terms govern, all prior proposals, agreements or other communications between the parties, oral or written, regarding the subject matter of this Agreement. This Agreement shall not be modified except by a subsequently dated written amendment signed by the parties, and shall prevail over any conflicting "pre-printed" terms on a Customer purchase order or other document purporting to supplement the provisions hereof.

Exhibit A

Fiorano Product List

Each of the individual items below is a separate Fiorano product (the "Product"). The Products in this list collectively constitute the Software. Fiorano reserves the right to modify this list at any time in its sole discretion. In particular, Product versions might change from time to time without notice.

Fiorano SOA Enterprise Server

Fiorano ESB Server

FioranoMQ Server Peer / FioranoMQ (standalone version)

Fiorano Peer Server

Fiorano SOA Tools

Fiorano Mapper Tool

Fiorano Database Business Component

Fiorano HTTP Business Component

Fiorano SMTP Business Component

Fiorano FTP Business Component

Fiorano File Business Component

Fiorano MOM Business Components (MQSeries, MSMQ, JMS)

NOTE: Other business components may be added to or removed from this list from time to time at Fiorano's sole discretion.

Exhibit B

EXCLUDED COMPONENTS

(a) Any third party or open source library included within the Software

Exhibit C

Licensing Restrictions. The Software licensed hereunder is subject to the following licensing restrictions.

The parties understand that the modules of the Software are licensed as noted in this section. The term "Target System" means any computer system containing one or more Processors based upon any architecture, running any operating system, excluding computers running IBM MV-S, OS/390 and related "mainframe" operating systems. The Term "Processor" means a computation hardware unit such as a Microprocessor that serves as the main arithmetic and logic unit of a computer. A Processor might consist of multiple "Cores", in which case licenses shall have to be purchased on a per-Core basis. A Target System may have one or more Processors, each of which may have one or more Cores. In the sections below, Cores may replace Processors as applicable.

If the Software is Fiorano ESB Enterprise Server, FioranoMQ Peer, Fiorano SOA server or FioranoMQ Server (JMS), then the Software is licensed on a per Processor basis on a single Target System, where the total number of Processors on the Target System may not exceed the total number of Processors licensed, with the additional restriction that only a single instance of the Fiorano ESB Enterprise Server may run on a single Target System and that a separate license must be purchased for each instance of the Fiorano ESB Enterprise Server, Fiorano ESB Peer Server or FioranoMQ Server (JMS) Server for each Processor;

If the Software is Fiorano SOA Tools or Fiorano Mapper Tool , or any Fiorano Test and/or Development license, then the Software is licensed on a per-named-user basis, where the total number of named users may not exceed the total number of named users licensed;

If the Software is a Fiorano Business Component of any kind (including but not limited to Fiorano HTTP, File, SMTP, File, Database, and other Business Components, etc.), then the Software is licensed on the basis of the number of CPUs of the Target System on which the FioranoMQ Peer (to which the Business Component connects runs). A separate license needs to be purchased for each CPU of each Target System of each FioranoMQ Peer instance to which any Business Component connects.

Evaluations. Licenses used for evaluation cannot be used for any purposes other than an evaluation of the product. Existing customers must purchase new licenses to use additional copies of any Product and may not use evaluation keys in any form. All evaluation keys are restricted to 45-days and extensions need to be applied for explicitly. Any misuse of evaluation keys shall be subject to a charge of 125% (one hundred and twenty-five percent) of the license fee plus 20%

Non-Production Environments. For all non-production environments referenced on the Order Form (including all HA (high-availability), QA, Staging and Development environments), the following is understood: each non-production environment is an exact replica of the Production Environment from the standpoint of the number of copies of the Fiorano ESB Enterprise Server, FioranoMQ Peer, Fiorano SOA server and/or FioranoMQ Server (JMS) licensed. Each non-production environment is licensed on the exact same number and configuration of CPUs and/or Cores as the corresponding Production Environment.

Run-Time Libraries. The Fiorano ESB Enterprise Server, FioranoMQ Peer, Fiorano SOA server and FioranoMQ Server (JMS) products are "server" products, each of which has a runtime library associated with it. The runtime library may be freely bundled with and/or used for internal development purposes by all Users who have licensed at least one production copy of the corresponding Server Software.

Copyright (c) 2008-2012, Fiorano Software Pte Ltd. and Affiliates

Contents

Chapter 1: Introduction 48

1.1 What is Fiorano SOA Platform 48
1.2 Why Fiorano SOA Platform 48
1.3 Introduction to Fiorano SOA Platform Environment 49
1.3.1 Fiorano Servers 49
1.3.2 Fiorano Tools 50
1.3.3 Composing an Event Process 50
1.3.4 Deploying Event Processes 55
1.3.5 Monitoring Event Processes 55
1.3.6 Extending the Component Palette 59
1.3.7 Scalability 60
1.3.8 High Availability 60
1.3.9 Security Framework 61

Chapter 2: The Fiorano Environment 62

2.1 Fiorano System Architecture 62
2.1.1 Fiorano ESB Server 63
2.1.2 Fiorano Peer Server 63
2.2 Installation 64
2.2.1 Different Topologies 64
2.2.2 ESB Server 65
2.2.2.1 System Requirements 65
2.2.2.2 Installation Steps 66
2.2.3 ESB Peers 66
2.2.3.1 System Requirements 66
2.2.3.2 Installation Steps 66
2.3 Fiorano ESB Server 68
2.3.1 ESB Server Functionality 69
2.3.2 Launching ESB Server 70
2.3.2.1 From Fiorano Studio 70
2.3.2.2 From Script Files 70
2.3.3 Shutting Down ESB Server 71
2.3.3.1 From Fiorano Studio 71
2.3.3.2 From Script Files 72
2.3.4 ESB Server Configuration 73
2.3.4.1 Server Ports Configuration 73
2.3.4.2 Memory Configurations 83
2.3.4.3 Java Configurations 83
2.3.5 Setting Up Users and Groups 84

2.3.5.1 Managing Users 84
2.3.5.2 Creating a New User Account 85
2.3.5.3 Configuration Steps 85
2.3.5.4 Managing Groups 87
2.3.5.5 Creating New Group 88
2.3.5.6 Adding a User to a Group 88
2.3.5.7 Deleting a User from a Group 89
2.3.5.8 Deleting a Group 90
2.3.5.9 Setting Access Controls 91
2.3.5.10 Assigning Rights 91
2.3.5.11 Removing Network Rights 93
2.3.6 Clearing ESB Server Database 93
2.4 Fiorano Peer Server 95
2.4.1 Peer Server Functionality 95
2.4.2 Launching the Peer Server 95
2.4.2.1 Launching the Peer Server From Fiorano Studio 95
2.4.2.2 Launching the Peer Server Using Script Files 96
2.4.3 Shutting Down the Peer Server 97
2.4.3.1 Using Fiorano Studio 97
2.4.3.2 Using Script Files 97
2.4.4 Peer Server Configuration 99
2.4.4.1 Server Ports Configuration 99
2.4.4.2 Memory Configurations 107
2.4.4.3 Java Configurations 107
2.4.4.4 Changing to different ESB Network 108
2.4.5 Adding New Peer Server 109
2.4.6 Clearing Peer Server Database 114
2.5 Fiorano Web Console 115
2.5.1 Login Page 115
2.5.1.1 Events 116
2.5.1.2 Security 122
2.5.1.3 Applications 127
2.5.1.4 Monitoring 129
2.5.1.5 Server Status 133
2.5.1.6 Audit Management 134
2.5.1.7 Document Tracking 136
2.5.1.8 Services 139
2.5.1.9 Resource Search 141
2.5.2 Enabling Fiorano Web Console 142
2.6 Configuring Servers and Tools 143
2.6.1 Configuration File 143
2.6.2 Reference Matrix 144
2.6.3 Configuring the Jetty Server with SSL Support 146
2.6.3.1 SSL Configuration for Jetty 147
2.6.4 Using Basic Authentication with the Jetty Server 151
2.6.4.1 Configuring Jetty Server 151

2.6.4.2 Enabling Basic Authentication with bcwsgateway 152
2.6.4.3 Enabling Basic Authentication with WSStub 152
2.6.4.4 Testing Services from Dashboard 152
2.6.4.5 Testing Services from Web Service Consumer 154
2.6.5 Adding Additional Port for Peer to Peer Communication 155
2.6.6 Configuring Server Execution Timeout 161
2.6.7 SNMP Configuration 162
2.6.7.1 Fiorano SOA 9 Platform MIB 163
2.6.7.2 Fiorano SOA 9 Platform Agent 163
2.6.7.3 How to receive SNMP Traps 163
2.6.7.4 Getting the information 163
2.6.7.4 Trap Messages 165
2.6.7.5 Configuring the SNMP parameters for the Fiorano ESB Server 165
2.6.7.7 Using iReasoning MIB tool 168
2.6.8 Setting the Offline Server Log Levels 171
2.6.9 Changing the Running Server Log Levels 173
2.7 Fiorano Enterprise Repository 177
2.7.1 Event Process Repository 177
2.7.2 Component Repository 178
2.7.3 Peer Repository 179
2.7.4 Security Store 179
2.7.5 Runtime Store 180
2.7.6 State Based Workflow Repository 181
2.7.6.1 Disabling SBW Tracking 181
2.7.7 Event Repository 182
2.7.8 Custom Event Listener 182
2.7.9 Alert Repository 182
2.7.10 Policy Repository 183
2.7.11 Changes in Repository Location 183
2.8 Events Tracking 184
2.8.1 Configuring Event Tracking 184
2.8.1.1 Disabling Event Tracking 185
2.8.2 Configuring Specific Database 186
2.8.3 Database Table Structure 187
2.9 Connecting to Server Using JConsole 190
2.9.1 How to connect to the FES/FPS using jconsole [running JDK6] SOA 9 190
2.9.2 Starting multiple servers in same machine 192
2.10 Running Fiorano SOA Profiles as NT Services 192
2.10.1 Configuring Server JVM Settings for running as NT-Service 192
2.10.2 Configuring Email Alerts for Server JVM Restart/Sudden Exit/JVM Hung Events . 193
2.10.3 How to install/uninstall a StandAlone (Non-High Availabilty)Profile as NT Service193
2.10.4 How to install/uninstall a Profile as NT Service with High Availability Profile(HA) 194
2.10.5 Viewing Server Logs 196
2.10.6 FAQs 196
2.11 Event Monitoring 198
2.11.1 Event Types And Subscription 198

2.11.2 Audit Events 199
2.11.2.1 Subscribing to Audit Events 200
2.11.3 System Events 200
2.11.3.1Event Description 201
2.11.3.2 Subscribing to System Events 201
2.11.4 Event Topics 202
2.11.5 Sample Subscriber Applications 203
2.11.6 Built-In Event Tracking / Reporting 203
2.11.7 Event Parameters 204
2.11.7.1Examples 205
2.11.7.2 Measuring Response Time 205
2.11.7.3 Measuring Throughput 205
2.11.7.4 Service Status Monitoring 206
2.11.7.5 Messaging Threshold 206
2.11.7.6 Server Memory Threshold 206

Chapter 3: Component and Component Instances...207

3.1 Service Components Characteristics 207
3.1.1 Synchronous Components 207
3.1.2 Asynchronous Components 208
3.1.3 Design Choices 208
3.2 Service Component Characteristics, Configuration, and Deployment 209
3.2.1 Component Launch Semantics 209
3.2.2 Setting Component Launch Type in the Fiorano Studio 210
3.2.3 Launching Components Using the Fiorano Studio 210
3.2.3.1 Launching a Component in a Running Application 211
3.2.3.2 Stopping a Running Component Instance 211
3.3 Service Component Configuration 213
3.3.1 CPS for Component instance configuration 213
3.3.1.1 Launching the CPS 213
3.3.1.2 Customizable and Expert Properties 215
3.3.1.3 Online Help for components 215
3.3.1.4 Runtime Arguments 217
3.3.2 Component Dependencies and System Libraries 218
3.3.2.1 Viewing the Resources of a Component 219
3.3.3 Add New Library Dependencies 221
3.3.3.1 Adding New Resource/Dependency 221
3.3.3.2 Adding Service Dependencies 222
3.3.3.3 Adding Resources to Class Path 225
3.3.3.4 Component Classpath Generation 227
3.3.4 Creating New System Libraries 228
3.3.4.1 Adding a New System Library 229
3.3.5 Configuring Logging Parameters 232
3.4 Component Deployment 235
3.4.1 Adding Ports for the Component 239

3.4.2 Adding Log Modules for the Component 241
3.4.3 Adding Runtime Arguments for the Component 242
3.4.4 Adding New Parameters to the Component 244
3.4.5 Adding Node Name to a Component Instance 246
3.4.6 Manual Deployment 247
3.4.6.1 From Scriptgen Tool 248
3.4.6.2 From the configureBC and runBC utilities 251
3.4.7 External Deployment 255
3.4.7.1 Deploying a Synchronous Component in JBoss Application Server 255
3.4.7.2 Additional Features for Component Administration 258
3.4.8 InMemory Launch 262
3.5 Export and Import Service Components 264
3.5.1 Exporting a Component 264
3.5.2 Importing a Component 265
3.6 Component Creation 266
3.6.1 Template Engine 267
3.6.1.1 Component Creation from the Command Line 267
3.6.1.2 Creating a Setting 268
3.6.1.3 Variables 268
3.6.1.4 Modifying the Templates 270
3.6.1.5 Defining Components 271
3.6.1.6 Getting familiar with wizard and service configuration 272
3.6.1.7 Generating Code for the Defined Component 287
3.6.1.8 Building the Component 288
3.6.1.9 Deploying the Component 288
3.6.2 Component Creation in Fiorano Studio 289
3.6.3 Java Components 291
3.6.3.1 Defining Asynchronous Component 291
3.6.3.2 Creating a Synchronous Component 305
3.6.4 Non-Java components 323
3.6.4.1 Defining a C component 323
3.6.4.2 Creating a C++ component. 326
3.6.4.3 Creating a C# Component 328
3.7 Service Component Testing 332
3.7.1 Testing Synchronous Components 332
3.7.1.1 Testing in Configuration Property Sheet (CPS) 332
3.7.1.2 Testing using JUnit test cases 340
3.7.2 Testing Asynchronous Components 344
3.7.2.1 Configuring a JUnit test case 344
3.7.2.2 Executing a JUnit test case 347
3.8 Component Generation - SimpleJMS, MultiThreaded and POJO 349
3.8.1 EDBC Templates 349
3.8.1.1 Scripts 349
3.8.1.2 Simple JMS 349
3.8.1.3 Multi threaded 352
3.8.1.4 POJO 355

3.9 Eclipse IDE Support 358
3.9.1 Importing the Project into Eclipse 358
3.9.2 Defining Variables 362
3.9.3 Defining ANT_HOME 365
3.9.4 Defining JDK 368
3.9.5 Compiling Deploying and Registering the Component 369
3.10 Text Schema Editor 372
3.10.1 Text Format Layout Concepts 374
3.10.2 Launch Fiorano Text Schema Editor 374
3.10.2.1 Defining Text File Schemas 376
3.10.2.2 Using the Text Schema Editor 384
3.10.2.3 Warnings 390
3.10.2.4 Limitations 390
3.11 Public Key, Cryptography Keystore, And Truststore 391
3.11.1 Using Public Key Cryptography for Authentication 391
3.11.2 Keystore and Truststore 392
3.11.2.1 Generating a Client Keystore 394
3.11.2.2 Getting the Digital Certificate of Server 395
3.11.2.3 Creating the Client Truststore 397
3.11.2.4 Using the Keystore and Truststore in an SSL Application 398
3.11.3 Generating self-signed X509 Certificates 404
3.11.3.1 Generate a Certificate Signing Request (CSR) and sign the request using openssl 404
3.11.3.2 Generate self-signed certificate without generating CSR using openssl 404
3.11.3.3 Generate a CSR using keytool and sign the request using openssl 404
3.12 Component Control Protocol (CCP) 405
3.12.1 Communication Channels 405
3.12.2 Events 405
3.12.3 Component Lifecycle and CCP 405
3.12.4 Sample component 407

Chapter 4: Pre-built Components 408

4.1 Bridges 410

4.1.1 EJBAdapter 410
4.1.1.1 Configuration and Testing 411
4.1.1.1.1 Managed Connection Factory Panel 411
4.1.1.2 Interaction Configurations Panel 414
4.1.1.2.1 Using Named Configurations 416
4.1.1.3 Functional Demonstration 417
4.1.1.4 Input Message 417
4.1.1.5 Output Message 418
4.1.2 FTPGet 418
4.1.2.1 Managed Connection Factory Panel 419

4.1.2.1.1 Connection Properties tab 419
4.1.2.1.2 Advanced Settings tab 425
4.1.2.1.3 Testing the Connection 428
4.1.2.2 Interaction Configurations Panel 429
4.1.2.2.1 Source Setting 429
4.1.2.2.2 Monitor Settings 433
4.1.2.2.3 Target Settings 436
4.1.2.2.4 Using Named Configurations 438
4.1.2.2.5 Miscellaneous Settings 439
4.1.2.3 Miscellaneous Settings 439
4.1.2.4 Testing the Interaction Configurations 440
4.1.2.5 Input Schema 443
4.1.2.6 Output Schema 444
4.1.2.7 Functional Demonstration 445
4.1.2.7.1 Scenario 1 445
4.1.2.7.2 Scenario 2 445
4.1.2.7.3 Scenario 3 (Scenario 1 Using SFTP Protocol) 446
4.1.2.8 Configuring FTPGet Component 447
4.1.2.9 Use Case Scenario 448
4.1.2.10 Scheduling 448
4.1.2.11 Useful Tips 449
4.1.3 FTPPut 449
4.1.3.1 Managed Connection Factory Panel 450
4.1.3.1.1 Connection Properties 450
4.1.3.1.2 Advanced Settings tab 455
4.1.3.1.3 Using Named Configurations 457
4.1.3.1.4 Testing the Connection 459
4.1.3.2 Interaction Configurations Panel 460
4.1.3.2.1 Source Settings 460
4.1.3.2.2 Monitor Settings 462
4.1.3.2.3 Target Settings 465
4.1.3.2.5 Miscellaneous Settings 466
4.1.3.3 Input and Output 466
4.1.3.4 Testing the Interaction Configurations 470
4.1.3.5 Functional Demonstration 471
4.1.3.5.1 Scenario 1 471
4.1.3.5.2 Scenario 2 472
4.1.3.5.3 Scenario 3 (Scenario 1 Using SFTP Protocol) 472
4.1.3.6 Configuring FTPPut Component 473
4.1.3.7 Use Case Scenario 474
4.1.3.8 Useful Tips 474
4.1.4 IWay 475
4.1.4.1 Configuration and Testing 475
4.1.4.1.1 Using Named Configurations 475
4.1.4.2 Input Schema 477
4.1.4.3 Output Schema 477

4.1.4.4 Functional Demonstration 477
4.1.4.5 Use Case Scenario 478
4.1.4.6 Useful Tips 478
4.1.5 POP3 479
4.1.5.1 Managed Connection Factory 479
4.1.5.1.1 Attributes 479
4.1.5.2 Interaction Configuration 483
4.1.5.2.1 Attributes 483
4.1.5.3 Sample Input and Output 487
4.1.5.3.1 Input Schema 488
4.1.5.3.2 Output Schema 488
4.1.5.4 Functional Demonstration 490
4.1.5.4.1 Scenario 1 490
4.1.5.4.2 Scenario 2 491
4.1.5.5 Useful Tips 491
4.1.6 SAPR3 491
4.1.6.1 Configuration and Testing 492
4.1.6.1.1 Connection Properties 492
4.1.6.1.2 Interaction Properties 493
4.1.6.2 Functional Demonstration: 495
4.1.6.2.1 Scenario 495
4.1.6.3 Useful Tips 497
4.1.7 SMS Bridge 497
4.1.7.1 Configuration and Testing 497
4.1.7.2 Managed Connection Factory Configuration 498
4.1.7.2.1 Using Named Configurations 499
4.1.7.3 Functional Demonstration 501
4.1.7.3.1 Scenario 1 501
4.1.7.4 Useful Tips 502
4.1.8 SMTP 502
4.1.8.1 Managed Connection Factory Panel 503
4.1.8.1.1 SMTP Server 503
4.1.8.1.2 Authentication Details 504
4.1.8.1.3 TimeOut Settings 504
4.1.8.1.4 AS1 Configuration 504
4.1.8.2 Interaction Configurations Panel 507
4.1.8.2.1 Sender Information 507
4.1.8.3 Input 508
4.1.8.4 Output 510
4.1.8.5 Functional Demonstration 510
4.1.8.5.1 Scenario 1 510
4.1.8.6 Use Case Scenario 511
4.1.8.7 Useful Tips 511
4.1.9 SapR3Monitor 512
4.1.10 HL7Receiver 512
4.1.10.1 Configuration and Testing 512

4.1.10.2 Functional Demonstration 514
4.1.10.2.1 Scenario 1 515
4.1.10.2.2 Scenario 2 515
4.1.11 HL7Sender 516
4.1.11.1 Configuration and Testing 516
4.1.11.2 Functional Demonstration 519
4.1.11.2.1 Scenario 1 519
4.1.11.2.2 Scenario 2 520
4.1.12 SocketAdapter 521
4.1.12.1 Configuration 521
4.1.12.2 Functional Demonstration 523
4.1.12.3 Useful Tips 524
4.1.13 SAPB1 524
4.1.13.1 Configuration 525
4.1.13.2 Testing 527
4.1.13.3 Functional Demonstration 527
4.1.13.3.1 Scenario 1 527
4.1.13.4 Sample Input 528
4.1.13.5 Sample Output 528
4.1.14 SalesForce 528
4.1.14.1 Connection Configuration 529
4.1.14.1.1 Call and Addressing Properties 529
4.1.15 AS2Connector 532
4.1.15.1 Configuration and Testing 532
4.1.15.1.1 Connection Configuration 532
4.1.15.1.2 AS2 Configuration 534
4.1.15.2 Input and Output 536
4.1.15.2.1 Input 536
4.1.15.3 Functional Demonstration 538
4.1.15.3.1 Scenario 1 538
4.1.16 EBMSAdapter 539
4.1.16.1 Configuration 539
4.1.16.1.1 Connection Properties 540
4.1.16.1.2 Expert Properties 541
4.1.16.1.3 EBMS Properties 542

4.2 Collaboration 545

4.2.1 Chat 545
4.2.2 C# Chat 546
4.2.3 VB Chat 546
4.2.4 VC Chat 546

4.3 DB 547

4.3.1 DB 547
4.3.1.1 Database Connection Configuration 549
4.3.1.2 Interaction Configurations 552
4.3.1.2.1 SQL Configuration 552
4.3.1.2.2 Stored Procedure Configuration 583
4.3.1.2.3 Monitor Table Configuration 586
4.3.1.2.4 SQL Statement Details Configuration 591
4.3.1.2.5 Editing Query Configuration 596
4.3.1.2.6 Removing Query Configuration 598
4.3.1.2.7 Testing Query Configuration 598
4.3.1.3 Child Queries 599
4.3.1.3.1 Nested Query 599
4.3.1.3.2 Post Processing Query 600
4.3.1.3.3 Failover Query 600
4.3.1.3.4 Child Query Configuration 600
4.3.1.4 Miscellaneous Configurations 603
4.3.1.4.1 Request Level Post Processing Query 603
4.3.1.4.2 Adapter Mode 604
4.3.1.4.3 Output Options 605
4.3.1.4.4 Post Processing Execution 606
4.3.1.4.5 Advanced Configuration 607
4.3.1.4.6 Using Named Configurations 614
4.3.1.5 Input Schema 615
4.3.1.6 Output Schema 616
4.3.1.7 Functional Demonstration 617
4.3.1.7.1 Scenario 1 617
4.3.1.8 Use Case Scenario 618
4.3.1.8.1 Scenario 1 618
4.3.1.8.2 Scenario 2 618
4.3.1.9 Scheduling 619
4.3.1.10 Useful Tips 619
4.3.2 DBProc 620
4.3.2.1 Managed Connection Factory Panel 620
4.3.2.1.1 Connection Properties 621
4.3.2.1.2 Advanced Settings 624
4.3.2.2 Interaction Configurations Panel 624
4.3.2.2.1 Attributes 625
4.3.2.2.2 Using Named Configurations 630
4.3.2.3 Input Schema 631
4.3.2.4 Output Schema 632
4.3.2.6 Functional Demonstration 633
4.3.2.6.1 Scenario 1 633
4.3.2.7 Useful Tips 633
4.3.3 DBQueryOnInput 633

4.3.3.1 Managed Connection Factory Panel 634
4.3.3.1.1 Connection Properties 634
4.3.3.1.2 Advanced Settings 637
4.3.3.2 Interaction Configurations Panel 638
4.3.3.2.1 Attributes 638
4.3.3.3 Input and output 641
4.3.3.3.1 Input 641
4.3.3.3.2 Output 642
4.3.3.4 Functional Demonstration 642
4.3.3.4.1 Scenario 1 642
4.3.3.4.2 Scenario 2 644
4.3.3.5 Useful Tips 645
4.3.4 DBQuery 645
4.3.4.1 Managed Connection Factory Panel 646
4.3.4.1.1 Connection Properties 646
4.3.4.1.2 Advanced Settings 649
4.3.4.2 Interaction Configurations Panel 650
4.3.4.2.1 Attributes 650
4.3.4.2.2 Using Named Configurations 671
4.3.4.3 Input 672
4.3.4.3.1 SELECT 672
4.3.4.3.2 INSERT 673
4.3.4.3.3 UPDATE 673
4.3.4.3.4 DELETE 673
4.3.4.4 Output 674
4.3.4.5 Functional Demonstration 675
4.3.4.5.1 Scenario 1 675
4.3.4.6 Useful Tips 676

4.4 Error 676

4.4.1 Exception Listener 676
4.4.1.1 Configuration and Testing 677
4.4.1.1.1 Attributes 678
4.4.1.1.2 Connection to Enterprise Server 681
4.4.1.2 Functional Demonstration 681
4.4.1.2.1 Scenario 1 681
4.4.1.2.2 Scenario 2 682
4.4.1.3 Use Case Scenario 683
4.4.1.4 Useful Tips 683

4.5 File 683

4.5.1 File Reader 684
4.5.1.1 Interaction Configurations 684

4.5.1.1.1 Attributes 685
4.5.1.1.2 Using Named Configurations 695
4.5.1.2 Input and Output 696
4.5.1.2.1 Input 696
4.5.1.2.2 Output 697
4.5.1.3 Testing the Interaction Configurations 697
4.5.1.4 Functional Demonstration 697
4.5.1.4.1 Scenario 1 697
4.5.1.5 Use Case Scenario 698
4.5.1.6 Useful Tips 698
4.5.2 File Writer 698
4.5.2.1 Interaction Configurations 699
4.5.2.1.1 Attributes 700
4.5.2.1.2 Using Named Configurations 708
4.5.2.2 Sample Scenario 709
4.5.2.3 Input and Output 711
4.5.2.3.1 Input 711
4.5.2.3.2 Output 711
4.5.2.4 Testing the Interaction Configurations 712
4.5.2.5 Functional Demonstration 713
4.5.2.5.1 Scenario 1 713
4.5.2.6 Use Case Scenario 714
4.5.3 File Transmitter 715
4.5.3.1 Configuration and Testing 715
4.5.3.2 Functional Demonstration 718
4.5.3.2.1 Scenario 1 718
4.5.3.2.2 Scenario 2 719
4.5.3.3 Useful Tips 720
4.5.4 File Receiver 721
4.5.4.1 Configuration and Testing 721
4.5.4.1.1 Using Named Configurations 722
4.5.4.2 Functional Demonstration 723
4.5.4.2.1 Scenario 1 723
4.5.4.3 Useful Tips 724

4.6 Flow 725

4.6.1 Aggregator 725
4.6.1.1 Configuration and Testing 725
4.6.1.1.1 Attributes 726
4.6.1.2 Functional Demonstration 734
4.6.1.2.1 Scenario 1 734
4.6.1.2.2 Scenario 2 735
4.6.1.3 Useful Tips 736
4.6.2 CBR 736
4.6.2.1 Configuration and Testing 737

4.6.2.1.1 Schema ... 737
4.6.2.1.2 Namespaces ... 738
4.6.2.1.3 Routing Rules ... 738
4.6.2.1.4 Processor ... 739
4.6.2.2 Functional Demonstration ... 741
4.6.2.2.1 Scenario 1 ... 741
4.6.2.3 Use Case Scenario ... 742
4.6.2.4 Useful Tips ... 742
4.6.3 Distribution Service ... 743
4.6.3.1 Configuration and Testing ... 743
4.6.3.2 Functional Demonstration ... 744
4.6.3.2.1 Scenario 1 ... 744
4.6.4 Join ... 744
4.6.4.1 Configuration and Testing ... 745
4.6.4.1.1 Mappings ... 745
4.6.4.1.2 Transformer factory class Name ... 746
4.6.4.1.3 Use context value from ... 746
4.6.4.1.4 Use properties and headers from ... 746
4.6.4.1.5 Prefer Properties and Headers from ... 747
4.6.4.1.6 Message matching configuration ... 747
4.6.4.2 Testing ... 748
4.6.4.3 Functional Demonstration ... 751
4.6.4.3.1 Scenario 1 ... 751
4.6.4.4 Sample Input ... 751
4.6.4.5 Useful Tips ... 752
4.6.5 Sleep ... 752
4.6.5.1 Configuration ... 752
4.6.5.2 Functional Demonstration ... 753
4.6.5.2.1 Scenario 1 ... 753
4.6.5.3 Useful Tips ... 753
4.6.6 Timer ... 754
4.6.6.1 Configuration and Testing ... 754
4.6.6.1.1 Scheduler Configuration ... 754
4.6.6.1.2 Message Format ... 755
4.6.6.1.3 Message Content ... 755
4.6.6.2 Output Schema ... 755
4.6.6.3 Functional Demonstration ... 756
4.6.6.3.1 Scenario 1 ... 756
4.6.6.4 Useful Tips ... 757
4.6.7 WorkList ... 757
4.6.8 WorkList Manager ... 758
4.6.9 XMLSplitter ... 758
4.6.9.1 Configuration and Testing ... 758
4.6.9.1.1 Attributes ... 759
4.6.9.2 Sample Input and Output ... 763
4.6.9.3 Functional Demonstration ... 764

4.6.9.3.1 Scenario 1 764
4.6.9.3.2 Scenario 2 764
4.6.9.4 Use Case Scenario 766
4.6.9.5 Useful Tips 766
4.6.10 XMLVerification 767
4.6.10.1 Configuration and Testing 767
4.6.10.1.1 Attributes 767
4.6.10.2 Functional Demonstration 768
4.6.10.2.1 Scenario 1 768
4.6.10.3 Useful Tips 769
4.6.11 Cache 769
4.6.11.1 Configuration and Testing 770
4.6.11.1.1 Attributes 770
4.6.11.1.2 Using Named Configurations 772
4.6.11.2 Input and Output 773
4.6.11.2.1 Input 773
4.6.11.2.2 Output 774
4.6.11.3 Functional Demonstration 774
4.6.11.3.1 Scenario 1 774
4.6.11.3.2 Add operation 775
4.6.11.3.2 Update operation 776
4.6.11.3.3 Lookup operation 778
4.6.11.3.4 Delete operation 779
4.6.11.4 Useful Tips 780

4.7 MOMs 781

4.7.1 JMSIn 4.0 781
4.7.1.1 Configuration and Testing 782
4.7.1.1.1 Using Named Configurations 784
4.7.1.1.2 Working with Fiorano MQ HA profiles 786
4.7.1.1.3 Secured Connections to Fiorano MQ (SSL) 786
4.7.1.2 Schema 788
4.7.1.2.1 Input Schema 788
4.7.1.2.2 Output Schema 789
4.7.1.3 Functional Demonstration 790
4.7.1.3.1 Scenario 1 790
4.7.1.4 Useful Tips 790
4.7.2 JMSIn 5.0 791
4.7.2.1 Connection 791
4.7.2.1.1 Connection Configuration 791
4.7.2.1.2 Send Configuration 793
4.7.2.1.3 Producer Configuration 794
4.7.2.1.4 Message Definition 795
4.7.2.1.5 Additional Configuration 797
4.7.2.2 Functional Demonstration 797

4.7.2.2.1 Sample Input 798
4.7.2.2.2 Sample Output 798
4.7.2.3 Useful Tips 798
4.7.3 JMSOut 4.0 799
4.7.3.1 Configuration and Testing 800
4.7.3.1.1 Interaction Configurations 802
4.7.3.1.2 Using Named Configurations 805
4.7.3.2 Input Schema 808
4.7.3.3 Functional Demonstration 808
4.7.3.3.1 Scenario 1 808
4.7.3.4 Useful Tips 809
4.7.4 JMSOut 5.0 809
4.7.4.1 Configuration 810
4.7.4.1.1 Connection Configuration 810
4.7.4.1.2 Receive Configuration 812
4.7.4.1.3 Message Definition Configuration 814
4.7.4.1.4 Additional Configuration 815
4.7.4.2 Functional Demonstration 816
4.7.4.2.1 Scenario 1 816
4.7.4.3 Useful Tips 816
4.7.5 JMSReplier 817
4.7.5.1 Configuration and Testing 818
4.7.5.1.1 Using Named Configurations 818
4.7.5.2 Input Schema 821
4.7.5.3 Output Schema 822
4.7.5.4 Functional Demonstration 823
4.7.5.4.1 Scenario 1 823
4.7.5.5 Use Case Scenario 824
4.7.5.6 Useful Tips 824
4.7.6 JMSRequestor 4.0 825
4.7.6.1 Configuration and Testing 826
4.7.6.1.1 Using Named Configurations 831
4.7.6.2 Functional Demonstration 834
4.7.6.2.1 Scenario 1 834
4.7.7 JMSRequestor 5.0 835
4.7.7.1 Configuration 835
4.7.7.1.1 Connection Configuration 835
4.7.7.1.2 Requestor Configuration 838
4.7.7.1.3 Request Configuration 839
4.7.7.1.4 Producer Configuration 840
4.7.7.1.5 Response Configuration 840
4.7.7.1.6 Error Destination Configuration 841
4.7.7.1.7 Consumer Configuration 842
4.7.7.1.8 Additional Configuration 846
4.7.7.2 Functional Demonstration 846
4.7.7.2.1 Scenario 1 846

4.7.7.3 Useful Tips 847
4.7.8 MQSeriesIn 848
4.7.8.1 Configuration and Testing 848
4.7.8.1.1 Creating queues on IBM WebSphere MQ using WebSphere MQ explorer .. 848
4.7.8.2 Managed Connection Factory 853
4.7.8.2.1 Attributes 853
4.7.8.3 Interaction Configuration 855
4.7.8.3.1 Attributes 855
4.7.8.3.2 Using Named Configurations 867
4.7.8.4 Input and Output 868
4.7.8.4.1 Input 868
4.7.8.4.2 Output 873
4.7.8.5 Functional Demonstration 874
4.7.8.5.1 Scenario 1 874
4.7.8.5.2 Scenario 2 875
4.7.8.5.3 Scenario 3 877
4.7.8.6 Useful Tips 878
4.7.9 MQSeriesOut 878
4.7.9.1 Configuration and Testing 878
4.7.9.1.1 Creating Queues on IBM WebSphere MQ using WebSphere MQ Explorer.. 878
4.7.9.2 Managed Connection Factory 883
4.7.9.2.1 Attributes 883
4.7.9.3 Interaction Configuration 886
4.7.9.3.1 Attributes 886
4.7.9.3.2 Using Named Configurations 894
4.7.9.4 Input and output 897
4.7.9.4.1 Input 897
4.7.9.4.2 Output 897
4.7.9.5 Functional Demonstration 903
4.7.9.5.1 Scenario 1 903
4.7.9.5.2 Scenario 2 904
4.7.9.5.3 Scenario 3 905
4.7.9.5.4 Scenario 4 906
4.7.9.5.5 Scenario 5 907
4.7.9.5.6 Scenario 6 907
4.7.9.6 Useful Tips 908
4.7.10 MSMQ Receiver 908
4.7.10.1 Configuration and Testing 908
4.7.10.1.1 Using Named Configurations 909
4.7.10.2 Input and Output 911
4.7.10.2.1 Input Schema 911
4.7.10.2.1 Output Schema 912
4.7.10.3 Functional Demonstration 912
4.7.10.3.1 Scenario 1 912
4.7.10.4 Use case scenario 913
4.7.10.5 Useful Tips 913

4.7.11 MSMQ Sender 914
4.7.11.1 Configuration and Testing 914
4.7.11.1.1 Using Named Configurations 914
4.7.11.2 Input and Output 917
4.7.11.2.1 Input Schema 917
4.7.11.2.2 Output Schema 917
4.7.11.3 Functional Demonstration 918
4.7.11.3.1 Scenario 1 918
4.7.11.4 Use case scenario 919
4.7.11.5 Useful Tips 919
4.7.12 TibcoRVIn 919
4.7.13 TibcoRVOut 920

4.8 Performance 921

4.8.1 Receiver 921
4.8.1.1 Configuration and Testing 921
4.8.1.2 Functional Demonstration 922
4.8.1.2.1 Scenario 1 922
4.8.1.3 Useful Tips 923
4.8.2 Sender 923
4.8.2.1 Configuration and Testing 923
4.8.2.2 Functional Demonstration 925
4.8.2.2.1 Scenario 1 925
4.8.2.3 Useful Tips 925

4.9 Samples 926

4.9.1 Binary File Reader 926
4.9.2 CRM 926
4.9.3 Composite BC 926
4.9.4 LDAP Lookup 927
4.9.5 LDAP Authenticator 927
4.9.6 Market Prices GUI 927
4.9.7 Prices 928
4.9.7.1 Configuration and Testing 928
4.9.7.2 Input and Output 928
4.9.7.2.1Input Schema 928
4.9.7.2.2 Output Schema 928
4.9.7.3 Use Case scenario 929
4.9.8 RFQ Manager 929
4.9.9 Trade Bus 929
4.9.10 ERP 930

4.10 Script 930

4.10.1 BeanShell Script 930
4.10.1.1 Interaction Configuration 931
Transport Configurations 933
4.10.1.2 Sample Input and Output 934
4.10.1.3 Functional Demonstration 934
4.10.1.3.1 Scenario 1 934
4.10.1.3.2 Scenario 2 935
4.10.1.4 Useful Tips 935
4.10.2 Groovy Scrip 935
4.10.2.1 Configuration and Testing 935
4.10.2.1.1 Interaction Configuration 935
4.10.2.1.2 Sample Input and Output 936
4.10.2.2 Functional Demonstration 937
4.10.2.2.1 Scenario 1 937
4.10.2.2.2 Scenario 2 937
4.10.2.3 Useful Tips 937
4.10.3 Java Script 938
4.10.3.1 Configuration and Testing 938
4.10.3.1.1 Interaction Configuration 938
4.10.3.1.2 Sample Input and Output 938
4.10.3.2 Functional Demonstration 939
4.10.3.2.1 Scenario 1 939
4.10.3.3 Useful Tips 940
4.10.4 Perl Script 940
4.10.4.1 Configuration and Testing 940
4.10.4.1.1 Interaction Configuration 940
4.10.4.1.2 Sample Input and Output 940
4.10.4.2 Using Named Configurations 941
4.10.4.3.1 Scenario 1 941
4.10.4.3.2 Scenario 2 942
4.10.4.4 Useful Tips 942
4.10.5 Python Script 943
4.10.5.1 Configuration and Testing 943
4.10.5.1.1 Interaction Configuration 943
4.10.5.1.2 Sample Input and Output 943
4.10.5.2 Functional Demonstration 944
4.10.5.2.1 Scenario 1 944
4.10.5.2.2 Scenario 2 944
4.10.5.3 Useful Tips 945

4.11 Transformation 945

4.11.1 EDI 2 XML 945
4.11.1.1 Configuration and Testing 945

4.11.1.2 Functional Demonstration 946
4.11.1.2.1 Scenario 1 946
4.11.2 HL7 Reader 947
4.11.2.1 Configuration and Testing 947
4.11.2.1.1 Interaction Configuration 947
4.11.2.1.2 Sample Input and Output 948
4.11.2.2 Functional Demonstration 949
4.11.2.2.1 Scenario 1 949
4.11.2.3 Useful Tips 950
4.11.3 HL7 Writer 950
4.11.3.1 Configuration and Testing 951
4.11.3.1.1 Interaction Configuration 951
4.11.3.1.2 Sample Input and Output 951
4.11.3.2 Functional Demonstration 952
4.11.3.2.1 Scenario 1 952
4.11.3.3 Useful Tips 952
4.11.4 Text2XML 953
4.11.4.1 Configuration and Testing 953
4.11.4.1.1 Managed Connection Factory 953
4.11.4.1.2 Interaction Configuration 954
4.11.4.2 Sample Input and Output 954
4.11.4.3 Functional Demonstration 956
4.11.4.3.1 Scenario 1 956
4.11.4.4 Use Case Scenario 957
4.11.4.5 Useful Tips 957
4.11.5 XML 2 EDI 957
4.11.5.1 Configuration and Testing 957
4.11.5.2 Functional Demonstration 959
4.11.5.2.1 Scenario 1 959
4.11.6 XML2PDF 960
4.11.6.1 Configuration and Testing 960
4.11.6.1.1 Sample Input and Output 961
4.11.7 XML 2 Text 965
4.11.7.1 Configuration and Testing 965
4.11.7.1.1 Managed Connection Factory 965
4.11.7.1.2 Interaction Configuration 966
4.11.7.2 Sample Input and Output 966
4.11.7.2.1 Sample Input 967
4.11.7.2.2 Sample Output 968
4.11.7.3 Functional Demonstration 968
4.11.7.3.1 Scenario 1 968
4.11.7.4 Useful Tips 969
4.11.8 XSLT 969
4.11.8.1 Interaction Configurations 969
4.11.8.1.1 Attributes 970
4.11.8.2 Functional Demonstration 980

4.11.8.2.1 Scenario 1 980
4.11.8.3 Use Case Scenario 982
4.11.8.4 Useful Tips 982
4.11.9 JSONConverter 982
4.11.9.1 Configuration 982
4.11.9.2 Functional Demonstration 984
4.11.9.2.1 Scenario1: JSON to XML 984
4.11.9.2.2 Scenario 2: XML to JSON 985
4.11.9.3 Useful Tips 986

4.12 Util 986

4.12.1 Compression 986
4.12.2 Decompression 986
4.12.3 Decryption 987
4.12.3.1 Configuration and Testing 987
4.12.3.1.1 Decryption Configuration 988
4.12.3.1.2 Binary Output Required 989
4.12.3.1.3 Binary Input Required 989
4.12.3.1.3 Monitoring configuration 990
4.12.3.1.4 Using Named Configurations 990
4.12.3.2 Functional Demonstration 991
4.12.3.2.1 Scenario 1 991
4.12.3.3 Use Case Scenario 992
4.12.3.4 Useful Tips 992
4.12.4 DiskUsageMonitorService 992
4.12.4.1 Configuration and Testing 993
4.12.4.2 Functional Demonstration 993
4.12.4.2.1 Scenario 1 993
4.12.4.3 Useful Tips 994
4.12.5 Display 994
4.12.5.1 Configuration and Testing 994
4.12.5.2 Functional Demonstration 994
4.12.5.2.1 Scenario 1 994
4.12.5.2.2 Scenario 2 995
4.12.5.3 Use Case Scenario 996
4.12.6 Encryption 996
4.12.6.1 Configuration and Testing 996
4.12.6.1.1 Encryption Configuration 997
4.12.6.1.2 Binary Output Required 999
4.12.6.1.3 Monitoring configuration 1000
4.12.6.1.4 Using Named Configurations 1000
4.12.6.3 Functional Demonstration 1002
4.12.6.3.1 Scenario 1 1002
4.12.6.4 Use Case Scenario 1003
4.12.7 Feeder 1004

4.12.7.1 Configuration and Testing 1004
4.12.7.2 Input and Output 1005
4.12.7.2.1 Input Schema 1005
4.12.7.2.2 Output Schema 1005
4.12.7.3 Functional Demonstration 1005
4.12.7.3.1 Scenario 1 1005
4.12.7.4 Use Case Scenario 1007
4.12.7.5 Useful Tips 1007
4.12.8 PrintPDF 1009
4.12.8.1 Configuration and Testing 1009
4.12.8.1.1 Error Handling Configuration 1009
4.12.8.1.2 Using Named Configurations 1010
4.12.8.1.3 Sample Input and Output 1010
4.12.8.2 Recommendations 1011
4.12.8.3 Limitations 1012
4.12.9 NEFTConverter 1012
4.12.9.1 Configuration and Testing 1012
4.12.9.1.1 NEFTConverter CPS 1013
4.12.9.2 Functional Demonstration 1014
4.12.9.2.1 Scenario 1 1014
4.12.9.2.2 Scenario 2 1015

4.13 Web 1017

4.13.1 HTTPAdapter 1017
4.13.1.1 Configuration and Testing 1018
4.13.1.1.1 Managed Connection Factory 1018
4.13.1.2 Interaction Configuration 1020
4.13.1.2.1 Attributes 1020
4.13.1.3 Input and Output 1029
4.13.1.3.1 Input 1029
4.13.1.3.2 Output 1036
4.13.1.4 SSL Setup 1038
4.13.1.5 Using Named Configurations 1039
4.13.1.6 Functional Demonstration 1040
4.13.1.6.1 Scenario 1 1040
4.13.1.7 Use Case Scenario 1042
4.13.1.8 Useful Tips 1042
4.13.2 HTTPReceive 1043
4.13.2.1 Configuration and Testing 1043
4.13.2.1.1 General Configuration 1044
4.13.2.1.2 Multithread Configuration 1054
4.13.2.1.3 Using Named Configurations 1055
4.13.2.2 Input and Output 1056
4.13.2.2.1 Input 1056
4.13.2.2.2 Output 1058

4.13.2.4 Functional Demonstration 1061
4.13.2.4.1 Scenario 1 1061
4.13.2.5 Use Case Scenario 1062
4.13.2.6Useful Tips 1062
4.13.3 HTTP Stub 1063
4.13.3.1 Configuration and Testing 1063
4.13.3.1.1 Deployment Configuration 1064
4.13.3.1.2 FES Connection Configuration 1064
4.13.3.1.3 Execution Configuration 1065
4.13.3.2 Input and output 1070
4.13.3.2.1 Input 1070
4.13.3.2.2 Output 1070
4.13.3.3 Functional Demonstration 1072
4.13.3.3.1 Scenario 1 1072
4.13.3.3.2 Scenario 2 1073
4.13.3.4 Use Case Scenario 1074
4.13.3.5 Useful Tips 1074
4.13.4 SimpleHTTP 1075
4.13.4.1 Configuration and Testing 1075
4.13.4.1.1 Interaction Configurations 1075
4.13.4.2 Functional Demonstration 1077
4.13.4.2.1 Scenario 1 1077
4.13.4.2.2 Scenario 2 1078
4.13.4.3 Useful Tips 1079
4.13.5 RESTStub 1080
4.13.5.1 Configuration 1080
4.13.5.1.1 WADL Definition 1081
ID 1084
Path 1085
QueryType 1085
Name 1086
ID 1086
Status 1087
Representations 1088
Element 1088
MediaType 1089
Profile 1090
4.13.5.1.2 Transport Security 1093
4.13.5.1.3 Miscellaneous Configuration 1097
4.13.5.2 Port Generation 1098
4.13.5.3 Schema Generation 1098
4.13.5.3.1 Request Schema 1099
4.13.5.3.2 Response Schema 1100
4.13.5.4 Functional Demonstration 1101
4.13.5.5 Useful Tips 1102
4.13.6 RESTConsumer 1103

4.13.6.1. Configuration 1103
4.13.6.1.1. WADL Definition 1104
ID 1106
Path 1107
QueryType 1107
Name 1108
ID 1108
Status 1109
Representations 1110
Element 1110
MediaType 1111
Profile 1111
4.13.6.1.2 Transport Security 1115
4.13.6.1.3 Additional Configuration 1117
4.13.6.2 Port Generation 1118
4.13.6.3 Schema Generation 1119
4.13.6.3.1 Request Schema 1119
4.13.6.3.2 Response Schema 1121
4.13.6.4 Functional Demonstration 1123
4.13.6.5 Useful Tips 1124

4.14 WebService 1125

4.14.1 WSStub 1125
4.14.1.1 Configuration 1125
4.14.1.1.1 WS Definition 1126
4.14.1.1.2 Basic 1128
4.14.1.1.3 Advanced 1132
Input tab 1134
Output tab 1134
Faults tab 1134
SOAP Binding 1135
Binding Operations 1135
Body tab 1137
Headers Tab 1137
MIME MultiPartRelated Binding 1139
Operation tab 1140
Input tab 1141
Output tab 1141
Faults tab 1142
4.14.1.1.4 WSDL 1144
4.14.1.1.5 Common Panels 1145
4.14.1.1.6 WS Standards 1146
Request Security 1146
Security actions 1146
Response Security 1149

4.14.1.1.7 Transport Security 1154
4.14.1.1.8 Miscellaneous Configuration 1159
4.14.1.1.9 Additional Configuration 1160
4.14.1.1.10 Attachment Configuration 1161
4.14.1.1.11 Encrypt/Decrypt Configuration 1163
4.14.1.1.12 Port Generation 1163
4.14.1.2 Functional Demonstration 1165
4.14.1.3 Useful Tips 1167
4.14.2 Web Service Consumer (4.0) 1169
4.14.2.1 Connection Configuration 1170
4.14.2.1.1 Using Named Configurations 1178
4.14.2.2 Input and Output 1179
4.14.2.2.1 Input Schema 1179
4.14.2.2.2 Output Schema 1180
4.14.2.3 Accessing Share Point Web Services 1181
4.14.2.4 Functional Demonstration 1183
4.14.2.4.1 Scenario 1 1183
4.14.2.4.2 Scenario 2 1183
4.14.2.4.3 Scenario 3 1185
4.14.2.5 Use Case Scenario 1187
4.14.2.6 Useful Tips 1187
4.14.3 Web Service Consumer (5.0) 1188
4.14.3.1 Configuration and Testing 1188
4.14.3.1.1 General 1188
4.14.3.1.2 Authentication Type 1191
4.14.3.1.3 Using Named Configurations 1192
4.14.3.2 Input and Output 1192
4.14.3.2.1 Input Schema 1192
4.14.3.2.2 Output Schema 1194
4.14.3.3 Functional Demonstration 1195
4.14.3.3.1 Scenario 1 1195
4.14.3.3.2 Scenario 2 1196

4.15 Finance 1197

4.15.1 FIXAcceptor 1197
4.15.1.1 Configuration 1197
4.15.1.1.1 Common Session Configuration 1197
4.15.1.1.2 Dictionary 1198
4.15.1.1.3 Session Configuration 1199
4.15.1.2 Properties 1202
4.15.1.2.1 Session Properties 1202
4.15.1.2.2 Validation Properties 1204
4.15.1.2.3 Socket Properties 1204
4.15.1.2.4 Storage Properties 1206
4.15.1.2.5 Miscellaneous Properties 1206

4.15.1.3 Functional Demonstration 1206
4.15.1.3.1 Scenario 1 1207
4.15.1.3.2 Scenario 2 1207
4.15.1.4 Useful Tips 1208
4.15.2 FIXInitiator 1208
4.15.2.1 Configuration 1208
4.15.2.1.1 Common Session Configuration 1208
4.15.2.1.2 Dictionary 1210
4.15.2.1.3 Session Configuration 1210
4.15.2.2 Properties 1215
4.15.2.2.1 Session Properties 1215
4.15.2.2.2 Validation Properties 1216
4.15.2.2.3 Socket Properties 1217
4.15.2.2.4 Storage Properties 1219
4.15.2.2.5 Miscellaneous Properties 1219
4.15.2.3 Functional Demonstration 1219
4.15.2.3.1 Scenario 1 1219
4.15.2.3.2 Scenario 2 1220
4.15.2.4 Useful Tips 1221

Chapter 5: Event Processes 1223

5.1 What are Event Processes? 1223
5.2 Creating Event Processes 1223
5.2.1 Creating a New Event Process 1224
5.3 Configure Event Processes 1226
5.3.1 Configuring Components through Custom Property Sheet 1226
5.3.2 Configuring Common Component Properties 1226
5.3.3 Adding Additional Jars/Libraries to Components 1227
5.3.4 Setting up Component Port Properties 1227
5.3.5 Defining Data Transformation 1228
5.3.6 Defining Exception Flows 1228
5.3.6.1 Using the Exception Listener Service Component: 1229
5.3.7 Using the Error Ports View 1230
5.3.8 Document Tracking 1230
5.3.8.1 Configuring Document Tracking 1231
5.3.8.2 Configuring Specific Database 1231
5.3.8.3 Database Table Structure 1231
5.3.8.4 Structure of IMAGE/BLOB field 1233
5.3.9 Message Selector on Route 1233
5.3.9.1 Defining Message Selector on Route 1234
5.3.10 Setting Alerts and Notification 1236
5.3.11 Configuring the Application Context 1238
5.4 Using External Event Processes 1243
5.4.1 Importing Remote Service Instance 1243
5.4.2 Using External Event Processes 1245

5.5 Debugging Event Processes 1246
5.5.1 Viewing Component Logs 1247
5.5.2 Setting Event Interceptors 1250
5.5.2.1 Setting an Event Interceptor on a Route 1250
5.5.2.2 Viewing Intercepted Messages 1251
5.5.2.3 Viewing Content of an Intercepted Message 1252
5.5.2.4 Viewing Component Launch and Kill Time 1253
5.5.2.5 Viewing Component Pending (Queued) Messages 1254
5.6 Modifying Event Processes 1255
5.6.1 Replacing a Component at Runtime 1255
5.6.2 Adding a New Component Instance at Runtime 1256
5.7 Monitoring Event Processes 1257
5.7.1 Tracking Events within Processes 1257
5.7.2 Defining a Workflow 1257
5.7.2.1 Starting a Workflow 1257
5.7.2.2 Viewing Tracked Documents of a Workflow 1258
5.7.2.3 Tracking Documents across Workflows 1258
5.7.3 Setting up Database to store Tracked Documents 1259
5.7.4 Re-Injecting Tracked Documents 1259
5.7.4.1 Re-Injection Document Structure 1259
5.7.4.2 Re-Injection of Tracked Documents 1260
5.7.4.3 Re-Injected Workflow 1262
5.8 Import and Export Event Processes 1262
5.8.1 Importing Event Processes 1263
5.8.2 Exporting an Event Process 1265
5.8.3 Exporting Multiple Applications 1266
5.9 Deploying Event Processes 1267
5.9.1 Connectivity and Resource Check 1268
5.9.2 Enabling/Disabling the Component Cache 1270
5.10 Launching Components and Event Processes from Studio 1272
5.10.1 Launching an Event Process 1273
5.10.2 Stopping an Event Process 1274
5.10.3 Synchronizing Event Processes 1274
5.10.4 Launching and Stopping Individual Components 1274
5.11 The Event Process Command Line Interface 1276
5.11.1 List of Ant Tasks provided by command Line Interface 1277
5.11.2 Launching an Event Process from Command Line 1278
5.11.3 Launching Components from Command Line 1279
5.11.4 Executing Components Manually 1280
5.12 Best Practices in Deployment 1283
5.12.1 Creating Port Bindings Between Components in Different Event Processes 1284
5.13 Testing Event Processes 1285
5.14 Sample Event Processes 1285
5.14.1 Bond Trading 1285
5.14.2 Database Replication 1287
5.14.3 EAI Demo 1288

5.14.4 Order Entry 1289
5.14.5 Portal Integration 1291
5.14.6 Purchasing System 1292
5.14.7 Retail Television 1296
5.14.8 Revenue Control Packet 1297
5.14.9 Simple Chat 1300
5.14.10 WorkList Sample 1301

Chapter 6: High Availability 1303

6.1 ESB Server High Availability 1303
6.2 Peer Server High Availability 1304
6.3 Fiorano Replicated High Availability Working 1305
6.3.1 HA Locking Mechanism 1306
6.3.2 Server States 1306
6.3.3 Configuring Fiorano SOA High Availability Servers 1308
6.3.4 Configuration Steps 1309
6.3.4.1 Setting up the LockFile 1309
6.3.4.2 Configuring the FES HA Profile 1311
6.3.5 Verifying HA Setup 1319
6.3.6 Shutting down the HA Server 1320
6.4 Fiorano High Availability Working In Shared Mode 1321
6.4.1 Shared HA Precondition 1321
6.4.2 Server States 1321
6.4.3 Configuring Fiorano SOA High Availability Servers 1322
6.4.4 Configuration Steps 1323
6.4.4.1 Setting up the Lock File 1323
6.4.4.2 Setting up the shared database 1324
6.4.4.3 Configuring the FES/FPS HA Profile 1329
6.4.4.4 Changing the location of log files 1330
6.4.5 Verifying HA Setup 1332
6.4.6 Shutting down the HA Server 1332
6.4.7 Troubleshooting Steps 1333
6.5 Limitations of Fiorano SOA High Availability 1333
6.6 Reference Matrix – HA Profile 1334
6.7 Determining Server State 1335

Chapter 7: Scalability, Load Balancing and Memory Optimization 1336

7.1 Server-Level Load Balancing 1336
7.1.1 Scaling by adding more peers to the network 1336
7.1.2 Scaling by distributing load across multiple service instances 1337
7.1.3 An example of Load Balancing 1337
7.2 Thread Count of Components 1338

7.3 Scalability 1339
7.3.1 Transparent Resource Addition 1339
7.3.2 Dynamic Change Support 1339
7.3.3 Parallel Data Flow 1339
7.4 Memory Optimization 1340
7.4.1 JVM Parameters 1340
7.4.2 Separate Machines for Servers 1341
7.4.3 Distribute Components 1341
7.4.4 Inter-Connect Flows 1342
7.4.5 Size of Event Flows 1342
7.4.6 Size of Messages 1342
7.4.7 DB Adapter Tuning 1343
7.5 Component Memory Tuning 1343
7.5.1 Tuning Memory for Service Components 1343
7.5.1.1 Know about Heap sizes 1343
7.5.1.2 Default Heap size 1344
7.5.1.3 Setting Heap sizes 1345
7.5.1.4 Garbage Collection 1346
7.5.1.5 Monitoring Component JVM Statistics 1346
7.5.1.6 Tuning the memory settings 1347
7.5.2 Recommendations 1351
7.5.2.1 Component Overloading 1351
7.5.3 Components 1352
7.5.3.1 File Reader 1352
7.5.3.2 File Writer 1354
7.5.3.3 XSLT 1356
7.5.3.4 CBR 1356
7.5.3.5 Aggregator 1359
7.5.3.6 Distribution 1362
7.5.4 Walkthrough 1364
7.5.4.1Application 1364
7.5.4.2 Tuning Process 1365
7.6 Memory Management of Fiorano Peer Server 1367
7.6.1 Physical Machine Configuration 1368
7.6.2 Java Virtual Machine 1368
7.6.3 Machine Setup 1368
7.6.4 Processing Message Size 1368
7.6.5 Peer Server Load 1369
7.6.6 Recommendations 1369
7.6.7 Interpreting and Applying Recommendations 1371
7.6.8 Handling Memory Problems 1379
7.6.7 Java Heap Space 1380
7.6.7.1 Debugging 1380
7.6.8 Permgen Space 1380
7.6.8.1 Debugging 1380
7.6.9 Requested Array size exceeds VM Limit 1380

7.6.10 Swap Space 1381
7.6.11 Unable to Create New Native Thread 1381
7.6.12 Enabling GC logging 1381
7.7 In-Memory Execution and Load Balancing of Components Across Peer Servers 1382
7.7.1 Separate Process 1382
7.7.2 In-memory 1382

Chapter 8: Security 1384

8.1 Authentication 1384
8.2 Authorization 1385
8.3. Password Rules 1386
8.3.1 Password Strength 1386
8.3.2 Password Validation 1386
8.3.3 Custom Password Rule Implementation 1387
8.4 Deployment Manager 1389
8.5 Labels 1389
8.6 Rules 1390
8.7 Changing Security Database Implementation 1392
8.7.1 Security Related MBeans 1392
8.7.2 Modifying ACLManager Implementation 1393
8.7.3 Modifying Principal Manager Implementation 1394
8.7.4 Editing Destination Level Security Through ACL's 1395
8.7.5 RDBMS Realm 1396
8.7.5.1 Setting up 1396
8.7.5.2 Additional Configuration 1397
8.7.5.3 Sample Configurations 1397
8.7.5.4 Verifying 1399
8.7.6 LDAP Security Realm 1399
8.7.6.1 Sample Configuration – Netscape Directory Server 1400
8.7.6.2 Sample Configuration – ApacheDS1.5.4 1401
8.7.7 XML Security Realm 1404
8.7.7.1 Configuring Principal Manager 1404
8.7.7.2 Configuring ACL Manager 1405
8.7.7.3 Sample xml files 1405
8.7.7.4 Group.xml 1406
8.7.7.5 acl.xml 1406
8.8 Event Process Security 1407
8.8.1 How ACLs Work 1407
8.8.1.1 Application Level Permission 1407
8.8.1.2 Global/System Level Permission(s) 1408
8.8.3 Default Allowed Set Of Users 1410
8.8.4 Changing Default Permissions 1411
8.8.4.1 Changing Global Permissions 1411
8.8.4.2 Changing Application Level Permissions 1413
8.8.5 Principal Store Synchronization 1414

Chapter 9: Fiorano Mapper 1417

9.1 Key Features of Fiorano Mapper 1417
9.2 Fiorano Mapper Environment 1417
9.2.1 Menu Bar 1419
9.2.1.1 File 1419
9.2.1.2 Edit 1419
9.2.1.3 Structure 1419
9.2.1.4 View 1419
9.2.1.5 AutoMap 1420
9.2.1.6 Tools 1420
9.2.1.7 Help 1420
9.2.2 Toolbar 1420
9.2.3 MapView 1422
9.2.3.1 Input Structure Panel 1423
9.2.3.2 Lines Panel 1423
9.2.3.3 Output Structure Panel 1424
9.2.3.4 Details Pane 1424
9.2.4 MetaData View 1427
9.2.4.1 Error Messages Panel 1427
9.3 Working with Input and Output Structures 1428
9.3.1 Loading the Input Structure 1428
9.3.1.1 Loading an Existing XML Input Structure 1428
9.3.1.2 Loading a New Input XML Structure 1431
9.3.2 Viewing Source of Input Structure 1432
9.3.3 Clearing the Input Structure 1432
9.3.4 Loading the Output Structure 1433
9.3.4.1 Loading an XML Output Structure 1434
9.3.4.2 Loading a CSV Output Structure 1435
9.3.5 Viewing the Output Structure Source 1438
9.3.6 Clearing the Output Structure 1438
9.4 Working with the Visual Expression Builder 1439
9.4.1 Function Palette 1439
9.4.1.1 Arithmetic Functions 1439
9.4.1.2 Math Functions 1442
9.4.1.3 String Functions 1444
9.4.1.4 Control Function 1447
9.4.1.5 Conversion Functions 1448
9.4.1.6 Advanced Functions 1450
9.4.1.7 Date-Time Functions 1452
9.4.1.8 SQL Functions 1459
9.4.1.9 NodeSet Functions 1463
9.4.1.10 Boolean functions 1467
9.4.1.11 Lookup functions 1478
9.4.1.12 JMS Message Functions 1480
9.4.1.13 User Defined functions 1481

9.4.2 Funclet Easel 1483
9.4.2.1 Source Node 1483
9.4.2.2 Destination Node 1483
9.5 Creating Mappings 1487
9.5.1 Understanding Types of Nodes 1487
9.5.2 Types of Mappings 1490
9.5.2.1 Name-to-Name Mapping 1490
9.5.2.2 For-Each Mapping 1491
9.5.3 Duplicating a For-Each Mapping 1492
9.5.4 Linking Nodes to Define Mappings 1494
9.5.4.1 Using the Automatic Mapping option to Define Mappings 1494
9.5.4.2 Using the Visual Expression Builder to Define Mappings 1495
9.5.5 Mapping XML Formats 1498
9.5.6 Mapping XML Formats to CSV Files 1498
9.5.7 Mapping XML Formats to RDBMS Queries 1498
9.5.7.1 Mapping XML Formats to RDBMS-Insert Queries 1498
9.5.7.2 Mapping XML Formats to RDBMS-Update Queries 1499
9.5.7.3 Mapping XML Formats to RDBMS-Delete Queries 1501
9.6 Adding User XSLT 1502
9.7 Testing the Transformation 1504
9.8 Managing Mappings 1507
9.8.1 Exporting Mappings to a File 1507
9.8.2 Importing Project from the File 1508
9.8.3 Validating All Mappings 1508
9.8.4 Displaying All Mappings 1508
9.8.5 Removing Mappings for a Node 1508
9.8.6 Copying functions in a Mapping 1509
9.8.7 Clearing All Mappings 1509
9.8.8 Clearing Data 1509
9.8.9 Modifying the RDBMS Output Structure Settings 1510
9.8.10 Configuring Mapper Settings 1510
9.8.11 Managing XSLT Properties 1511
9.9 Customizing the Mapper User Interface 1512
9.10 Fiorano Mapper –Custom Funclets 1514
9.10.1 Creating Custom Funclets 1514
9.10.1.1 Integrating JavaScript functions 1514
9.10.1.2 Integrating Java functions 1518
9.10.2 Editing Existing User Defined Funclets 1520
9.10.3 Updating UserDefined funclets 1521
9.10.4 Removing User Defined Funclets. 1521
9.10.4.1 Removing User defined funclets from Fiorano Mapper 1521
9.10.4.2 Deleting entire extension 1522

Chapter 10: Common Components Configurations . 1524

10.1 Component Instance Properties 1524

10.1.1 Properties 1524
10.1.2 Deployment 1526
10.1.3 Execution 1528
10.1.4 Log Module Instances 1530
10.1.5 Runtime Arguments 1530
10.2 Port Properties 1531
10.2.1 Input Port Properties 1531
10.2.2 JMS Destination 1531
10.2.2.1 Destination Encryption 1532
10.2.3 Messaging 1533
10.3 Output Port Properties 1536
10.3.1 JMS Destination 1536
10.3.2 Messaging 1536
10.3.3 Preventing message loss 1537
10.3.4 Components with implicitly defined JMS messaging properties 1537
10.4 Managed Connection Factory 1538
10.4.1 SSL Security 1541
10.5 Interaction Configurations 1543
10.5.1 Scheduler Configurations 1545
10.5.2 Advanced Configuration 1546
10.6 Transport Configurations 1547
10.7 Error Handling 1548
10.7.1 Connection Error 1549
10.7.2 JMS Error 1551
Remedial Actions 1551
10.7.3 Response Generation Error 1552
Remedial Actions 1552
10.7.4 Request Processing Error 1553
10.7.5 Invalid Request Error 1554
EDBC Components 1554
BC Components 1555
10.7.6 Retry Configuration 1555
10.8 Schema Editor 1556
10.9 Schema Repository 1561
10.10 XPath Editor 1563
10.11 Named Configurations in CPS 1566
10.11.1 Load Named Configuration 1566
10.11.2 Save Named Configuration 1567

Chapter 11: Monitoring 1569

11.1 Configuring Transports 1569
11.1.1 SMTP Transport 1570
11.1.2 JMS Transport 1571
11.2 Managing Alerts 1572
11.2.1 SMTP Alert 1572

11.2.2 Creating New SMTP Alerts 1573
11.2.3 JMS Alert 1574
11.2.3.1 Adding a New JMS alert 1576
11.2.3.2 Editing/Removing Alert 1576
11.3 Managing Policies 1577
11.3.1 Managing Backlog Policies 1577
11.3.2 Creating Policy 1579
11.3.3 Managing Low Memory Policies 1580
11.3.3.1 Creating Low Memory Policy 1582
11.3.4 Managing Policy Executions 1583
11.3.5 Editing/Removing Policy 1584

Chapter 12: Audit Management 1585

12.1 Audit Policies 1586
12.1.1 Enabling Audit Policies 1587
12.1.2 Disabling Audit Policies 1588
12.1.3 Audit Policy Store Synchronization 1588
12.2 Handling Peer Server Audit Events 1589
12.3 Audit Security Permissions 1589
12.4 Description of Audit Events 1590
12.4.1. Authentication Event 1590
12.4.2. Authorization Event 1590
12.4.3. Security Database Modification Event 1591
12.4.4. Event Process Repository Modification Event 1592
12.4.5. Service Repository Modification Event 1593
12.4.6. Event Process Life Cycle Event 1594
12.4.7. Component Life Cycle Event 1595
12.4.8. Principal Store Synchronization Event 1596
12.4.9. Alert Repository Modification Event 1597
12.5 Audit Event Viewer 1598
12.5.1 Remembering Search Preferences 1599
12.5.2 Using Saved Search Preferences 1600

Chapter 13: SOA Best Practices 1601

13.1 Development Model 1601
13.1.1 Event Process Development 1601
13.1.2 Service Component Development 1602
13.1.3 Error Handling 1603
13.1.4 Explicit Transformations 1603
13.1.5 Version Control Integration 1604
13.2 Testing 1604
13.2.1 Component Level Testing 1604
13.2.2 Process Level Testing 1604

13.3 Deployment Model 1604
13.3.1 Server Deployment 1604
13.3.2 Event Process and Component Deployment 1605
13.4 Performance Tuning and Memory Optimization 1605
13.4.1 Servers 1605
13.4.2 Service Components 1605
13.5 Troubleshooting 1607

Chapter 14: Named Configuration 1608

14.1 Advantages 1608
14.1.1 Faster Event Process Composition 1608
14.1.2 Easy Change Management 1609
14.1.3 Easy Resource Management 1609
14.1.4 Smaller Event Process Size 1609
14.2 Configuration Types 1609
14.2.1 Service Instance Configuration 1609
14.2.2 Resource Configuration 1610
14.2.3 Route Configuration 1610
14.2.4 Selectors Configuration 1610
14.2.5 Transformation Configuration (Route / App-Context / XSLT Transformations) .. 1610
14.2.6 Port Configuration 1611
14.2.7 Workflow Configuration 1611
14.2.8 Runtime Argument Configuration 1611
14.2.9 Connection Factory Configuration 1612
14.3 Defining & Using Configurations 1612
14.3.1 Creating / Modifying Configurations 1612
14.3.1.1 Configuration Repository view in eStudio 1612
14.3.1.2 'Save As Named Configuration' in component CPS 1613
14.3.2 Deleting / Renaming Configurations 1613
14.3.3 Creating New Configurations from Existing Configurations 1614
14.3.4 Using Configurations 1614
14.3.4.1 Route via the Properties Panel 1614
14.3.4.2 Route via the Configuration Repository View 1615
14.3.4.3 Route using Transformation Configurations 1616
14.4 Finding and Clearing Usages 1616
14.5 Validating Configurations 1616
14.5.1 Error Handling 1617
14.6 Resolving Named Configurations 1617
14.6.1 Service Instances 1617
14.6.2 Port Instances 1617
14.6.3 Routes 1617
14.7 Security 1618

Chapter 15: Fiorano BPM Tool 1619

15.1 Enabling BPM Tool 1619
15.2 Configuring DB 1620
15.3 Resource Service 1620
15.4 BPM Editor 1620

Chapter 16: Frequently Asked Questions 1621

Chapter 17: Trouble Shooting 1624

Hostname Resolution Issue 1624
Peer on Multi-homed machine 1624
Firewall Issues 1624
eStudio login issue 1625
File encoding issue 1625
Client Connections in Peer 1625
File Handles in Linux 1625
Problems When Non-ascii Chars Used in build.properties 1626
JRE Problems 1626
HA Troubleshooting 1626

Index 1629

Chapter 1: Introduction

The Fiorano SOA® Platform User Guide has been developed for all users including advanced users who are familiar to using API documentation and runtime libraries to create, customize, test and deploy business components after testing their behavior. This guide is also designed to assist Enterprise and Peer Server Administrators in managing the entire Fiorano Network using Fiorano administration tools and configuration files. It also provides information for Event Process Orchestration and transformation capabilities of the SOA platform. This guide introduces the developer to a rich set of pre-built services bundled with Fiorano SOA Platform. It focuses the usage of these services in real-life scenarios.

1.1 What is Fiorano SOA Platform

Fiorano SOA Platform is a service-virtualization middleware platform that allows heterogeneous software services to be deployed across an enterprise service grid. Distributed Services deployed across a grid of service containers can now be assembled into composite applications to automate business processes. The suite includes a comprehensive set of tools to visually design, configure, deploy, manage, and optimize these business processes.

The Service-Grid architecture of the Fiorano SOA Platform provides a common Service-Oriented platform for Enterprise Application Integration (EAI), Business Process Management (BPM), and Automation of Business to Business (B2B) integration. Together these serve in connecting diverse software applications running within an organization and across partner organizations.

1.2 Why Fiorano SOA Platform

The power of the Fiorano SOA Platform lies in its unique implementation approach (based on distributed peer to peer architecture with centralized control) and the breadth of the **out-of-the-box** components bundled with the Suite.

The classic deployment of a business process (as followed by most products today) has two views: a high-level **business-view** in the form of an activity diagram and a low-level implementation-view of the data flows between applications to implement the higher-level business process. Fiorano replaces these two views with a single view, by taking the implementation view to a higher level, closer to the business-flow. This single-view concept has the critical advantage that changes to the high-level business process (made either by business analysts using graphical tools or via automated scripts) are directly reflected in the implementation, that is, changes do not require the system to be brought down and there is no concept of having to recompile and redeploy a changed process.

Besides the ability to modify a live business process on-the-fly, other key benefits of the Fiorano approach include

- **Deployment of components over an enterprise service-grid**. Fiorano supports a distributed peer-to-peer infrastructure model that allows components within a business process to exchange data directly, without going through a central server. While data-flow between components is direct, control is centralized for ease of administration, achieving an optimal balance in real-world implementations.

- **Multi-Language Multi-Platform Architecture**. Components can be developed in a variety of programming languages, including Java, C, C++, and C# among others. Since all Fiorano SOA Platform components (Fiorano Servers, tools) are developed in the Java programming language, Fiorano servers/tools can run on various operating systems that support Java, including HP-UX, Solaris, Windows, Linux, UNIX, and various mainframe architectures.
- **Single platform for both request/reply and event-driven (data-flow) interactions**. Fiorano uniquely combines request/reply and event-driven interactions within a single platform, enabling fully general-purpose distributed business processes to be deployed with minimal programming.
- **In-built high-availability for processes and data**. Fiorano supports out-of-the box high-availability, with full redundancy being provided using either a pure software or a mixed software/hardware approach for the most demanding enterprise applications
- **Standards support, thin installs**. Fiorano achieves it's flexibility without any proprietary APIs. Key standards support includes JMS, JCA, XSLT, and various web-services standards. Installation of the Fiorano platform does not require any application servers or databases, making the overall installation footprint the smallest in the industry.

1.3 Introduction to Fiorano SOA Platform Environment

A business process that defines the flow of data between software applications is composed of one or more *event processes* (also referred to as Composite Applications). Each event process is composed with Services (also referred to as components), which execute conditions, transformations, provide connectivity to software applications and so on. Components are connected with directional lines, called *routes* that define the flow of data from one component to another. The event process does not contain any code and is not compiled into any executable *.exe* or *a.out*. Event processes are stored as simple XML files.

Fiorano SOA Platform includes bundled graphical tools to compose event processes, as illustrated in screenshots through the rest of this and subsequent chapters.

The Fiorano Environment includes three key items: Servers, Tools, and Components, which are described briefly below.

1.3.1 Fiorano Servers

Enterprise Server: The Fiorano Enterprise Server provides a repository of security meta-data, processes, and components in Fiorano SOA. It also acts as the central server to deploy components onto Peer Servers and fetch/display component status from peer servers. In essence, this is the entry point for the tools to save applications, register components and deploy/monitor event processes. Once the event process is launched the entire flow of messages between the components is managed by the peer server(s) and the enterprise server is used only for monitoring and managing these peers; as such, the role of the enterprise server is that of a monitoring and repository server since it does not play a role in the actual process flow.

Peer Server: This is the runtime server in Fiorano SOA Platform. It provides messaging and deployment infrastructure for the event processes. The Peer Server's messaging infrastructure is built on world's most scalable and robust messaging server, FioranoMQ.

1.3.2 Fiorano Tools

Fiorano Studio: This tool provides a graphical interface to compose, deploy and monitor event processes. It connects to the Enterprise Server, gives you access to saved processes, provides insight into running processes and allows processes to be modified dynamically if needed. This tool includes the **Fiorano Mapper** tool that is used to define data transformations in an event process.

Services and Security Manager: This tool allows administrator to add users/groups and setup Access Control Lists in Fiorano SOA Platform. It also allows new components to be registered into the Enterprise Server.

Event Manager: This tool provides visibility into the runtime information of executing event processes on the peer servers. This information includes documents tracked in an event process, process events (when/who started/stopped the event process), component status, process logs and more.

Deployment Manager: This tool allows users to set deployment rules to govern the deployment of components across development, QA, Staging, and Production environments at the click of a button. For example, users can ensure that Components labeled "development" are disallowed from deployment on a subset of Staging Servers; selected components can be barred from executing on any selected set of peers, using version, label and application-name based restrictions.

Network Administrator: This tool provides access to the status of the Fiorano SOA Platform servers (that is, the Enterprise Server and the Peer Servers) running on the network. Apart from the status info (i.e. "is the server running?") it provides a lot of other information as well, including log, server configuration, security events and more.

1.3.3 Composing an Event Process

An event process is composed of components and data routes. Fiorano SOA Platform includes the following categories of components. Complete listing is available at http://www.fiorano.com/devzone/dev_zone.php. Users can create custom component categories and add custom components to such categories as required.

1. **Bridges:** This category contains components to send and receive files/data from FTP servers, receive emails using POP3, send emails using SMTP. Other components in this category are EJBAdapter, SAPR3 and SMSBridge.
2. **DB:** This category contains components to execute SQL operations (insert, update, delete, stored procedure execution and monitoring database table) on a variety of supported relational databases that includes Oracle, MS SQL, Sybase, etc.
3. **File:** This category contains FileReader and FileWriter components. FileReader can monitor a selected directory for new files as well, apart from reading a specified file path.
4. **Flow:** This category contains components to define conditions; content based routing, aggregate data, and split XML documents etc.
5. **MoMs:** This category contains components to send and receive messages from MSMQ, IBM MQSeries, TibcoRV, and JMS Servers.
6. **Scripts:** This category contains components to execute BeanShell, JavaScript, Perl, Python, and Groovy scripts.

7. **Transformation:** This category contains components to transform data from EDI, CSV, HL7, and XML to XML format and vice versa. It also includes a component to convert XML data into PDF documents.
8. **Utils:** This category contains components to compress and decompress data, encrypt and decrypt data.
9. **Web:** This category contains the HTTPAdapter to send data and the HTTPReceive adapter to receive data over the HTTP protocol.
10. **WebService:** This category contains a WebServiceConsumer to invoke WebServices hosted across the intranet/internet over HTTP protocol.

Configuring components and component input and output data channels

Components receive messages from one or more input data channels and send responses to one or more output data channels (also referred to as *ports*) or to the ON_EXCEPTION data channel.

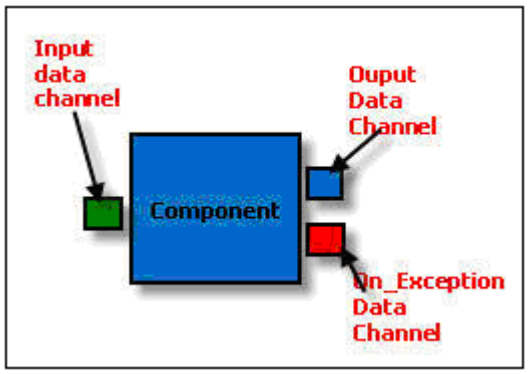


Figure 1.3.1: Service Component

After the component is dropped on to the application easel, it needs to be configured via its configuration interface. Every component has a customized configuration dialog, apart from the generic property window. For example, the DB component has a custom configuration dialog to specify connection details and SQL queries to execute, together with error handling details.

Settings in the custom configuration dialog may result in a modification of the metadata (that is, the XSD) of the input and output data channels. The Input channel metadata/structure enforces the data format in which the component expects the data. Output data channel metadata/structure defines the format of the output data when the component completes its execution.

For example, if the DB component is configured to execute the following query.

```
INSERT INTO mckoiuser.WEB_PURCHASEORDER
( EMAIL, FIRSTNAME, LASTNAME, CONTACT_NUMBER, SHIPPING_ADDRESS,
CITY, ZIPCODE, PRODUCT, QUANTITY )
VALUES ( ?, ?, ?, ?, ?, ?, ?, ?, ? )
```

Figure 1.3.2: Configure Component Query

The component input metadata/structure would be

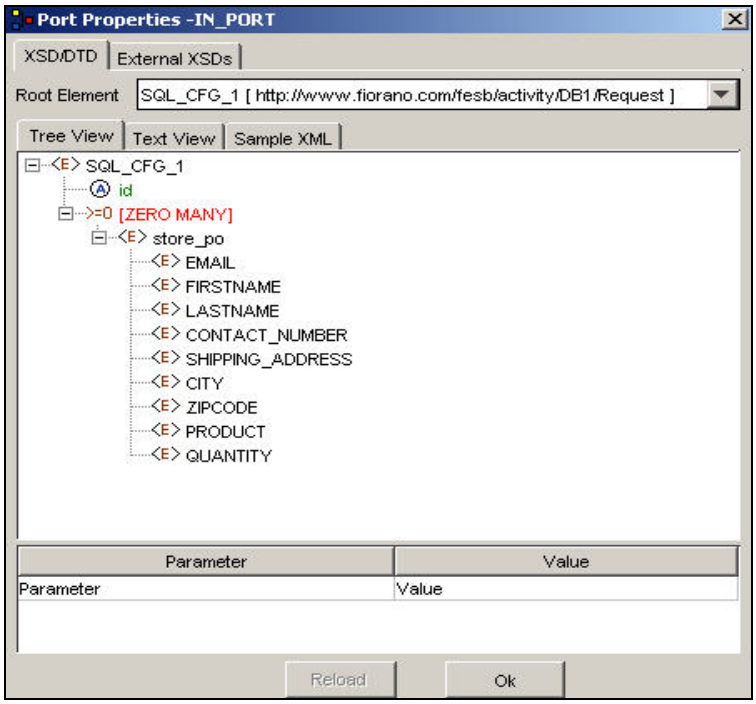


Figure 1.3.3: InPort Properties

The On_Exception channel contains a static metadata/structure with an error code, an error description, a stack trace and the original message that resulted in the exception, as shown in Figure 1.3.4.

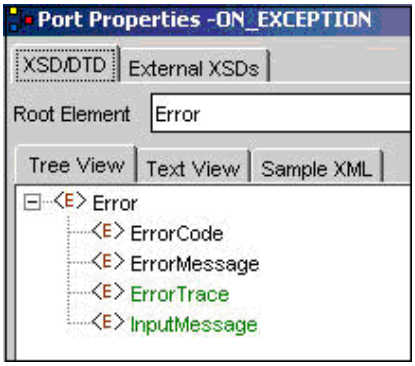


Figure 1.3.4: On_Exception Properties

Automatic triggering (scheduling) vs. triggering through sending a document to the data input channel.

Each component configuration window has a scheduling screen to schedule the automatic component execution. For example, a DB component can be configured to execute the configured SQL every 10 seconds/mins, and so on. Scheduling automatic execution of the component requires the default values to be specified. In case of the DB component, you can specify the default value for every '?' in the SQL query. Similarly, the FTPPut, FTPGet, and the File Adapter can be scheduled to automatically upload the files from a specified directory, download files from a specified remote FTP server and read files from a specified directory respectively.

In other cases, the component starts executing when it receives the messages from the input data channel.

Connecting components with routes

Fiorano provides data routes to transfer data flowing from one component to another. This data/message may contain one or more of XML, Binary, plain text or property data.

A data route is created from the output data channel of a component to an input data channel of another component. A data route can be created between the output data channel and input data channel of the same component. This feature can be used for retrying the message in a flexible way.

If the metadata/structures of the route end-points do not match, the route becomes fragmented indicating a mismatch of data formats. This error can be resolved by introducing an XSLT component between the two components.

The Fiorano SOA Platform supports the W3C XSLT 1.0 language for defining the XML to XML data transformations. Fiorano includes a graphical tool, the Fiorano Mapper to define these transformations. You can easily drag and drop the elements from the source Schema tree to the target Schema tree to define the mappings. The Fiorano Mapper includes comprehensive support for String, Boolean, Arithmetic, Date/Time, and other funclets to define complex data translations. Figure 1.3.5 illustrates a screenshot of the Fiorano Mapper.

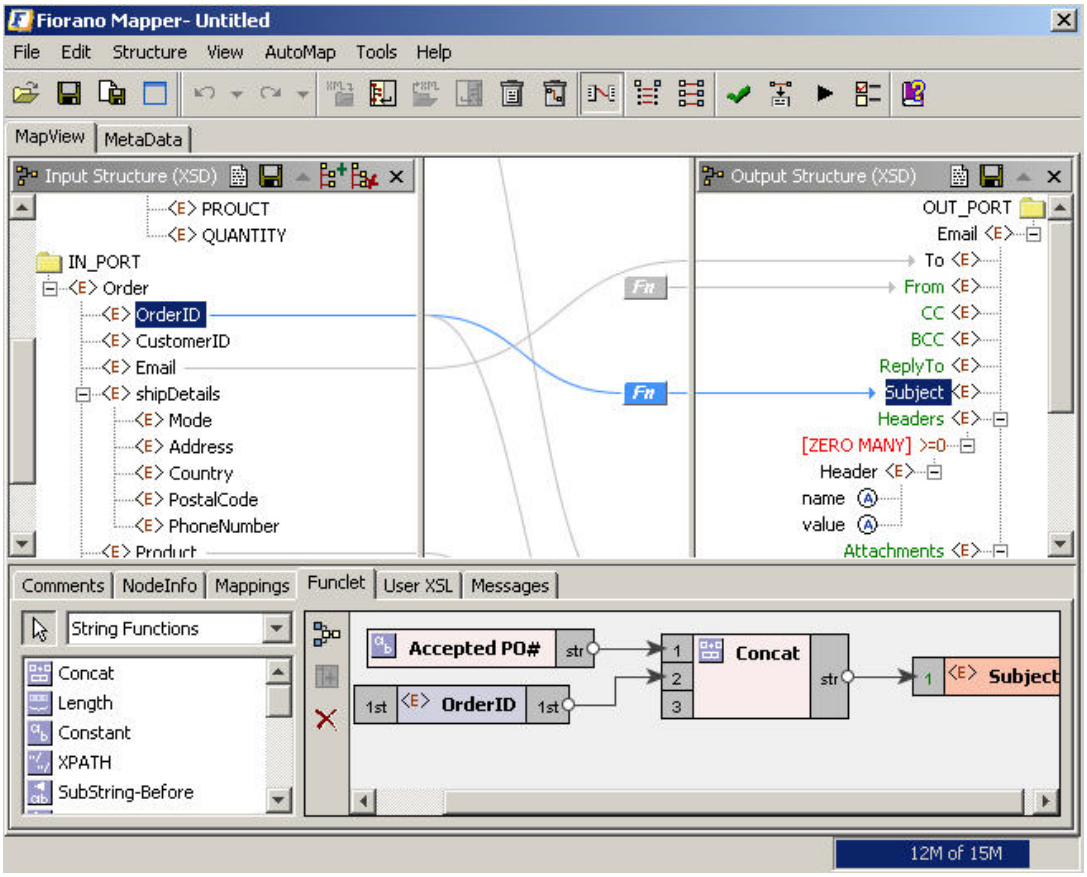


Figure 1.3.5: The Fiorano Mapper Screen

1.3.4 Deploying Event Processes

When a process is deployed, each Component in the Process is moved (that is, dynamically deployed at runtime) to the peer server on which it is to execute. Depending on the settings, Components can run as independent processes connected to the Peer via a Socket, or within the same process as the Peer itself ("in-memory" execution). The Peer Server creates the JMS destinations for each of the input and output data channels for each component connected to it. Executing components pick up messages from the input destination, process these messages and send successful responses to the output destination(s) as required while failed messages are sent to the On_Exception destination.

By default, each component processes one message at a time. The Components can be configured using MultiThreading options (min_threads, max_threads, # of jobs) to process multiple messages simultaneously.

1.3.5 Monitoring Event Processes

The Event Process repository is centrally stored in the Enterprise Server. The Enterprise Server provides API access to the event processes, to save, view, export, launch, debug, and stop and other actions as required. The Fiorano Studio provides an easy-to-use GUI to manage event processes as illustrated in Figure 1.3.6.

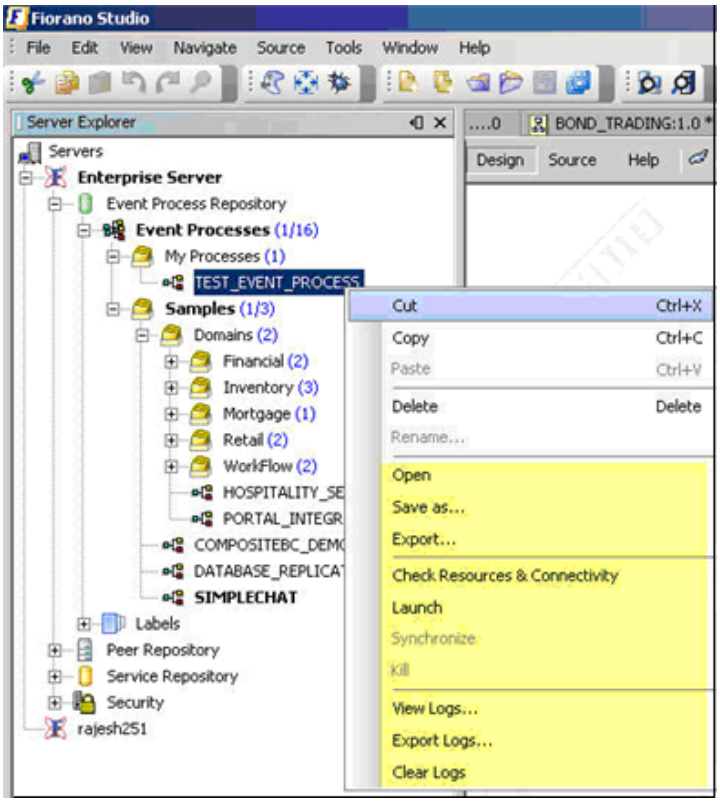


Figure 1.3.6: Server Explorer Screen

The Fiorano Studio can connect to the Enterprise Server from anywhere on the network. In addition to hosting the Studio, which allows users to create new event processes, the studio can be used for the following tasks:

1. Import/Export one/more Event Processes
2. Launch/Stop one/more Event Processes
3. View the Event Process status
4. View process logs

Debugging an Event Process

An Event Process can be debugged to intercept/forward the messages transmitted over the data route. The Debugger shows the complete data message intercepted on the route. A message can also be modified or discarded while in the debugger, provided the user has permission to perform these actions. Figure 1.3.7 and Figure 1.3.8 illustrates the message interceptor/debugger.

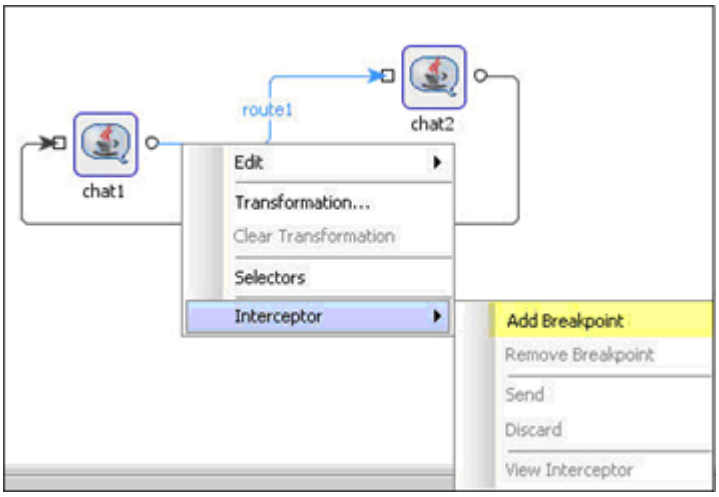


Figure 1.3.7: Add a Breakpoint

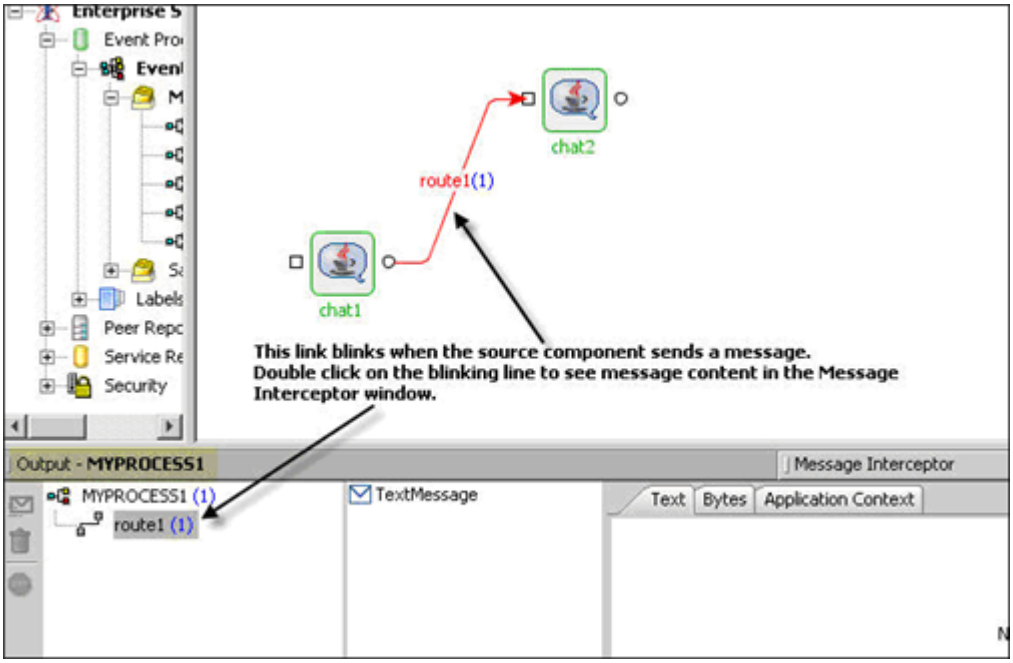


Figure 1.3.8: Intercepted Messages

Tracking documents along the flow (Audit Trial)

An Event Process can contain data channels marked for document tracking, as illustrated in Figure 1.3.9. All the messages that go through a marked data-channel are persisted in a relational database. Additionally, a data channel can be marked end of tracking to denote the end-of-processing steps for a message. The document tracking feature helps to determine the number of documents that have successfully reached the tracking end-points successfully as well as the number of documents that has failed to reach the end-point.

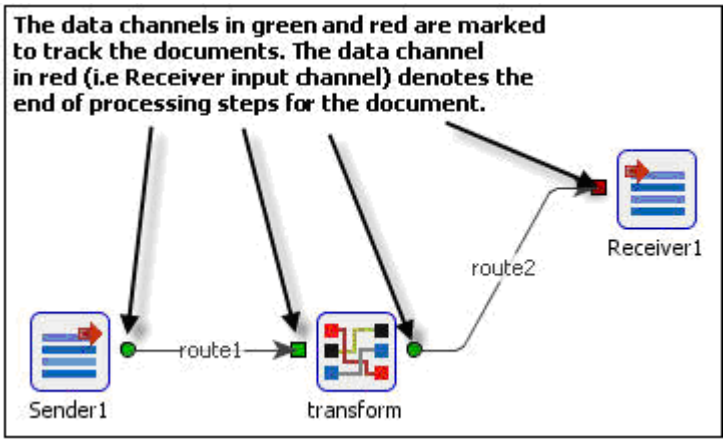


Figure 1.3.9: Tracking Document

Exception Handling

The Fiorano SOA Platform provides the most flexible/extensible exception handling at both component and system levels. As explained in previous sections, if a component runs into any error/exception while processing a document, the error details are published on the ON_EXCEPTION data queue of the component. As a process composer, one can route that data into a table in a database (using the DB component) and/or send out an email (using the SMTP component). The user can also compose a complex flow starting from the error data channel, as illustrated in Figure 1.3.10.

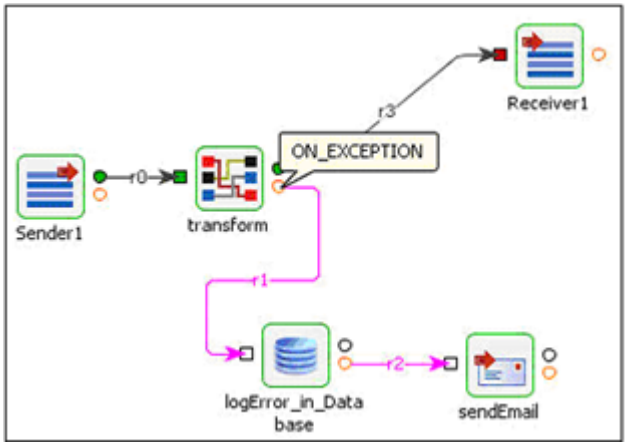


Figure 1.3.10: Exception Handling

In addition to the component-specific error handling, the Fiorano Enterprise Server can be configured to send out an email for various system events like unavailability of the Peer Server, stopped Component, security violation, Event Process termination, and so on.

Logging

The Fiorano SOA Platform provides flexible logging settings for an Event Process. Each component can have different log settings. The log settings determine how much log data gets stored on the disk, the number of lines per file, an option to include time-stamps, and so on.

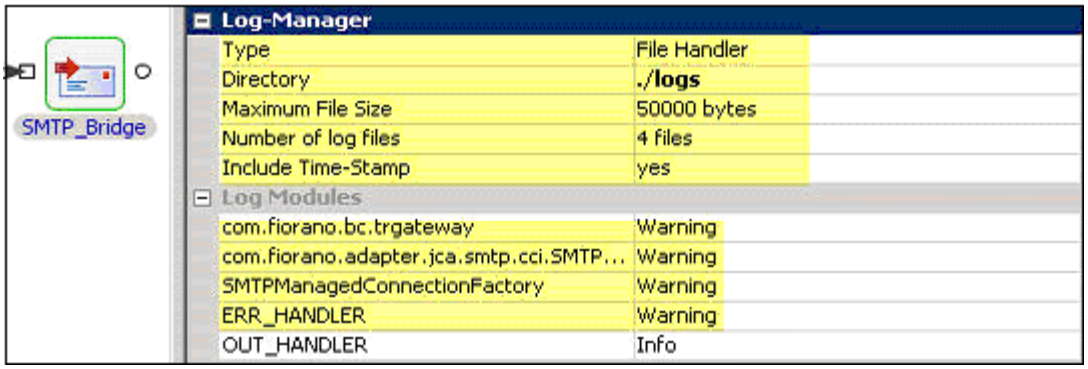


Figure 1.3.11: Log Manager

Fiorano provides APIs to access the runtime process logs while the event process is running with the above log settings. The Fiorano Studio tool provides a GUI to display the process/component logs, as shown the Figure 1.3.12.

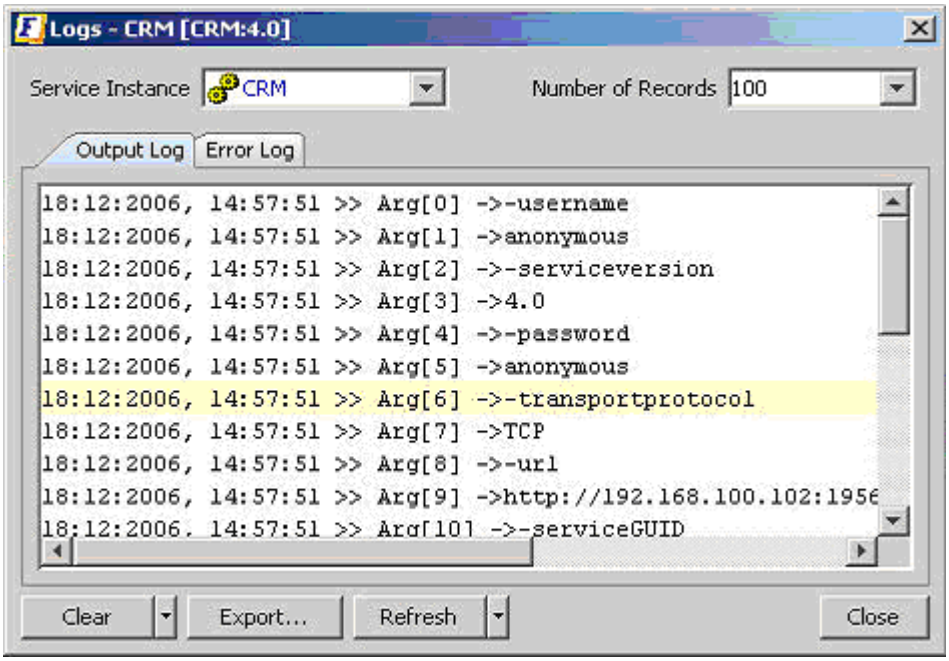


Figure 1.3.12: CRM Logs

Setting up Alerts and Notifications

The Fiorano SOA Platform provides support for setting up alerts and notifications on all system/process/component runtime events. The default installation includes a Mail and SMS notification handler to send out alerts. However, customers can customize the notification mechanism, using a pluggable alert handler API.

It is also important to note that both the Enterprise Server and the Peer Servers fully support management through open standards - JMX and SNMP. Tools like HP-OpenView, Ca-UniCenter, Big Brother, Nagios, and so on. can connect to the servers to view the status and modify system/process parameters.

1.3.6 Extending the Component Palette

The Fiorano SOA Platform suite ships with an SDK to develop components in multiple languages including C, C++, C#, and Java, among others. One can develop a component in any of these languages and make it available on the Studio palette with the help of easy–to-use wizards and scripts.

Once a component is available on the Studio palette, it can be dragged and dropped onto the easel to create an Event Process. A component on the palette can be instantiated in any number of event processes. Some of the benefits of the Java SDK include:

1. An inbuilt JMX container
2. Support for JMS/Tibco RV transport

3. Threading/Session handling

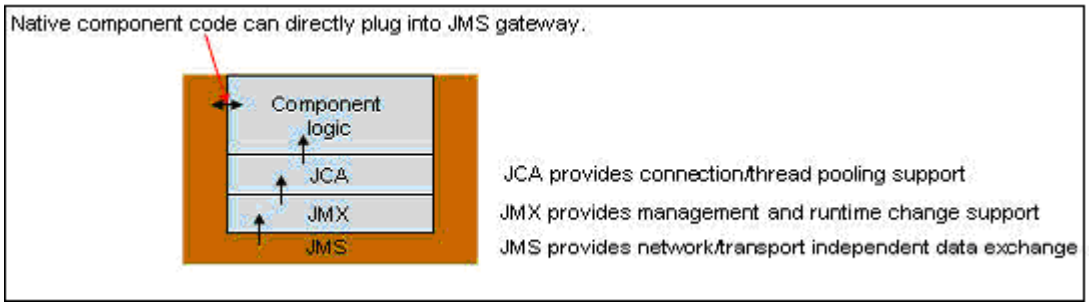


Figure 1.3.13: Adding Components to Palette

1.3.7 Scalability

When a given Peer Server goes out on CPU and RAM capacity, it is easy to install a Peer Server on a new machine and make it available on the Fiorano Network, allowing additional event processes to be launched and managed. There is no limit on the number of Peer Servers that can be added to the network. Linear scalability can be achieved by distributing the event processes onto multiple Peer Servers running on the network.

For more information on how peers are added to a running network, see Chapter 7.

1.3.8 High Availability

The Fiorano SOA Platform achieves system high availability by providing backup/standby instances for the Enterprise Server and the Peer Server.

The Enterprise Server that acts as the process/component repository provides high availability by running a hot-standby Enterprise Server on a separate machine across the network. The hot-standby server and the active server exchange heart-beat packets and the data through a back-channel connection. If the active server goes down, the standby server becomes active and provides high availability for the process/component repository.

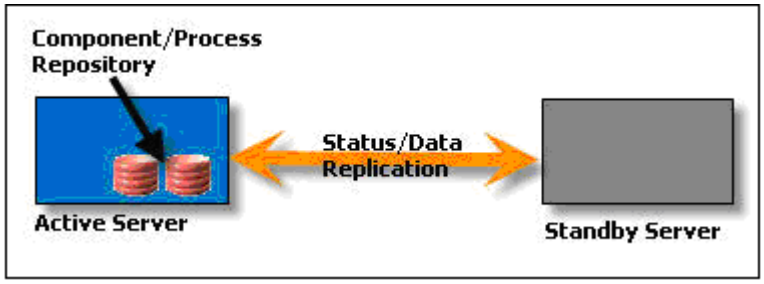


Figure 1.3.14: HA on Enterprise Server

Similarly, a Peer Server provides high availability of data and service failover by running a backup/hot-standby server on a separate machine. In the event the primary server machine goes down, the hot-standby instance resumes operations immediately without any manual intervention. The Peer Server HA-pair instances provide high availability of transactional data together with component-level failover. High availability is discussed in detail in Chapter 6.

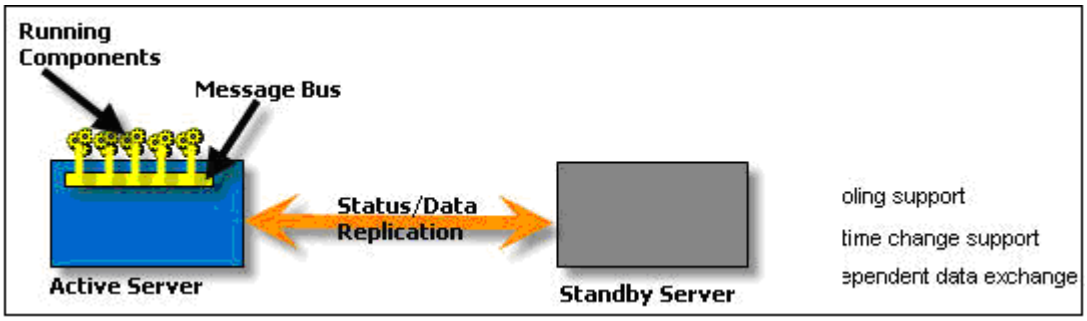


Figure 1.3.15: HA on Peer

1.3.9 Security Framework

The Fiorano SOA Platform security data (that is, both authentication and authorization data) is centrally managed in the Enterprise Server. Access to both the component and process data is authenticated/authorized when users connect to the Enterprise Server. The Fiorano Security Manager tool is used to configure the users/groups and user/group permissions. Security is discussed in more detail in Chapter 8.

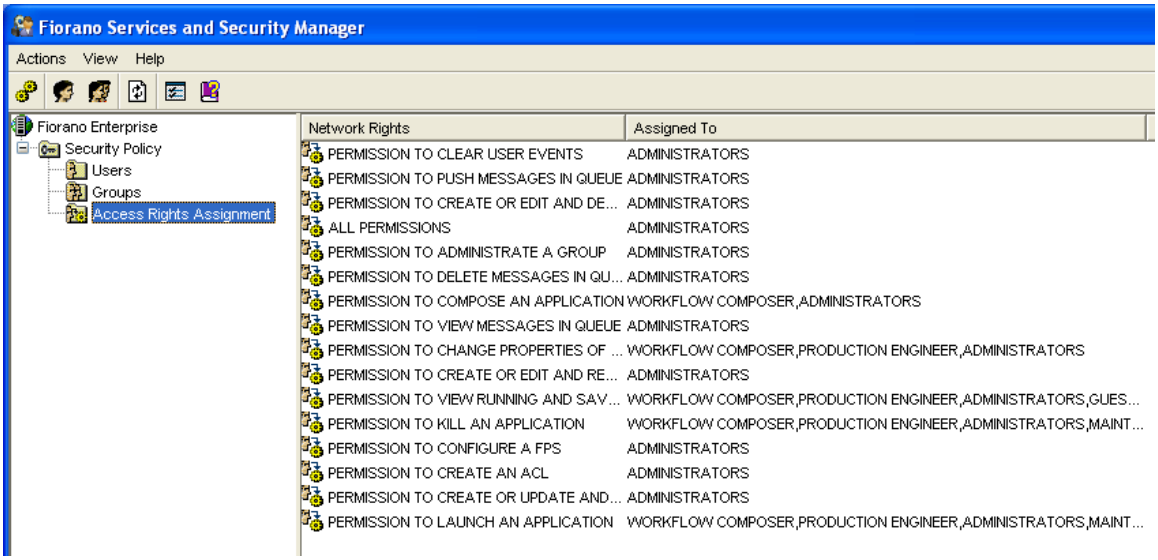


Figure 1.3.16: The Fiorano Services and Security Manager Screen

Chapter 2: The Fiorano Environment

This chapter explains the high-level system architecture of the Fiorano Enterprise Services Grid, which consists of Peer Service- Containers installed across the network together with the centralized Management and Repository Server as well as Management tools. The topics discussed in this chapter include the ESB Server, ESB peers, the Web-Gateway that manages Web-Services and Fiorano Orchestration and Management tools.

2.1 Fiorano System Architecture

The Fiorano SOA Platform include the:

1. Fiorano Peer Server Network
2. Fiorano ESB Server
3. Fiorano Service Components
4. Fiorano Tools Interface

Figure 2.1.1 illustrates how different entities of the platform interact with each other in carrying out their respective functionalities.

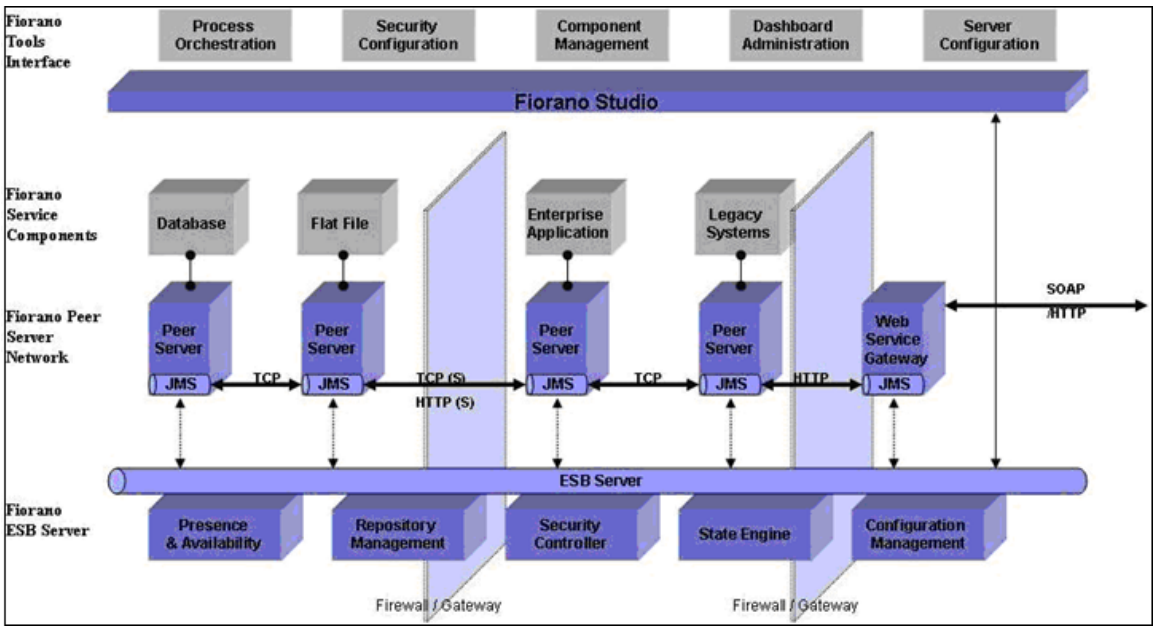


Figure 2.1.1: The Fiorano System Architecture

- The Fiorano Peer Server Network is the Enterprise Class, centrally managed Peer-to-Peer messaging backbone.
- The Fiorano ESB Server is the Administration Gateway to the ESB Peer Server network.

- The Fiorano Service components are either the interfaces to units of the Enterprise IT Infrastructure or implementations of commonly used integration elements (such as transformations, content based routers, and so on). The Fiorano Service components implemented in Java are either pure JMS or JCA components.
- The Fiorano Tools Interface is provided via the Fiorano Studio which offers intuitive visual interfaces for the implementation of ESB Server capabilities for end Users.

2.1.1 Fiorano ESB Server

The Fiorano ESB Server is the central controller of the Fiorano network. This control server acts as a monitoring agent for all other peers and ensures information coherence.

The various functions performed by the FES include:

1. Control of launching as well as termination of Fiorano Components as part of an event process on any Peer of a Fiorano network.
2. The update of status of all Peer Servers, Business Components and Event Processes running on Peers across the network.
3. The Launch of a Business Component on a backup node, in case the primary FPS on which the Business Component is running goes down.

2.1.2 Fiorano Peer Server

The Fiorano Peer is a container for launching business components at network end-points of a Fiorano network and manages the life cycle of the components.

The following are key functionalities of the Fiorano Peer:

1. Transfers the data among various components in a Peer to Peer fashion over JMS routes.
2. Routes Business component related controls and State information to the Enterprise Server.
3. Provides store and forward capabilities to handle network failures to provide for guaranteed delivery of messages (which flow across from one peer to another). A Fiorano Peer Server has inherent store and forward mechanism through which each peer stores the messages corresponding to all Peer Servers which are unavailable at that moment and forward these messages to Peer Servers as and when any come up again. This allows the Event Processes to continue execution even in the event of remote machine failure or network failure.

ESB Server to Peer Server communication

All data communication in the ESB network happens in direct Peer to Peer fashion between Peer Servers. Only control data flows between the Enterprise and the Peers. The types of control events handed by the ESB Server include service component state notifications, event process state notifications, HA events, document tracking events and so on.

ESB Component and Process Repository

The ESB Server manages the following repositories:

- The repository to maintain versions of registered and unregistered service components along with dependent resources and binaries.
- The Meta data information of the event processes in XML format.
- The repository of Peers in the ESB network and their configuration

Communicating with ESB Server

The ESB server is the single point of management and administration for the ESB network. The Fiorano Studio acts as the visual interface to the ESB Server function for the End-User.

2.2 Installation

The Fiorano SOA Platform is available in two editions:

1. **Enterprise Edition:** The Enterprise Edition includes the Fiorano Enterprise Server, Fiorano Peer Server, Fiorano Adapters, and all the Fiorano tools.
2. **Workstation Edition:** The Workstation Edition includes the Fiorano Peer Server and all the Fiorano tools. The Workstation Edition is not bundled with the Enterprise Server. This edition requires an Enterprise edition to be installed on the Fiorano Network.

The Enterprise and Workstation Edition are available free for a 45-day evaluation period. The evaluation versions contain all the features of the licensed versions. The use of this software is defined in the Fiorano End-User License Agreement.

2.2.1 Different Topologies

The Fiorano Peer-to-Peer distributed model provides unmatched flexibility in deciding installation topology. Users can decide on an installation topology based on various parameters:

- Availability of hardware
- Hardware configuration – RAM (Random Access Memory) and, CPU processing
- Expected system performance
- Number of licenses available

The exact topology architecture for each solution varies and is determined based on specific customer requirements.

All Servers on Same the Machine

Fiorano Platform installation (Enterprise Edition) can be installed on a single machine. The default Peer Server profile is configured to work with the Enterprise Server available on the local machine. As such, no further configuration (other than JVM settings, like setting the heap size) is needed to launch the Servers. The Fiorano Studio also comes pre-configured with the local Machine Enterprise Server connectivity information.

A single machine installation is the simple and fast way to get acquainted with the software. If the single machine has more than the recommended RAM and processing capabilities, **then the servers can be deployed on the same machine**. However, if a large number of processes are required to run and the available hardware configuration does not support the memory requirements of the business process, then users have to consider distributed deployment of servers.

Enterprise Server on a Machine and Peer Servers on Separate Machines

Another popular installation strategy is to have a dedicated machine for the central controller – the Fiorano Enterprise Server. Peers can be distributed across other machines. With this approach it is possible to distribute the load across multiple mid-range machines as compared to using a single high-end machine.

For example, the Figure 2.2.1 illustrates an installation topology spread across five machines.

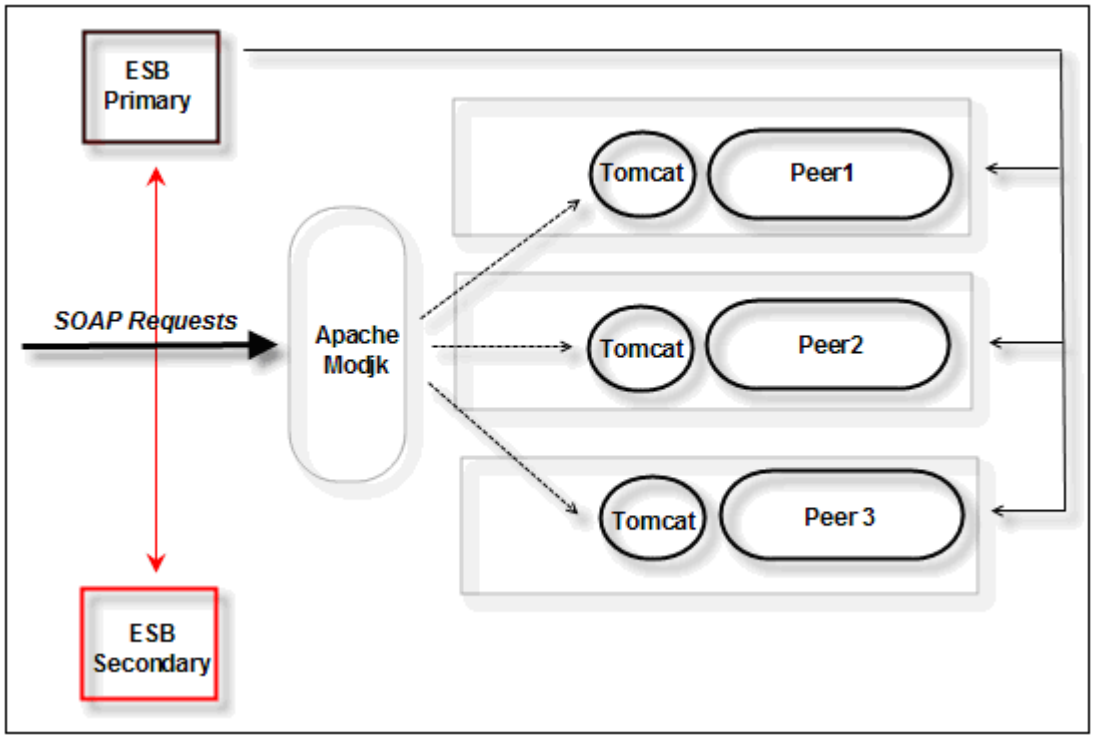


Figure 2.2.1: Installation Topology

2.2.2 ESB Server

The Enterprise Server is the centralized management and repository Server which manages the various components of the Fiorano Network.

2.2.2.1 System Requirements

The ESB Server is a 100% Java product and can be deployed on any platform with a JRE version 1.5 or above. For optimum performance, a minimum of 1 GB RAM is recommended. A complete Enterprise Edition installation requires 600 MB of disk space.

2.2.2.2 Installation Steps

The windows installer is wizard driven on which the default installation directory can be configured. The non-windows installer includes a wizard driven approach or a tar file that can be easily unzipped into a directory and then untarred.

Solaris Platform installation instructions:

- Use gtar in Solaris to untar
- GNU tar is bundled as: /usr/sfw/bin/gtar

Check the link below for further information:
http://forum.java.sun.com/thread.jspa?threadID=5106899&messageID=9363883

2.2.3 ESB Peers

The ESB Peer Server is a runtime container for service components. The Peers communicate with each other directly in a Peer-to-Peer method without going through the Enterprise Server.

2.2.3.1 System Requirements

The Peer Server is a 100% Java product and can be deployed on any platform with a JRE 1.5 version or above. For optimum performance a minimum of 1 GB RAM or above is recommended.

The actual RAM and the hard disk space required for deployment depends on the number of Business Services running on a single node, complexity of the Business Services deployed on a single node, CPU requirements by the Business Services, anticipated data flow, expected performance from the system, and so on.

2.2.3.2 Installation Steps

The Windows installer is wizard driven on which the default installation directory can be configured. The non-windows installer includes a wizard driven approach or a tar file that can be easily unzipped into a directory and then untarred.

Solaris Platform installation instruction:

- Use gtar in Solaris to untar
- GNU tar is bundled as: /usr/sfw/bin/gtar

You can check the link below for further information:
http://forum.java.sun.com/thread.jspa?threadID=5106899&messageID=9363883

In the wizard driven approach, you can configure the peer to register with a specific Enterprise Server when using the Workstation edition of the installer as shown in the Figure 2.2.2:

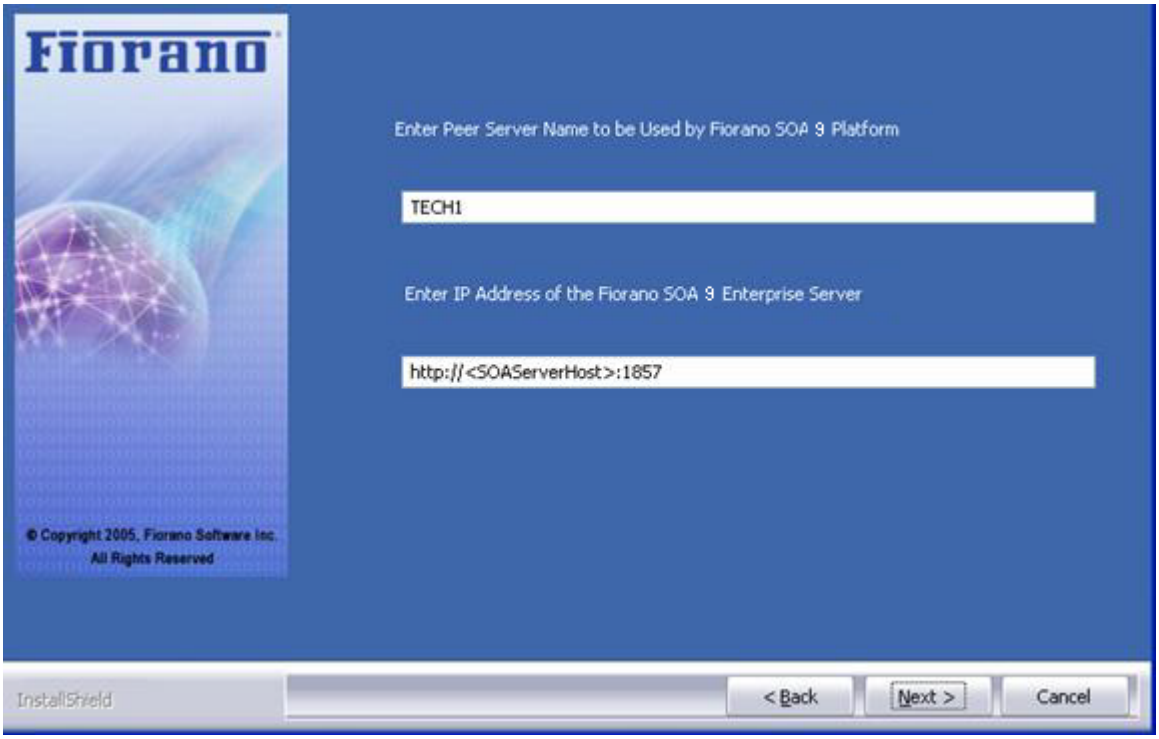


Figure 2.2.2: Installation Wizard

2.3 Fiorano ESB Server

The Fiorano ESB Server (FES) is a Centralized Management Server which manages the various components of the Fiorano Network and acts as a Metadata Repository of Event Processes and Fiorano Business components. In a Fiorano Network, the data flow takes place purely in a Peer-to-Peer fashion (between two or more FPS) without the intervention of FES. Thus the FES role during an Event Process execution is restricted to passing control signals for starting, stopping, and monitoring service components that comprise the event process, where as the actual flow of data and events between services is managed by Peer Servers on which the services execute.

The FES is the central controller of the Fiorano Network, acting as a monitoring agent for all the other Peers while ensuring information coherence. The Fiorano SOA Platform Management tools connects to the FES and request action for specific operations such as launching an Event Process or retrieving information about services in the network. For this, the FES sends out Control Events to the participating FPS in the network containing specific instructions about the incoming request.

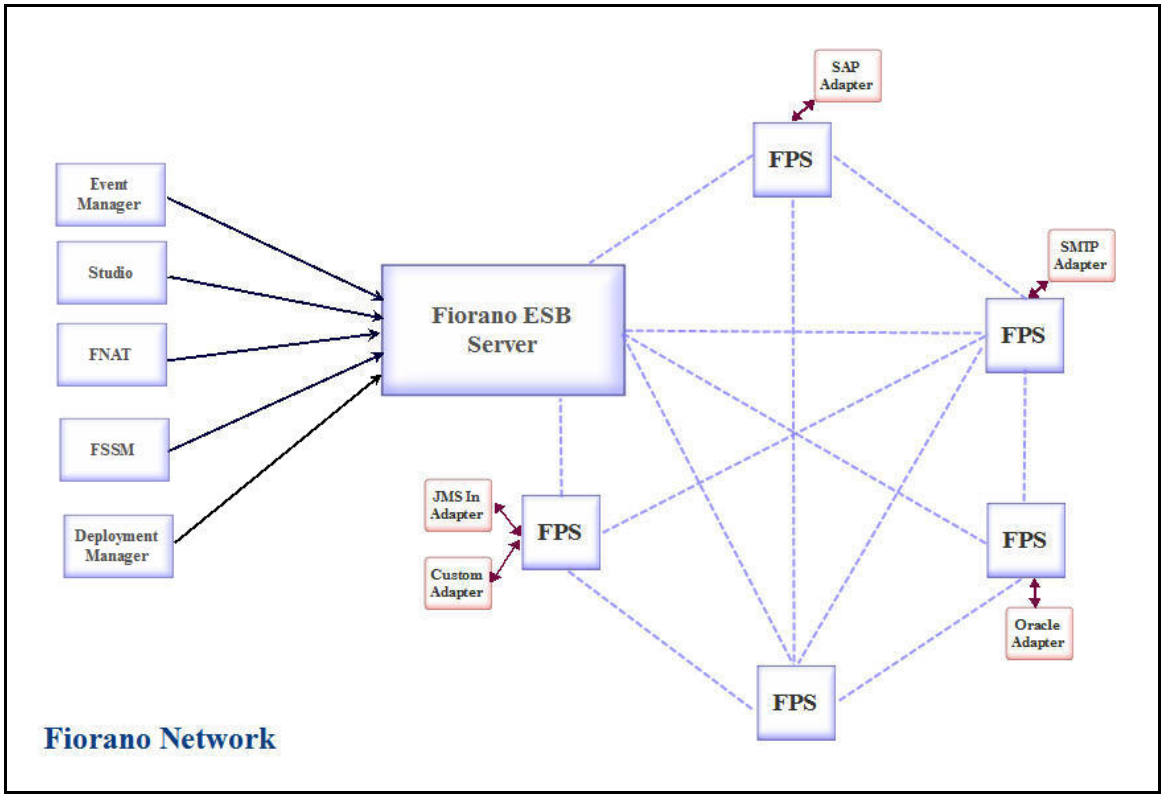


Figure 2.3.1: Fiorano Network

The FES is primarily a JMS Server (FioranoMQ Messaging Server) wrapped with additional role specific functionalities. The FES communicates with the FPS using the underlying JMS server and expose well defined JMX and Java API to communicate with the tools.

2.3.1 ESB Server Functionality

The role of a FES can be logically divided into the following activities:

Remote Deployment and Launching of Event Processes and components: The FES provides the ability to deploy, launch, and stop Event Processes and components on any Peer Server in the network.

Runtime Composition of Event Processes: The FES allows modifications to the Event Processes running at the time without stopping the Event Process on any server. This helps to update the running event processes in production without any downtime.

Configuration and Repository Management: In a distributed Service-Grid environment, information and data is scattered across the network. Each component that executes a part of the event process or workflow as well as the Peer Server hosting the component needs to be configured. Hence, there is an important need for remote configuration management of the overall system and this is managed via the FES.

The FES manages:

- The configurations of Fiorano ESB Peers.
- A repository to maintain versions of registered and unregistered Service Components along with dependent resources and binaries.
- The meta-data information of the Event Processes in XML format.

Presence and Availability Management (PAM): The FES maintains the State information of all the Peer Servers across the Fiorano Network. This State information is stored in a file-based data storage.

Event Process State Persistence: The FES persist the State of Event Processes and restores the State(s) upon restarting the server.

Event Tracking and Monitoring: The FES monitors events, logs and State information of Peer Servers, and Service-Components running across the Fiorano Network. The FES makes this information available to other tools for business activity monitoring.

Runtime Debugging: The FES allows debugging of Event P0072ocesses at runtime and also provides the ability to modify the intercepted data at runtime.

Security Controller: Security plays a critical role in a distributed system. There are two parts to Security: one relating to network security level and protocol level security and the other relating to user-level security.

- Protocol level security is the inherited feature of Fiorano Servers (both FES and the FPS) in that it can be configured to use a secure protocol for communication, including support for HTTPS and SSL protocols.

- User-level security is important in avoiding problems such as a Peer injection of malicious and corrupt data in to the system. A client when connecting to the Fiorano Network is required to supply credentials which are then authenticated by the system. In a Fiorano Network, the Enterprise Server through the underlying Realm service does the authentication of users and maintains the security policies. This Realms service is responsible for maintaining all user and group information and for authenticating all incoming connection. The network administrator can choose between a collection of Realm services, differing in storage and authentication mechanisms. This security architecture allows the administrator to set up Access Control Lists (ACLs) for each possible operation that could be carried out in the Fiorano Network as well as Control User actions based on the permissions assigned to them. For example, ACLs for an Event Process can specify which Users have the privilege to launch an event process on the network. Similarly, ACLs for a Business components can specify locations where the Business component can be run on the network. This allows the administrator to control the privileges available to each Users.

Failover Management: The FES also manages the Fail-Over of the service components in applications that is running. When the primary FPS on which a service component is running goes down, the FES redeploys and launches that component instance on a configured backup node.

High Availability: Fiorano ESB Servers can be configured to run in HA mode to maximize system availability, thereby eliminating all single points of failure and avoiding data loss.

2.3.2 Launching ESB Server

The FES can be launched from the Windows Start menu or by directly executing a script file.

2.3.2.1 From Fiorano Studio

Click Start→Programs→Fiorano→Fiorano SOA Platform →Fiorano Servers→Fiorano ESB Server

2.3.2.2 From Script Files

- To start the FES server with the default profile (which is profile1), run or double-click the script *server.bat/.sh –mode fes* available under <fiorano_installation_dir>\esb\server\bin folder.
- To start the FES server with a specific profile other than the default profile, run the script *server.bat/.sh* available under <fiorano_installation_dir>\esb\server\bin folder with the profile option as shown below:

server.bat/.sh –mode fes –profile <profilename>

Example: <profileName>=haprofile1/primary or <profileName>=haprofile1/secondary - This is to start servers in HA mode.

Note: In UNIX systems, servers, by default, will start in the Background mode. Pass runtime argument –**nobackground** to the server startup script to run the server in console mode. If the Background mode is not to be run, then run %FIORANO_HOME%/launcher/server.sh so that **nobackground**="" is changed to **nobackground**="true".

To install FES as a Windows NT service

Run the command:

install-server.service.bat –mode fes -profile %PROFILE_NAME%

Note: NT services will have to be re-installed if you make changes to any configuration files. This includes fiorano_vars as well as server.conf.

To remove the FES NT service, run the command:

uninstall-server.service.bat –mode fes -profile %PROFILE_NAME%

To install/uninstall FES as a service on UNIX, refer to **readmeWrapperService.txt** present under %INSTALL_DIR%/SOA/esb/server/bin/service directory.

2.3.3 Shutting Down ESB Server

The FES can be stopped from the Fiorano Studio or by directly executing a script file located under %INSTALL_DIR%/SOA/esb/server/bin directory. Shutting down the Enterprise Server automatically invokes a shutdown hook in the Enterprise Server's JVM which cleans up any resources and connections used by the server's JVM.

2.3.3.1 From Fiorano Studio

The FES can be stopped from Fiorano Studio only after logging into FES through Fiorano Studio. The Figure 2.3.2 illustrates the Shutdown menu item location:

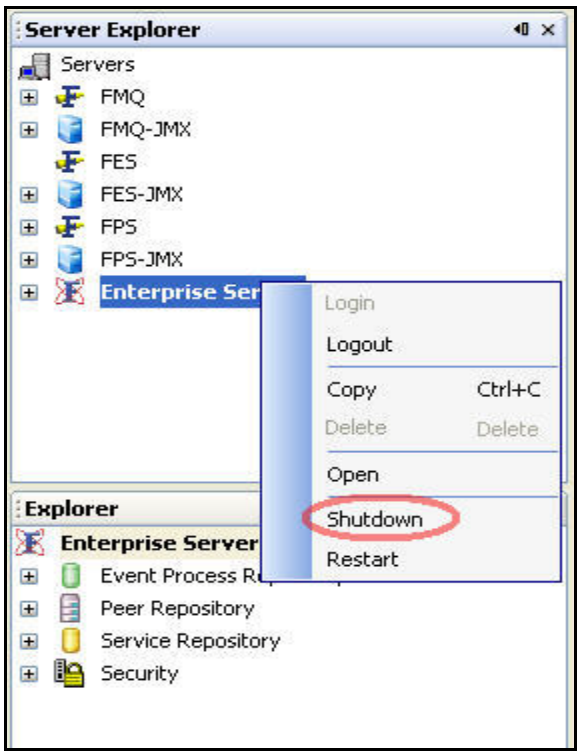


Figure: 2.3.2: Shutting Down FES

2.3.3.2 From Script Files

The Enterprise Server can be shutdown using the **shutdown-server** script present in <fiorano_installation_dir>/esb/server/bin folder.

Note the following:

- Shutdown scripts cannot be used to shutdown both primary and secondary servers running in a shared HA mode simultaneously, and using the RMI connection.
- In case, HA profiles from previous versions are migrated to the current version of the product, these profiles need to be re-configured to specify the BackupRMIServerPort property in order to make use of this functionality.

This script can be used to:

- Shutdown a Standalone Enterprise Server.
- Shutdown both the Primary and the Secondary HA servers simultaneously.

Options that can be specified to the script are:

- **-user:** Name of user trying to shutdown Fiorano Server
- **-passwd:** Password of user trying to shutdown Fiorano Server
- **-restart** or **–r:** Restarts the Fiorano Server
- **-ha:** Used to shutdown both active and passive servers running in HA.
- **-url:** URL of active Fiorano Enterprise Server
- **-mode:** mode of the server, that is, fes or fps. Defaults to value fps.
- **-?** or **–help:** Prints help message

Examples:

To shutdown standalone Enterprise Server:

shutdown-server -url tsp_tcp://localhost:1947 -user admin -passwd passwd

To shutdown both active/passive enterprise servers in HA:

shutdown-server -url tsp_tcp://localhost:1947 -user admin -passwd passwd -ha

The above examples of shutting down the Enterprise Server(s) involve connecting to an Active Enterprise Server, whose URL is specified using the **-url** parameter.

The following options can also be specified if a User wants to shutdown the Enterprise Server using the RMI connection.

- **-connectorType:** Connection type to server.Pass "RMI" for shutting down servers using RMI connection.
- **-address:** IP Address of server
- **-rmiPort:** rmi port of server

Note:

If the **-ha** option is given to shutdown both the servers of **ha** pair, the **address** and **rmiPort** options need to be specified for either the primary or the secondary server. This can be used when you are not sure which server of the HA pair is active.

Examples:

For shutting down the Standalone Server:

shutdown-server -connectorType RMI -user admin -passwd passwd -address localhost -rmiPort 2047

For shutting down both active/passive servers in HA:

shutdown-server -connectorType RMI -user admin -passwd passwd -address localhost -rmiPort 2047 -ha

2.3.4 ESB Server Configuration

FES can be configured either in online mode (that is, while the FES server is running) or offline mode (when the FES server is not running) using Fiorano Studio.

- Offline mode configuration can be done using the Profile Manager, which can be accessed from the Fiorano Studio. When a configuration is modified and saved, changes persist and are reapplied when the server is restarted.
- Online mode configuration can be performed using the JMX explorer, which can be accessed from the Fiorano Studio. Changes made using JMX are applied on the current running server and are persisted as well. However, some of the server configurations (such as server ports, memory settings and others) are applied only after the restart of the server.

Note: For more information on configuring Fiorano Servers, please see section 2.6 Configuring Servers and Tools.

2.3.4.1 Server Ports Configuration

FES server communicates with Peer Servers and tools using different ports.

2.3.4.1.1 External Ports

Ports that are opened to facilitate the communication between FES and external tools are known as external ports or external server ports. The FES checks these ports for any requests from external tools.

Configuration Steps

Offline Mode

1. Open the **FES** server profile in Studio (Tools→Configure Profile). The FES server profile can be selected from the following location as shown in the Figure 2.3.3.

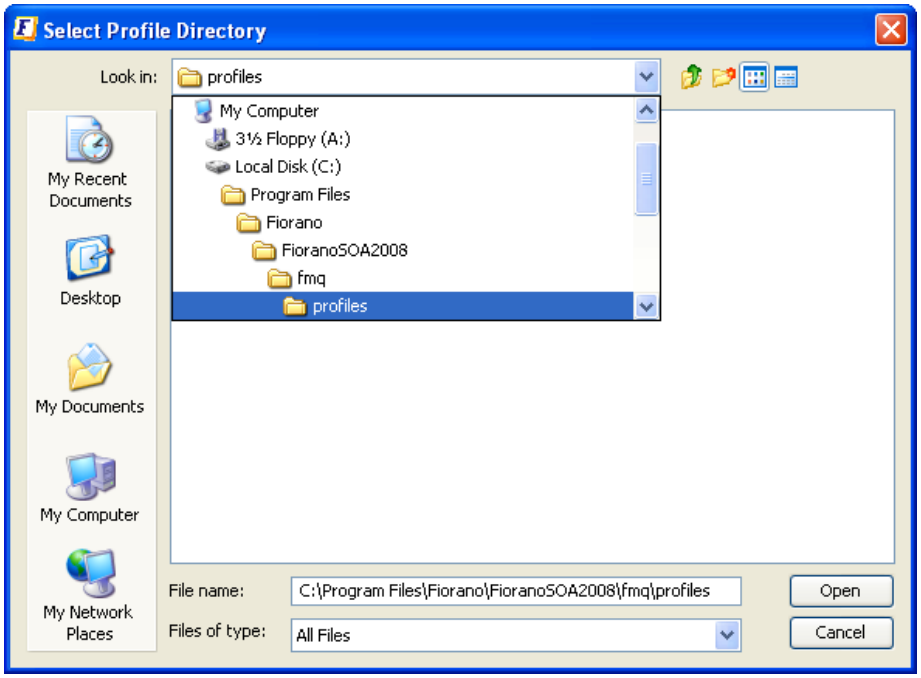


Figure 2.3.3: FES Profile Location

2. Select **the FESTransportManager** in the Server Explorer tab FES→Fiorano→Esb→FESTransportManager. In the **Properties of FESTransportManager** panel displayed on the right-hand side, change the port number in the **ServerUrl** property.

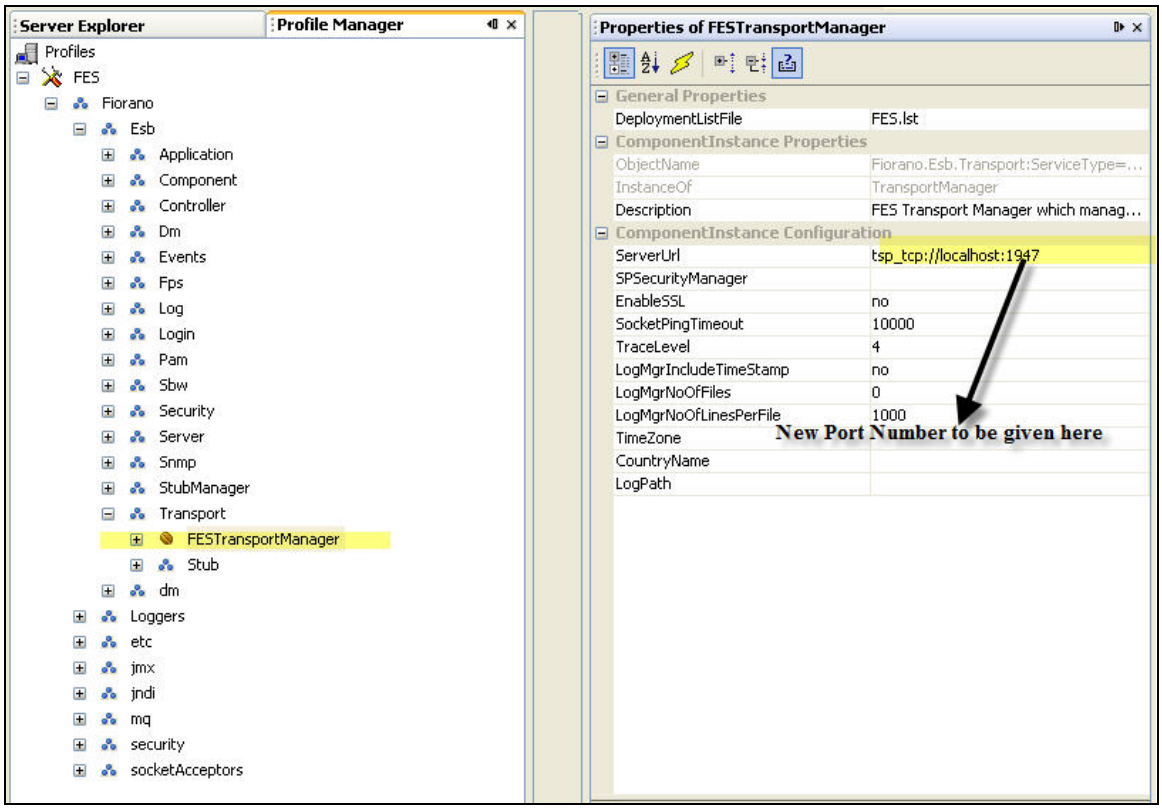


Figure 2.3.4: FES External Port

Note: FES clients (such as the Studio and other tools) should connect to this port after the FES is started with this profile saved.

3. Save the profile as shown in the Figure 2.3.5.

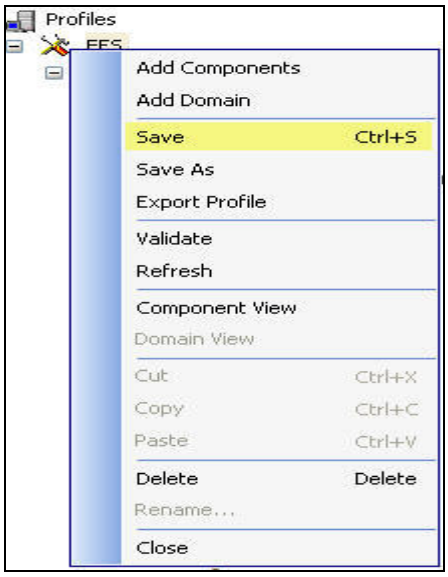


Figure 2.3.5: Saving Profile

Online Mode

1. Connect to the Enterprise Server's JMX interface through the Fiorano Studio.

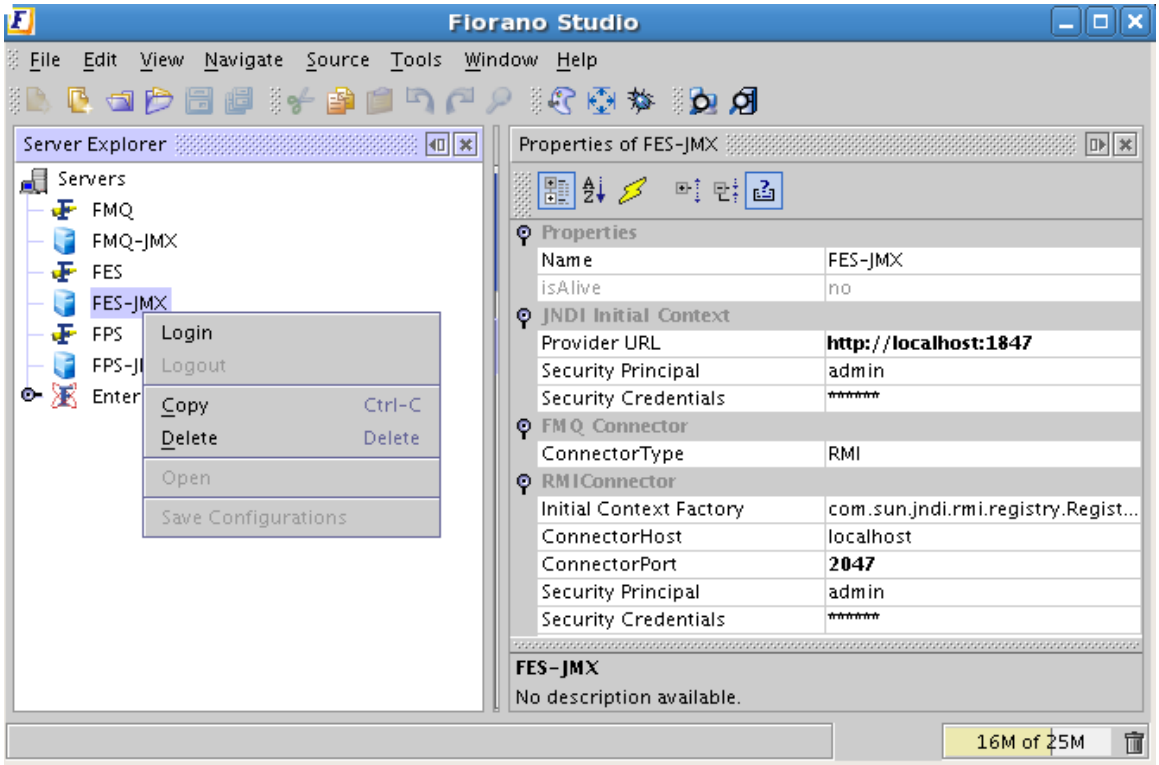


Figure 2.3.6: FES-JMX Server

2. Select **JMX Connection** → Fiorano → Esb → Transport → FESTransportManager. In the **Properties of config** panel on the right-hand side, change the **FES URL** property to reflect the new port.

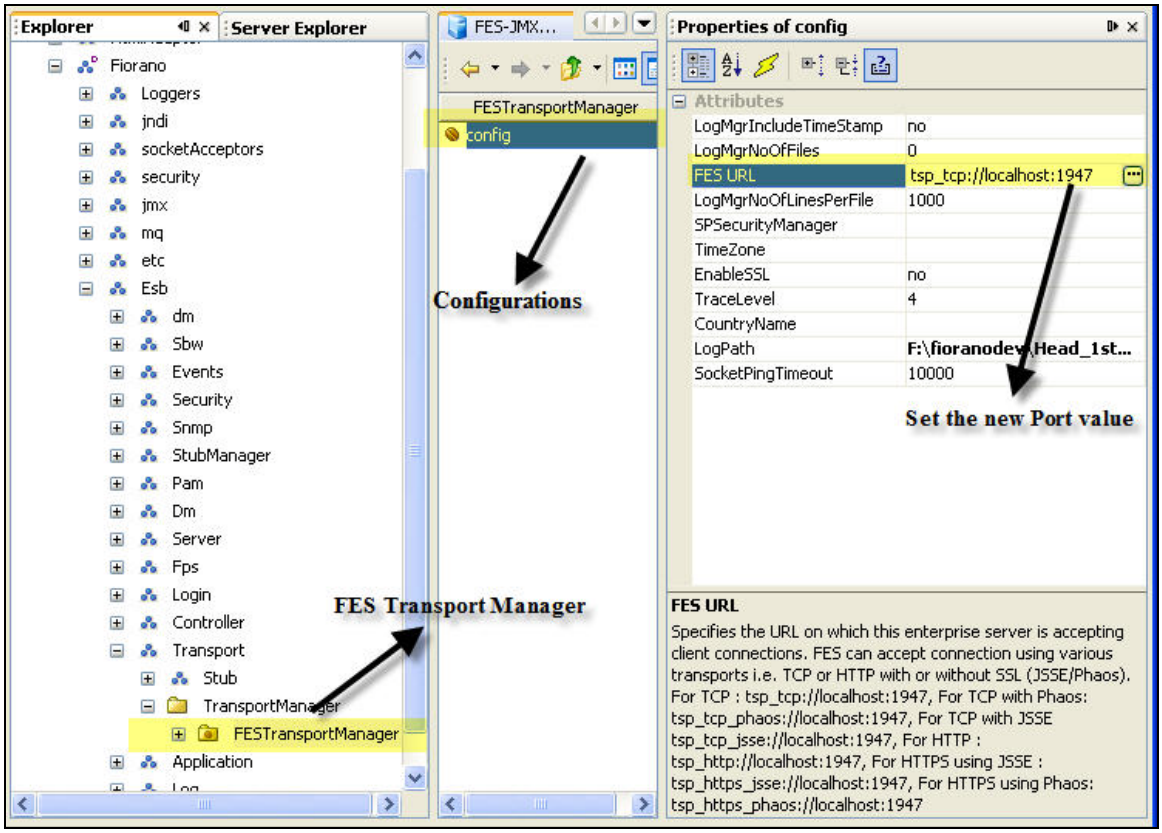


Figure 2.3.7: JMX Explorer

3. The server needs to be restarted after the value is set. A message dialog box appears with instructions for the properties that require the Server to be restarted.

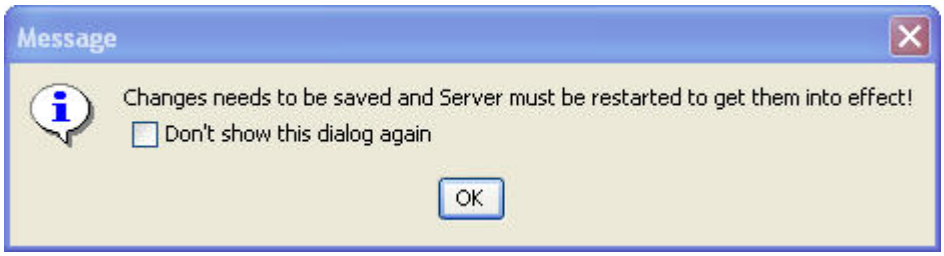


Figure 2.3.8: Save Dialog Box

4. Click the **OK** button to save the configurations. These configurations are reflected after the server is restarted.

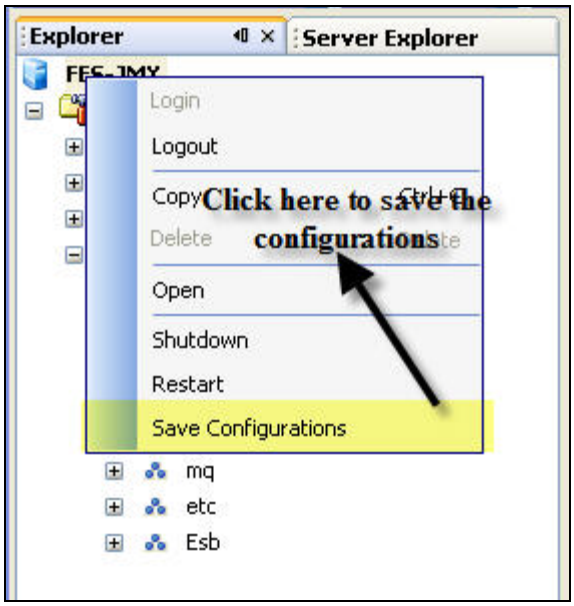


Figure 2.3.9: Save Configuration

2.3.4.1.2 Internal Ports

Ports that are opened to facilitate the communication between FES and FPS servers are known as internal ports or internal server ports. The FES checks these ports for any requests from external tools.

Configuration Steps

Offline Mode

1. Open the server profile in Studio (Tools→Configure Profile) as explained in section 2.3.4.1.1 External Ports.
2. Select the **ConnectionManager** in FES→Fiorano→SocketAdapters→port-1→ConnetionManager. In the **Properties of ConnectionManager** panel on the right-hand side, change the **Port** property.

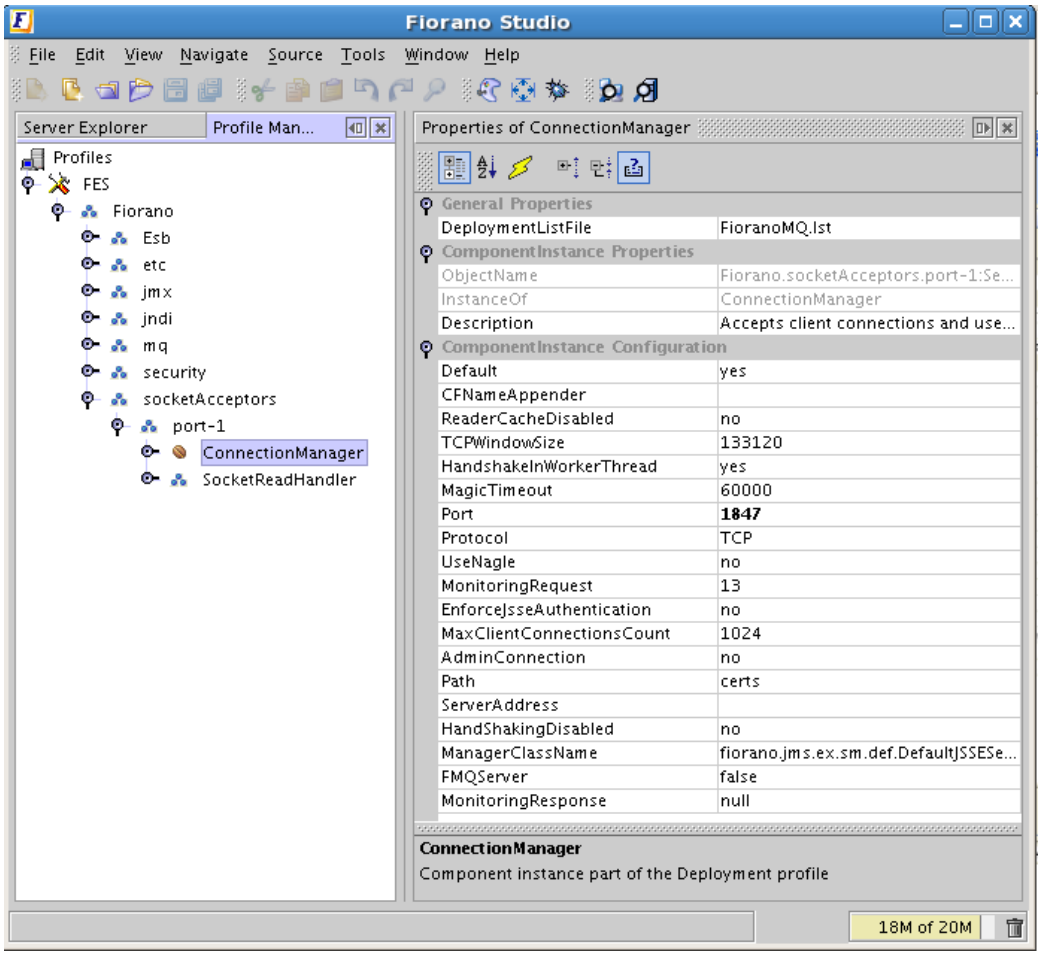


Figure 2.3.10: FES Internal Port

Note: The Peer Servers should connect to this port after FES is started with this saved profile.

3. Save the profile as explained in section 2.3.4.1.1 External Ports.

Online mode

1. Connect to the Enterprise Server's JMX interface through Fiorano Studio as shown in 2.3.4.1.1 External Ports.
2. Select **JMX Connection** → Fiorano → socketAcceptors ConnectionManager → **ConnectionManager**. In the **configuration properties** panel on the right-hand side, change the **Port** property to the new port.

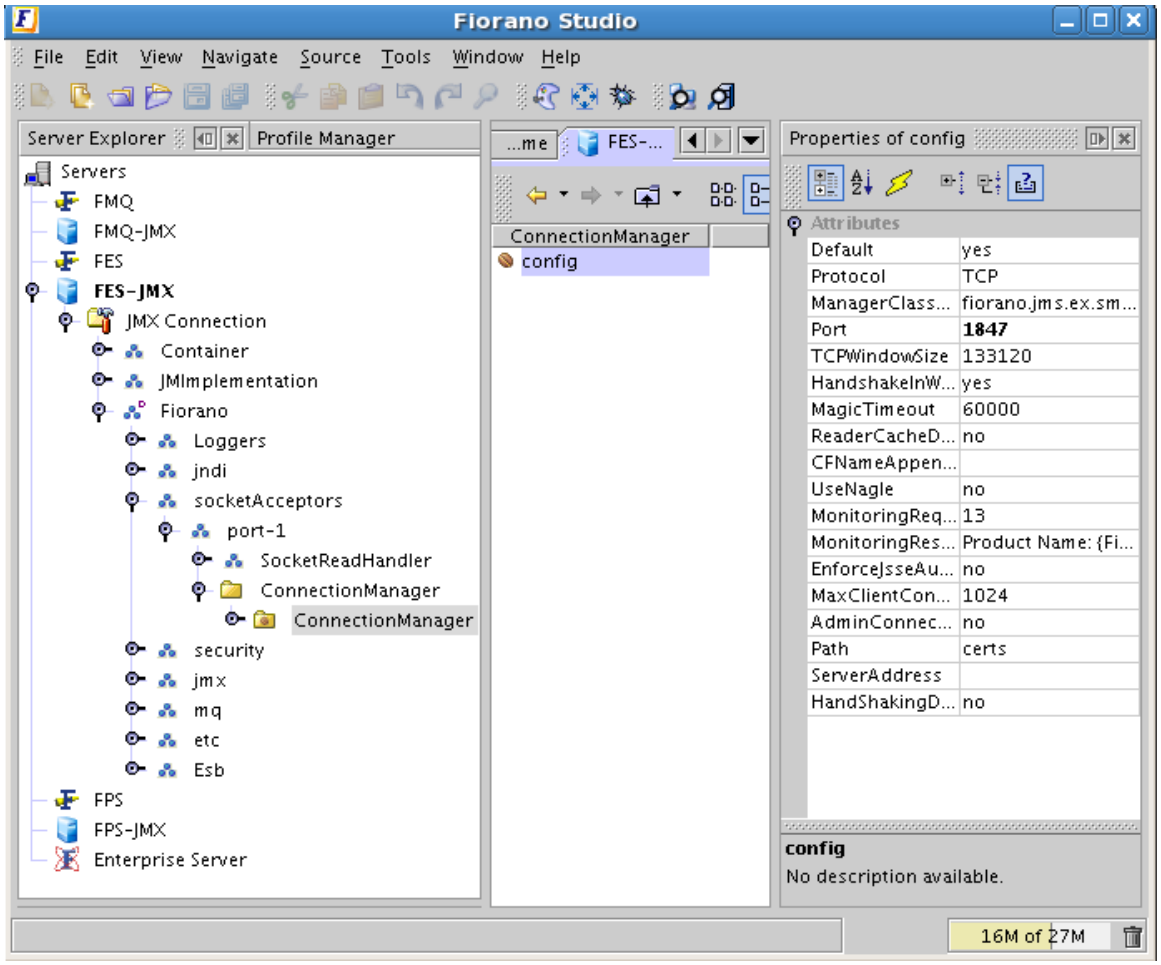


Figure 2.3.11: JMX Explorer

3. The server needs to be restarted after the value is set. A message dialog box appears with instructions for the properties that require the Server to be restarted, as explained in section 2.3.4.1.1 External Ports.
4. Click the **OK** button to save the configurations. These configurations are reflected after the server is restarted as explained in section 2.3.4.1.1 External Ports.

2.3.4.1.3 RMI Ports

Ports that are opened to facilitate the communication between FES with JMX clients are known as RMI Server Port.

Configuration Steps

The RMI ports can be configured in either offline or online mode.

Offline Mode

1. Open the server profile in Studio, navigate to Profile Management > Enterprise Server and select **FES**.
2. Select the **RMIBasedJMXConnector** from FES → Fiorano → JMX -> Connector. In the **Properties of RMIBasedJMXConnector** panel on the right-hand side, change the **RMIServerPort** property as required.

   The default RMI Ports for FES is 2047.

   Defaults:

   - FES HA Primary - 2047
   - FES HA Secondary - 2048

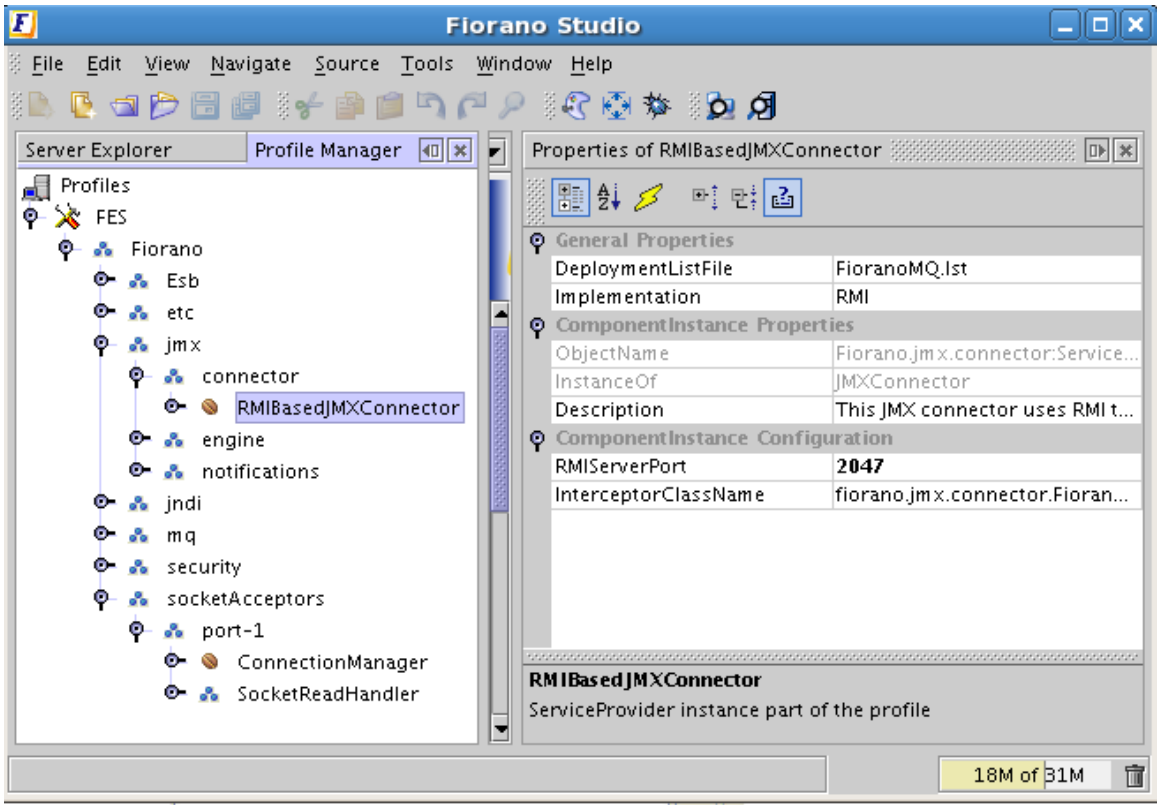


Figure 2.3.12: FES RMI Port

3. Save the profile as explained in section 2.3.4.1.1 External Ports.

Online Mode

1. Connect to the Enterprise Server's JMX interface through Fiorano Studio as explained in section 2.3.4.1.1 External Ports.
2. Select JMX Connection → Fiorano → jmx → connector → JMXConnector → RMI → RMIBasedJMXConnector. In the **Properties of Config** panel on the right- hand side, change the **RMIServerPort** properties as required.

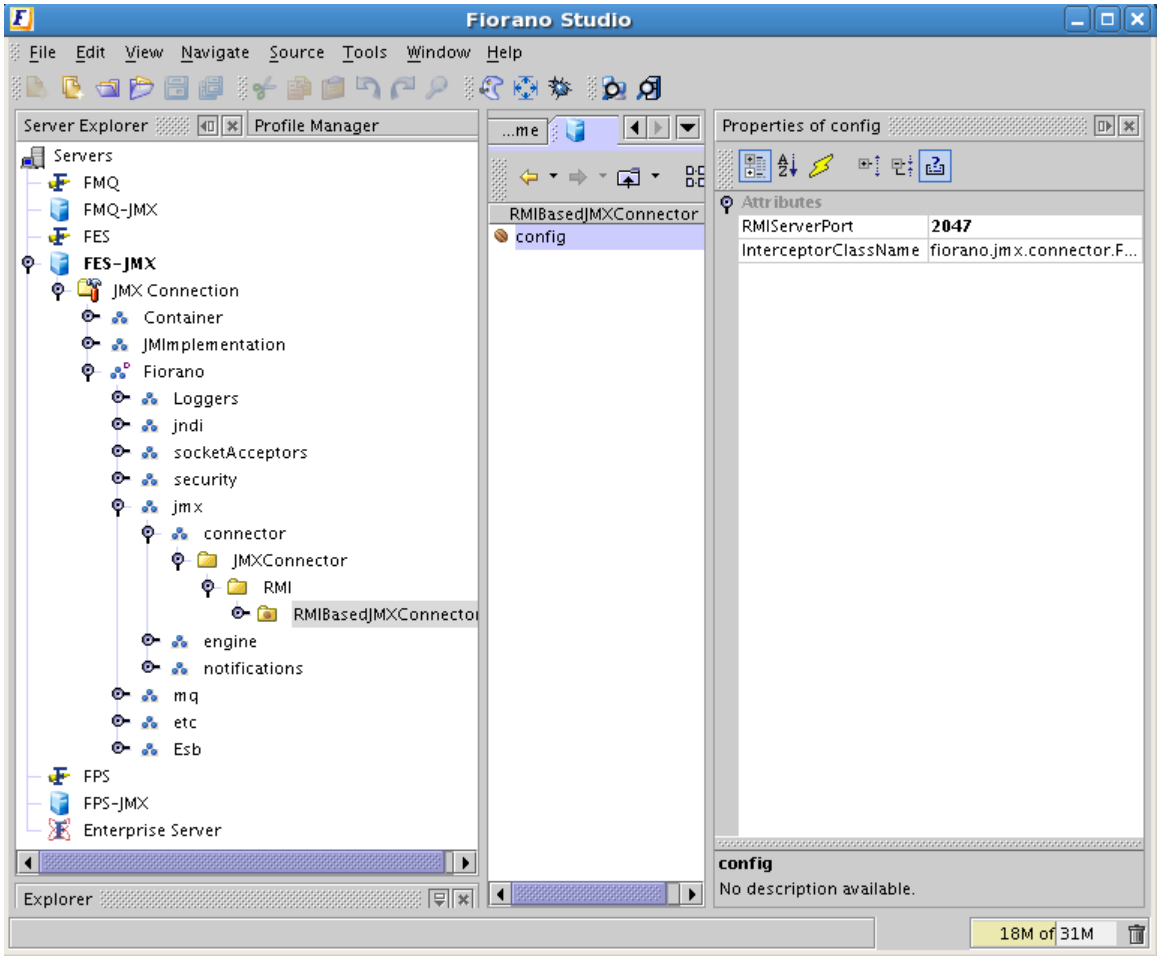


Figure 2.3.13: JMX Explorer

3. The server needs to be restarted after the value is set. A message dialog box appears with instructions for the properties that require the Server to be restarted as explained in section 2.3.4.1.1 External Ports.

4. Click the **OK** button to save the configurations. These configurations are reflected after the server is restarted as explained in section 2.3.4.1.1 External Ports.

2.3.4.2 Memory Configurations

Better server performance is possible with proper configuration of JVM parameters, particularly those related to memory usage. The allocation of memory for the JVM is specified using -X options when starting the server.

JVM option	Meaning	Default ESB Settings
-Xmx	Maximum heap size.	512MB
-Xms	Initial heap size	256MB
-Xss	Stack size for each thread.	JVM default (120k)

Note: The stack size limits the number of threads that you can run in a given JVM. A large stack size may result in memory running short as each thread is allocated more memory than needed.

Configuration Steps

1. Open the server.conf in %FIORANO_HOME%/esb/server/bin/ and change the values for –Xms and –Xmx argument under <jvm.arg> tag. If no value for –Xss is specified, default value will be used.
2. Save the file and restart the server.

2.3.4.3 Java Configurations

Enterprise server requires JRE version 1.5 or above for successful operation. The JAVA_HOME setting can be configured for the Enterprise Server as follows:

On UNIX:

The Enterprise Server by default uses JAVA_HOME value set for the console. This can be overridden by specifying JAVA_HOME value in **%FIORANO_HOME%/fiorano_vars.sh** file.

On Windows:

By default, JAVA_HOME is set to %FIORANO_HOME%/jre1.5.0_16 (shipped with the product) in **%FIORANO_HOME%/fiorano_vars.bat** file. To make the server use a different JAVA_HOME, modify the settings as required.

2.3.5 Setting Up Users and Groups

The Fiorano SOA Platform security policy enables administration and management of Groups and Users across the entire Fiorano Network. This section describes management of Users and Groups by assigning appropriate rights to them on the Fiorano Network.

The Fiorano SOA Platform users and groups can operate from all available nodes in the Fiorano Network. A User Group is identified by a unique name and contains a list of users who inherit all rights assigned to that group. Every User is assigned a unique User Name, Password, and a User Group Membership. Information pertaining to Users and User Groups is utilized during authentication and determines the resources that a User or a User Group is allowed to access.

2.3.5.1 Managing Users

Fiorano Studio can be used to manage all users in the Fiorano Network. The management tasks that can be performed are:

- Creating User Accounts
- Deleting User Accounts
- Changing the Password of a User

To view the list of Users, log onto the Enterprise Server and click the Users node in the security section. A list of Users is displayed as shown in Figure 2.3.14.

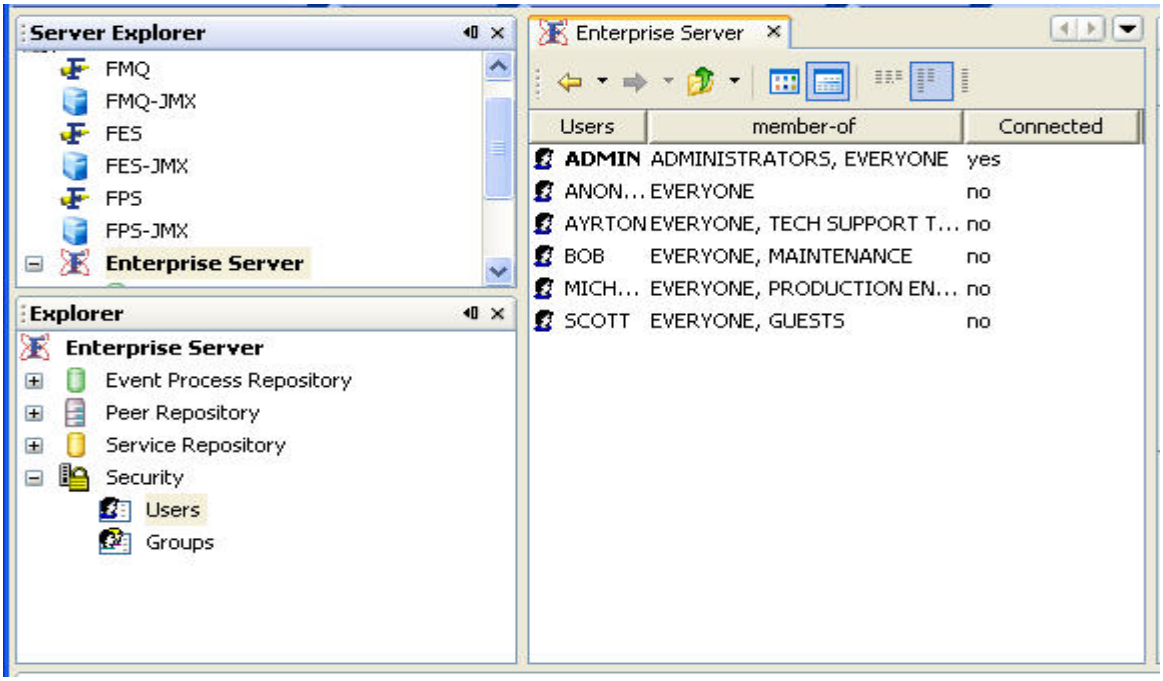


Figure 2.3.14: Users in the Fiorano Network

Note: Logged in users are shown in bold letters.

2.3.5.2 Creating a New User Account

You can create a new user account by logging onto the Enterprise Server with the administrator's privileges.

2.3.5.3 Configuration Steps

1. Login to **Enterprise Server** through Fiorano Studio.
2. Select security node from the Enterprise Server tree.
3. Right-click on the User and click **New User**. A dialog box is displayed with a prompt to enter the name of the User. Enter the new User Name and click the **OK** button.

   Note: The Password created for the New User, by default, is the same as the User Name assigned to the User.

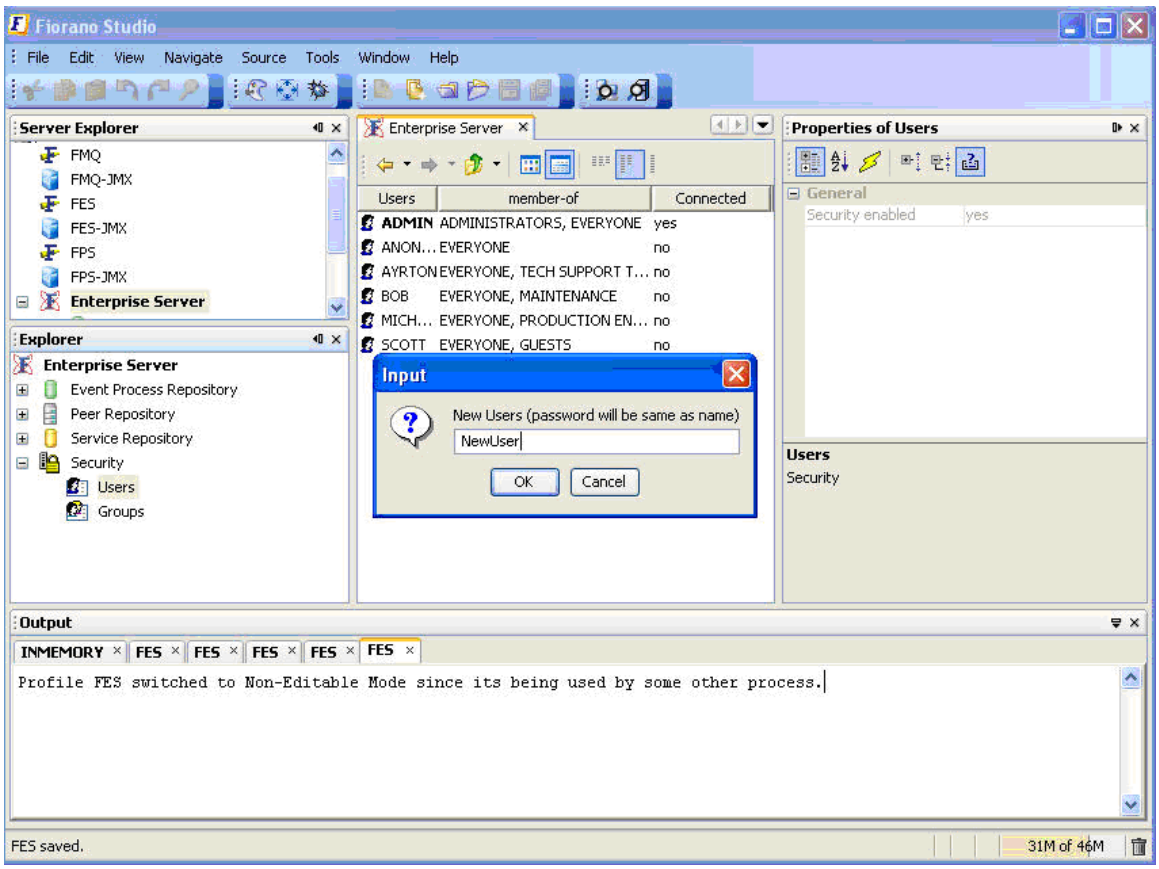


Figure 2.3.15: New User creation dialog box

4. To change the password, right-click on the user whose password is to be changed. A dialog box is displayed with a prompt for the Current Password and the New Password. Enter the new password and click the **Yes** button to complete the process.

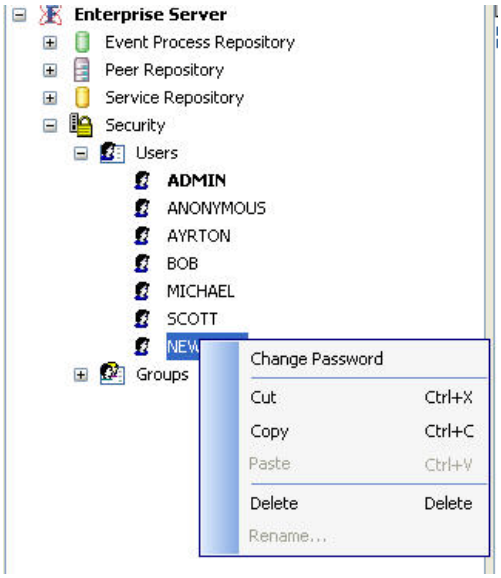


Figure 2.3.16: Change User Password

5. To delete a User Account, right-click on the user to be deleted and select **Delete** option. A dialog box is displayed with a prompt for confirmation. Click the **Yes** button to complete the process.

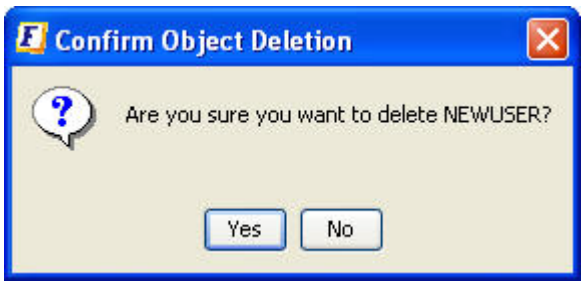


Figure 2.3.17: Confirmation of Deletion of User Account.

2.3.5.4 Managing Groups

The Fiorano SOA Platform by default creates a Group named EVERYONE. All Users are automatically included in this Group. When 'Groups' is selected from the security section, all groups are displayed in the right-hand side panel, as shown in the Figure 2.3.18.

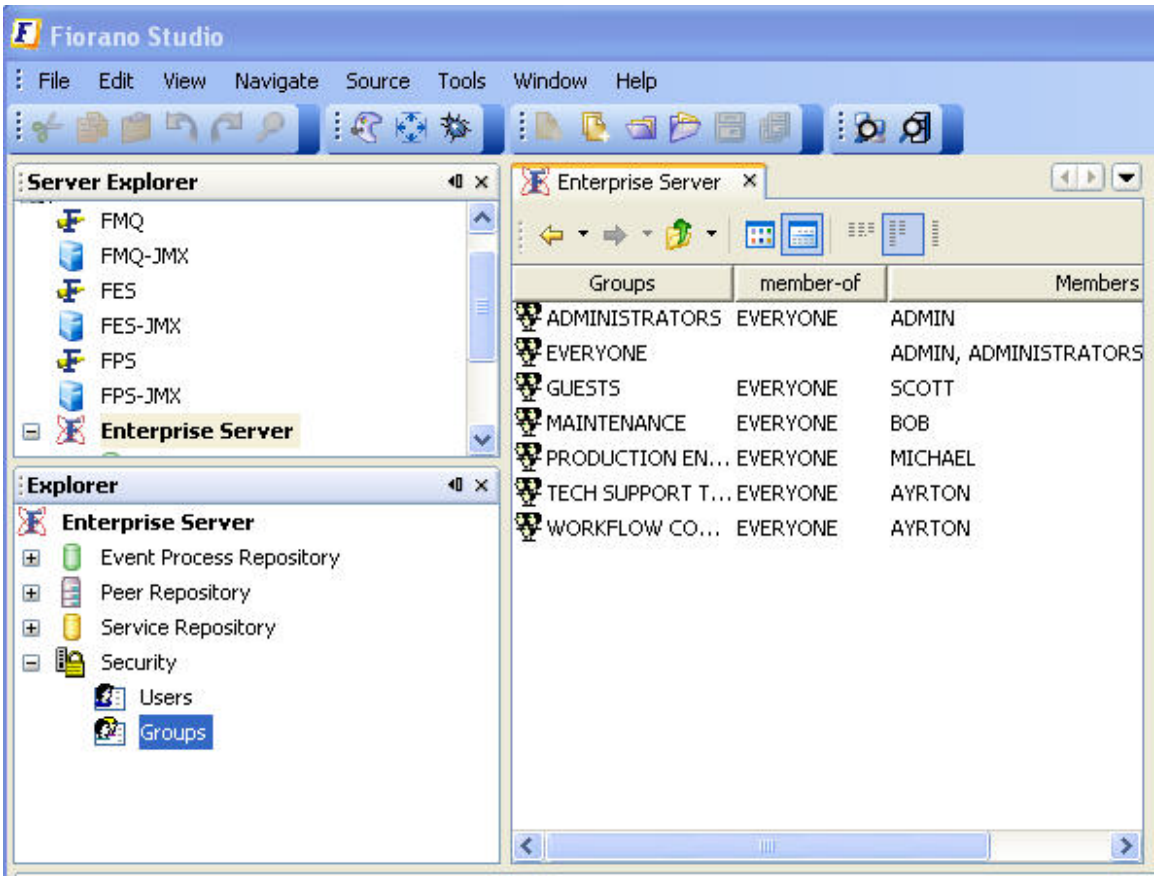


Figure 2.3.18: Groups in the Fiorano Network

The information pertaining to each group is organized under the following columns:

- **Group Name:** This column contains the names of the Groups
- **Members:** This column contains a list of Users who belong to that particular Group.

2.3.5.5 Creating New Group

Any User with administrative privileges can create a New Group by logging onto the Enterprise Server.

2.3.5.5.1 Configuration Steps

1. Choose the Enterprise Server under **Explorer** after logging onto Fiorano Studio**.**
2. Select the security node from the **Enterprise Server** tree.
3. Right-click on the **Groups** and select the **Add Groups** option. A dialog box is displayed with a prompt to enter the Name of the Group. Enter the Group Name and click the **OK** button.

Figure 2.3.19: Adding Group

2.3.5.6 Adding a User to a Group

You can add one or more users to a group as follows:

1. Select the Group to which the User is to be added.
2. Right-click on the group name and select its members.

3. Select the User that is to be added to the Group from the pop-up window. Multiple selections can be made by holding down the CTRL key.
4. Click the **OK** button.

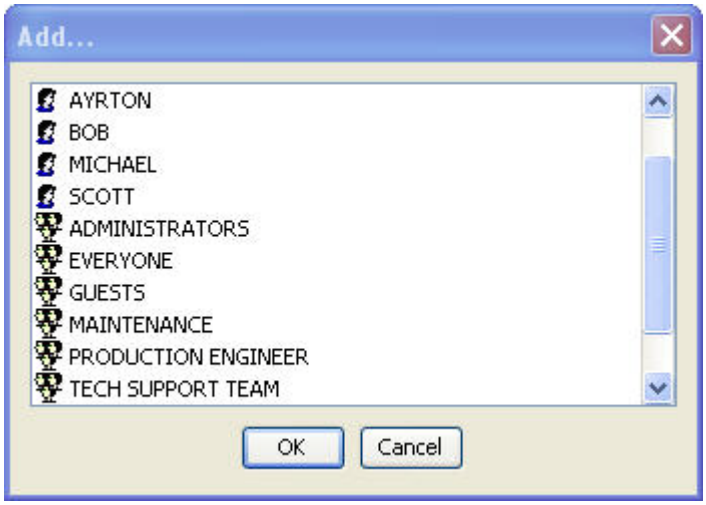


Figure 2.3.20: User List

2.3.5.7 Deleting a User from a Group

You can delete one or more User from a Group by:

1. Select the Group from which the User is to be deleted.
2. Right-click on the Group Name and select its members.
3. Select the User from the popup window and click the **Remove** button to remove the User from the Group.
4. Click the **OK** button to save the settings.

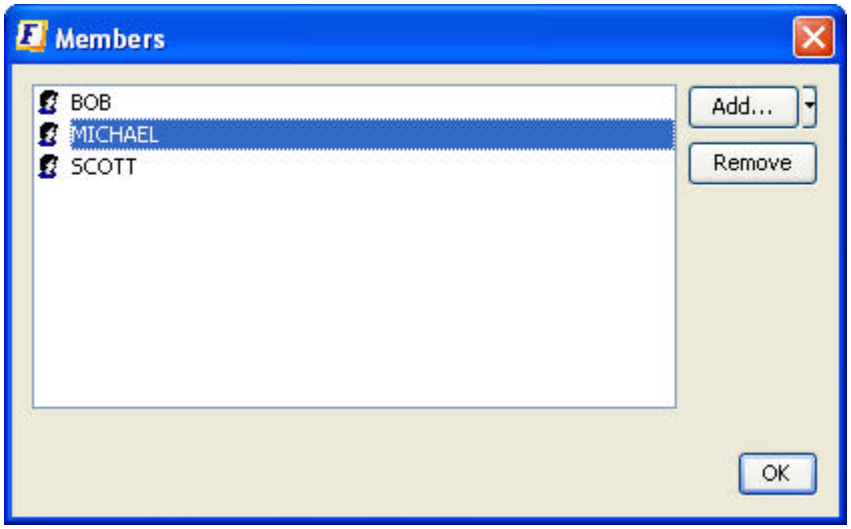


Figure 2.3.21: User List of a particular group

2.3.5.8 Deleting a Group

Any user with administrative privileges can delete a Group by logging onto the Enterprise Server.

2.3.5.8.1 To Delete a Group

1. Choose **Enterprise Server** under **Explorer** after logging onto Fiorano Studio
2. Select the security node from the Enterprise Server tree.
3. Select the Group to be deleted from the Groups. Right-click on the Group and select the **Delete** option from the pop-up menu.

Note: The deletion of ADMIN, ANONYMOUS, EVERYONE, ADMINISTRATORS, FPS, and EVERYNODE is not allowed. If an attempt is made to delete these accounts, a warning is displayed.

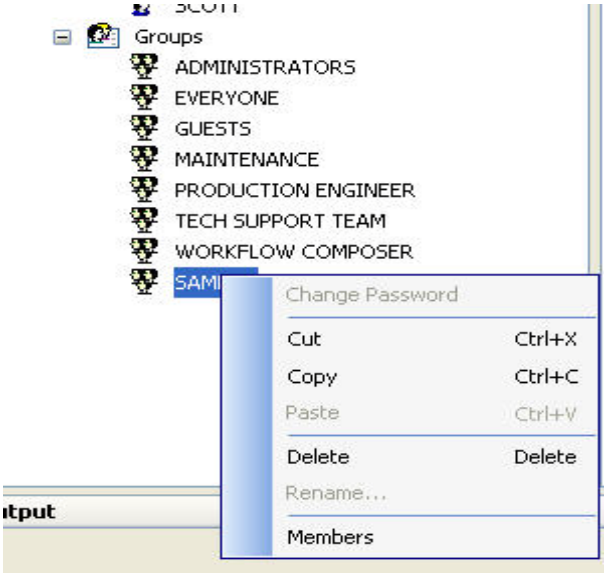


Figure 2.3.22: Menu to delete a Group/s

2.3.5.9 Setting Access Controls

Users connecting to the Fiorano Network are required to furnish their credentials which are then authenticated by the network. The authentication is performed by the Enterprise Server via the underlying Realm Component. This Realm Component is responsible for maintaining all User and User Group information as well as for authenticating any connection requests. The network administrator can choose from a collection of Realm Components, differing in storage and authentication mechanism.

This security architecture allows the administrator to set up ACLs for various resources. For example, ACLs for an Event Process can specify Users who have the privilege to launch an Event Process on the network. This allows the administrator to exercise control over the privileges available to each Users.

The following permissions can be given to a User or a User Group:

- Permission to create or delete a Principal (User and User Groups)
- Permission to compose an Event Process
- Permission to change properties of an Event Process
- Permission to terminate an Event Process
- Permission to view running and saved Event Processes
- Permission to configure an FPS
- Permission to create, update, and delete a Business Service
- Permission to create an ACL
- Permission to create, edit, and delete a Business Service ACL
- Permission to launch an Event Process

All actions that check for one or more of the above-mentioned permissions generate a security event. Permissions can be requested by any principal registered on the Fiorano Network. The Fiorano Studio allows the administrator to set access rights for individual Users.

The security module in the Fiorano Network resides within the Enterprise Server. The security architecture allows this module to be plugged, which in turn allows the enterprise administrator to choose a Realm Module from a list of modules provided by the Fiorano SOA Platform.

2.3.5.10 Assigning Rights

The FSSM (Fiorano Services and Security Manager) tool is used to assign rights to Users and to User Groups. Rights may be understood as rules associated with the Fiorano Network that are granted to Users and User Groups. They allow Users and User Groups to perform specific tasks on the Fiorano Network. The Fiorano SOA Platform has a well-defined security policy to protect the network against data loss or corruption due to malicious or accidental access. This policy is implemented by assigning appropriate permissions to Users and User Groups thereby preventing illegal access to the Fiorano Network.

When the **Access Rights Assignment** in the left-hand-side panel is selected, a list of all available permissions is displayed in the right-hand side panel, as shown in the Figure 2.3.23.

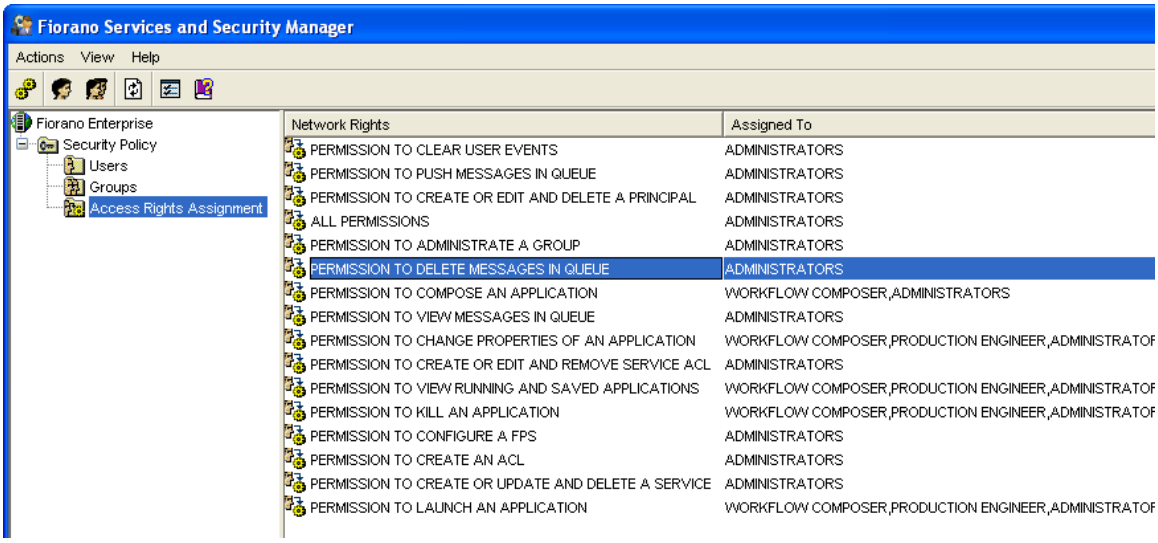


Figure 2.3.23: Realms Description

The right panel displays the following Network Rights:

- **Permission to create or delete a principal:** This permission allows a User or a User Group to create, edit, and delete Users and User Groups. Users and/or User Groups with this permission have the right to change passwords.
- **Permission to compose an Event Process:** This permission allows a User and/or a User Group to create new Event Processes using Fiorano Studio.
- **Permission to change properties of an Event Process:** This permission allows a User and/or a User Group to change the basic and advanced properties of the Event Process from the Event Process property sheet in Fiorano Studio.
- **Permission to view running and saved Event Processes:** This permission allows a User and/or a User Group to run Event Processes in the Fiorano Event Manager.
- **Permission to terminate an Event Process:** This permission allows a User and/or a User Group to terminate Event Processes from the Fiorano Studio.
- **Permission to configure a FPS:** This permission allows a User and/or a User Group to create, edit, and delete a Fiorano Peer Server using the Fiorano Network Administration tool.
- **Permission to create, update, and delete a Business Service:** This permission allows a User and/or a User Group to create, update, and delete Business Services using Fiorano Studio.
- **Permission to create an ACL:** This permission allows a User and/or a User Group to set access control on Fiorano Components.
- **Permission to create, edit and delete Business Service ACL:** This permission allows a User to set access control for Fiorano Components. With this permission, the User can specify the nodes on which a Fiorano component can run.
- **Permission to launch an Event Process:** This permission allows a User and/or a User Group to launch Event Processes.

2.3.5.10.1 To Assign Rights

FSSM can be used to assign rights to both Users and User Groups. To assign rights to a User, perform the following steps:

1. In the right-hand side of the panel, right-click on the field corresponding to the **PERMISSION TO KILL AN APPLICATION** option.
2. Click the **Properties** option. The **Access Control** dialog box is displayed, as shown in the Figure 2.3.24.
3. Click the **Add** button, select the User and click the **OK** button. The user is assigned the permission to kill an Event Process.

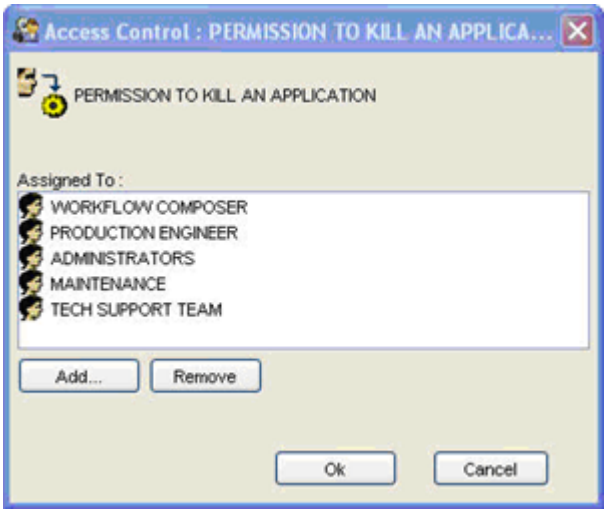


Figure 2.3.24: Access Control Dialog Box

2.3.5.11 Removing Network Rights

FSSM can be used to revoke permissions assigned to Users and User Groups. To do this, the User or User Group to whom the permission has been assigned should be deleted, as follows:

1. In the right-hand side panel, right-click the field corresponding to the **PERMISSION TO CLEAR USER EVENTS** option.
2. Click the **Properties** option. The **Access Control** dialog box is displayed.
3. Select the User and click the **Remove** button to delete the User from the list of Users assigned the permission to clear User Events.
4. Click the **OK** button to register the deletion of the user from the list of users assigned the permission to clear user events.

2.3.6 Clearing ESB Server Database

To clear the FES server database of the default profile (that is profile1), run or double-click the script **clearDBServer.bat/.sh –mode fes** available under <fiorano_installation_dir>\esb\server\bin directory.

To clear the FES server database of a profile other than the default profile, run the script **clearDBServer.bat/.sh** available under <fiorano_installation_dir>\esb\server\bin folder with the profile option as shown below:

clearDBServer.bat/.sh –mode fes –profile <profilename>

The following operations are available when this script is executed.

Select the datastore to clear:

1. **File Based Datastore** – Clears the local cache of the Enterprise Server including stored logs.
2. **Admin Datastore –** Clears the admin objects, that is, JMS Connection factories, queue and topic destinations, status of running Event Processes and component instances.
3. **Peer Repository** – Clears all the fetched peer server profiles from Enterprise Server runtimedata.
4. **Events Database** – Clears the Events Database using the configurations provided in eventsdb.cfg file present under: <fiorano_installation_dir>/esb/server/profiles/<profilename>/FES/conf directory.
5. **SBW Database** – Clears the SBW database using the configurations provided in the sbwdb.cfg file present under: <fiorano_installation_dir>/esb/server/profiles/<profilename>/FES/conf directory.

   The Enterprise Server processes System events, SBW events and Backlog events and takes appropriate actions. System events and SBW events are queued up to be inserted into an external database while Backlog events are queued up to be handled by various alert handlers. Before this processing happens, events are temporarily stored in persistent database that are created during runtime data of the Enterprise Server. After an event has been processed, it gets deleted from the temporary store. If these events are not able to be processed, the temporary datastore may grow to occupy a large amount of disk-space. Option 7, 8, and 9 can be used to delete the temporary persistent datastore of different events.
6. **Events Persistent Database** – Clears the temporary persistent datastore of system events.
7. **SBW Persistent Database** – Clears the temporary persistent datastore of SBW events.
8. **Backlog Persistent Database** – Clears the temporary persistent datastore of backlog events.
9. **All** – Clears all nine of the above.

This script can be executed in Quiet Mode as follows.

Example usage:

clearDBServer.bat -mode fes -profile profile1 -dbPath <DB Directory path> -q 1,2,4

- **-mode** - to clear fps or fes runtimedata
- **-dbPath** - runtime data directory for the profile
- **-profile** - profile name for which runtimedata is to be cleared

- **-q** - to run the script in quiet mode.

Note: Provide comma separated option values to this argument. Absence of any argument will lead to the default option; option 10, '**ALL'**.

2.4 Fiorano Peer Server

The Fiorano Peer Server acts a container for launching Business components at the network endpoints of a Fiorano network and manages the life cycle of its components.

2.4.1 Peer Server Functionality

The following list consists of the roles of the Fiorano Peer Server:

- Acts as the runtime container for the components.
- Routes the data between various components in a Peer to Peer fashion over JMS routes.
- Routes Business component related control and State information to the FES server.
- Provides store and forward capabilities to handle network failures, thus providing guaranteed delivery of messages (which flow across from one peer to another). A Fiorano Peer server has inherent storage mechanisms and forwarding mechanisms which enable each peer to stores messages corresponding to all Peer Servers that are unavailable at the moment and forward these to Peer Servers as and when they comes up again. This allows the Event Processes to continue execution even in the event of a remote machine failure or a network failure.
- Generating System and User Events
- Generating Logs

2.4.2 Launching the Peer Server

The FPS can be launched from the Windows Start menu or by directly executing a script file. When the peer server is launched, the enterprise server must be running on the configured machine for successful peer server startup. Please refer to section 2.4.4.4 Changing to different ESB Network for instructions on how to configure the Enterprise Server URL for a Peer Server. If the Enterprise Server is not reachable by the Peer Server or if it is not running, the Peer Server keeps waiting until it can reach the Enterprise Server on a specific connection URL. The Enterprise Server starts successfully when the Peer Server is able to establish a connection with the Enterprise Server.

2.4.2.1 Launching the Peer Server From Fiorano Studio

Click Start→Programs→Fiorano→Fiorano SOA Platform → Fiorano Servers→Fiorano ESB Peer

2.4.2.2 Launching the Peer Server Using Script Files

- To start the FPS server with the default profile (which is profile1), run the script *server.bat/.sh –mode fps* available under <fiorano_installation_dir>\esb\server\bin folder.

  Note: The server by default starts in FPS mode if mode argument is not provided.

- To start the FPS server with a specific profile other than the default profile, run the script *server.bat/.sh* available under <fiorano_installation_dir>\esb\server\bin folder with the profile option as shown below:
  - server.bat/.sh –mode fps –profile <profilename>

Example: <profileName>=haprofile1/primary or <profileName>=haprofile1/secondary when servers are to be started in HA mode.

Note: In UNIX systems, servers will by default start in the background mode. Pass runtime argument –**nobackground** to the server startup script to run the server in console mode. If the servers are not to be run in background mode, modify the file %FIORANO_HOME%/launcher/server.sh so that **nobackground**="" and is changed to **nobackground**="true".

To install FPS as a Windows NT service, run the command:

Note: NT services will have to be re-installed if changes are made to any configuration files. This includes fiorano_vars as well as server.conf.

install-server.service.bat –mode fps -profile %PROFILE_NAME%

To remove FPS as a Windows NT service, run the command:

uninstall-server.service.bat –mode fps -profile %PROFILE_NAME%

To install/uninstall FPS as a service on UNIX, refer to *readmeWrapperService.txt* present under %INSTALL_DIR%/SOA/esb/server/bin/service directory.

2.4.3 Shutting Down the Peer Server

The FPS can be stopped from the Fiorano Studio or by directly executing a script file. Shutting down the Peer Server automatically invokes a shutdown hook in the peer server's JVM which cleans up any resources and connections used by the server's JVM.

2.4.3.1 Using Fiorano Studio

FPS can be stopped only after logging into FPS from the Fiorano Studio. The Figure 2.4.1 shows the Shutdown menu item location:

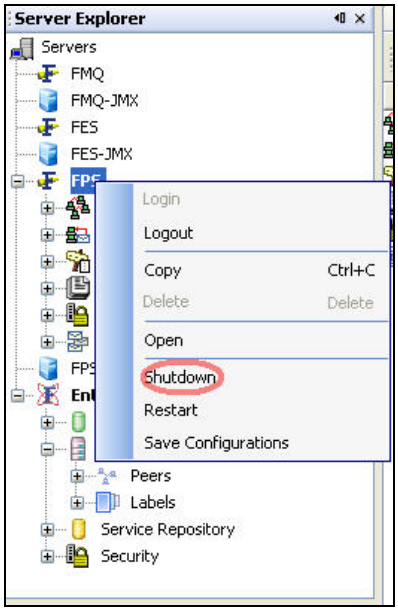


Figure 2.4.1: Stopping FPS Server

2.4.3.2 Using Script Files

The Peer Server can be shutdown using the **shutdown-server** script present in <fiorano_installation_dir>/esb/server/bin folder.

Note the following:

- Shutdown scripts cannot be used to shutdown both primary and secondary servers running in shared HA mode simultaneously using a RMI connection.
- If HA profiles from previous versions are migrated to current version of the product, these profiles need to be re-configured to specify BackupRMIServerPort properties to make use of this functionality.

This script can be used to:

- Shut down a standalone peer server.
- Shut down both the primary and secondary HA servers simultaneously.

Options that can be specified within the script are:

- **-user**: Name of the User trying to shutdown Fiorano server
- **-passwd**: Password of the User trying to shutdown Fiorano server
- **-restart** or **–r**: Restarts the Fiorano Server
- **-ha**: Used to shutdown both active and passive servers running in HA.
- **-mode**: mode of the server, that is, fes or fps. Defaults to value fps.
- **-url**: URL of the active Fiorano Enterprise Server to which the Peer Server has connected
- **-fpsname**: Fiorano ESB Peer name
- **-?** or **–help**: Prints help message

Examples:

To shut down a standalone Peer Server

shutdown-server -url tsp_tcp://localhost:1947 -user admin -passwd passwd -fpsname fps

To shut down both active/passive Peer Servers in HA

shutdown-server -url tsp_tcp://localhost:1947 -user admin -passwd passwd -fpsname hafps -ha

The above examples of shutting down the Peer Server(s) involves connecting to an Active Enterprise Server, whose url is specified by the **-url** parameter.

The following options can also be specified if a User wants to shutdown the Peer Server using a RMI connection. This involves connecting directly to the Peer Server to initiate the shutdown.

- **-connectorType**: Connection type to server. Pass RMI for shutting down servers using RMI connection.
- **-address**: IP Address of server
- **-rmiPort**: RMI port of server

Note: If the **-ha** option is given to shutdown both the servers of an **ha** pair, the **address** and **rmiPort** options can be specified as either the Primary or the Secondary server. This can be used if not certain of which server in the HA pair is the active one.

Examples:

To shut down standalone server

shutdown-server -connectorType RMI -user admin -passwd passwd -address localhost -rmiPort 2067

To shut down both active/passive servers in HA

shutdown-server -connectorType RMI -user admin -passwd passwd -address localhost -rmiPort 2077 -ha

2.4.4 Peer Server Configuration

FPS can be configured either in online mode (that is, while the FPS server is running) or offline mode (when the FPS server is not running) using the Fiorano Studio.

Offline mode configuration can be done using the Profile Manager, which can be accessed from the Fiorano Studio. When a configuration is modified and saved, changes persist and are applied when the server is restarted.

Online configuration of a Server can be performed using the JMX explorer, which can be accessed from the Fiorano Studio. Changes made using JMX are applied onto the Server that is currently running and persist so as to be reapplied when restarting. However, some of the Server configurations (such as server ports, memory settings and others) are applied only after the restart of the Server.

Note: For more information on configuring Fiorano Servers, please see section 2.6 Configuring Servers and Tools.

2.4.4.1 Server Ports Configuration

The FPS server communicates with the Enterprise Server using the Server Port and with the JMX interface through the RMI Port. The Configurations of these ports is described in more detail below.

2.4.4.1.1 Server Ports

Ports opened to facilitate communication between FPS and Enterprise server are known as Server Ports.

Configuration Steps

Configuration of Server Ports Using the Offline Mode

5. Open the **FPS** server profile in Studio (Tools→Configure Profile). This FPS server profile can be selected from the location shown in the Figure 2.4.2.

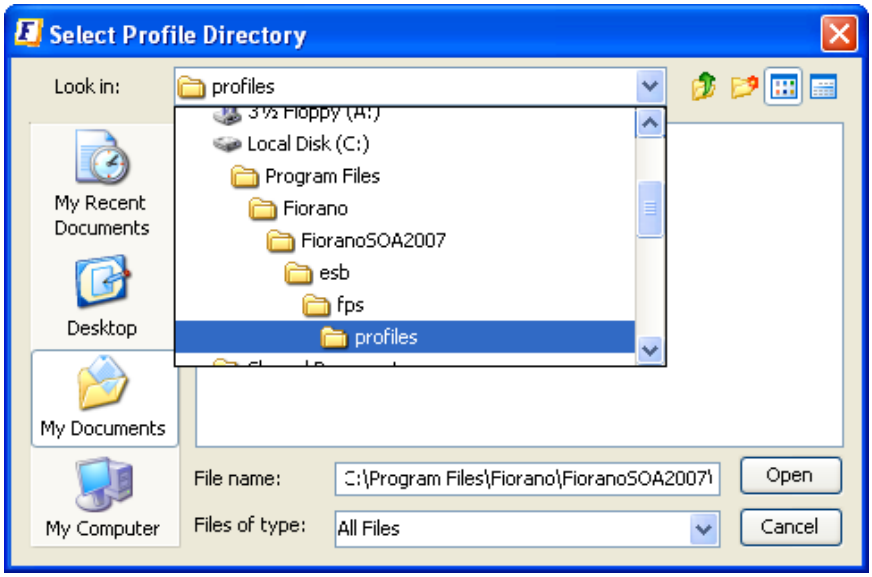


Figure 2.4.2: Profile Selection

6. Select the **ConnectionManager** in FPS→Fiorano→SocketAdapters→port-1→ConnectionManager. In the **Properties of the ConnectionManager** panel displayed on the right-hand side, change the **Port** property.

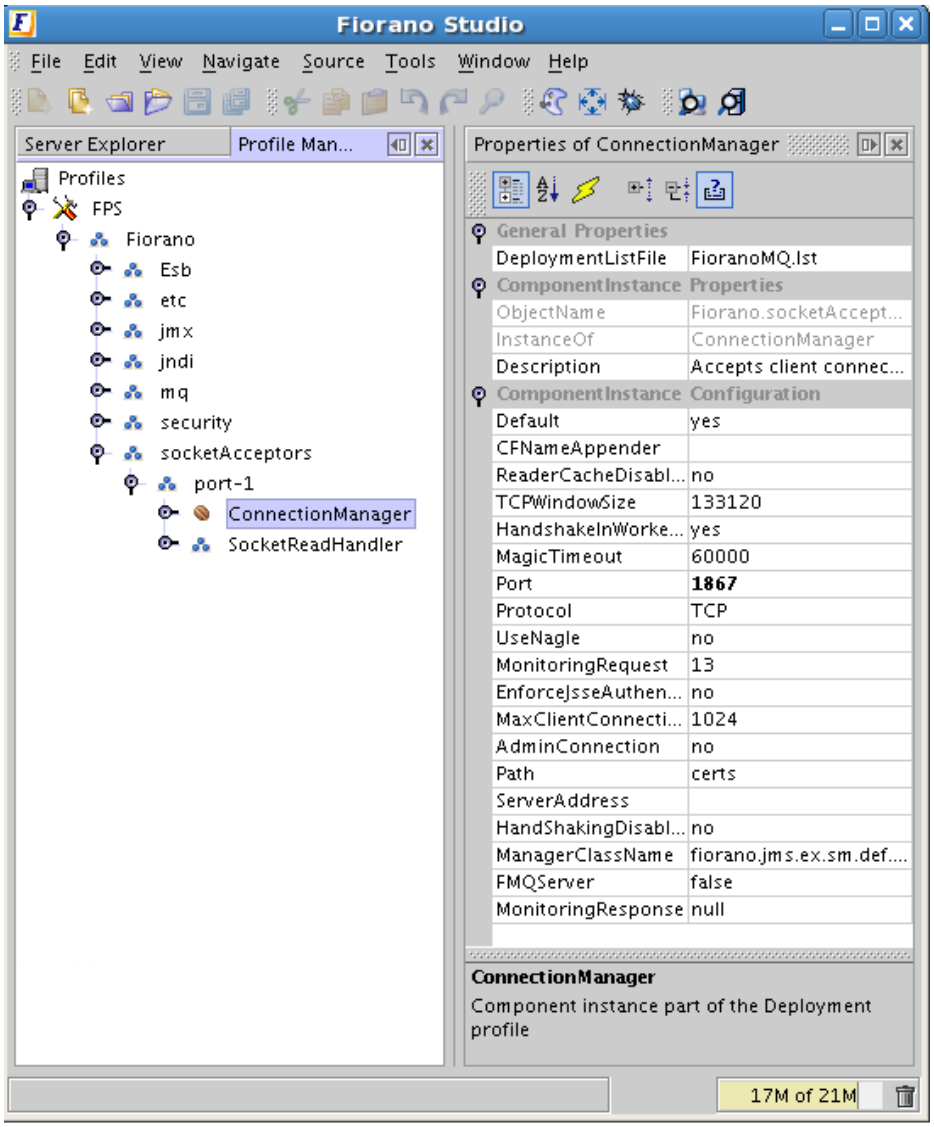


Figure 2.4.3: FPS Server Port

Note: The Peer Server connects to the Enterprise Server through this port after FPS is started with this profile saved.

7. Save the profile as shown in the Figure 2.4.4.

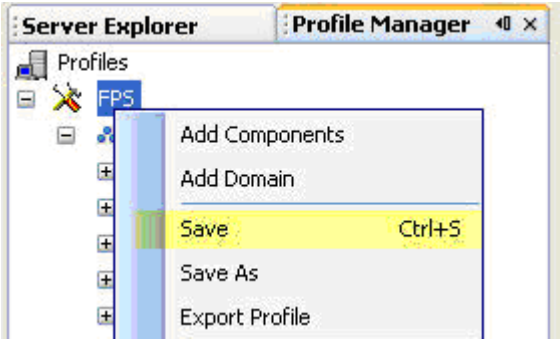


Figure 2.4.4: Save Profile

Configuration of Server Ports Using the Online Mode

1. Connect to the Fiorano Peer Server's JMX interface through Fiorano Studio as shown in Figure 2.4.5.

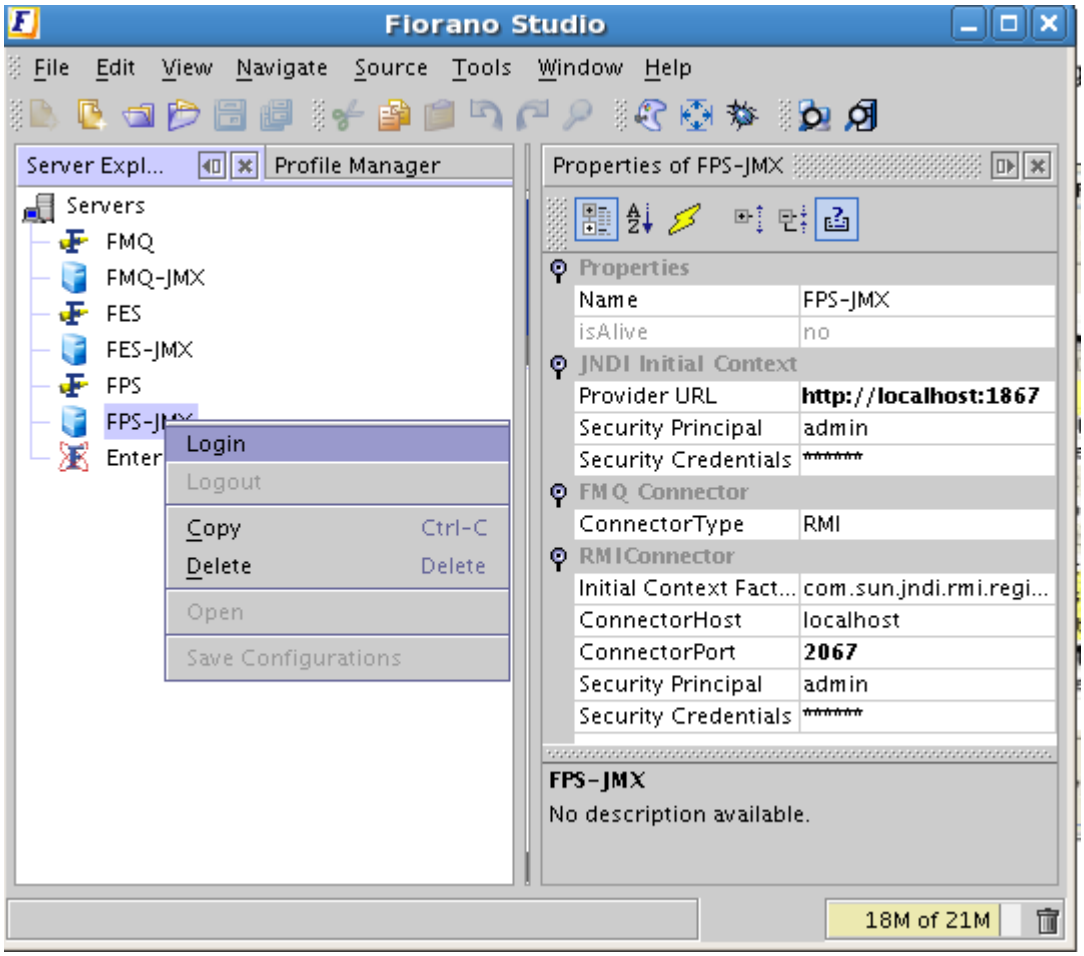


Figure 2.4.5: Login into Fiorano Peer Server's JMX interface

2. Select **JMX Connection** → Fiorano → socketAcceptors → ConnectionManager → **ConnectionManager**. In the **configuration properties** panel displayed on the right-hand side, change the **Port** property to the new port.

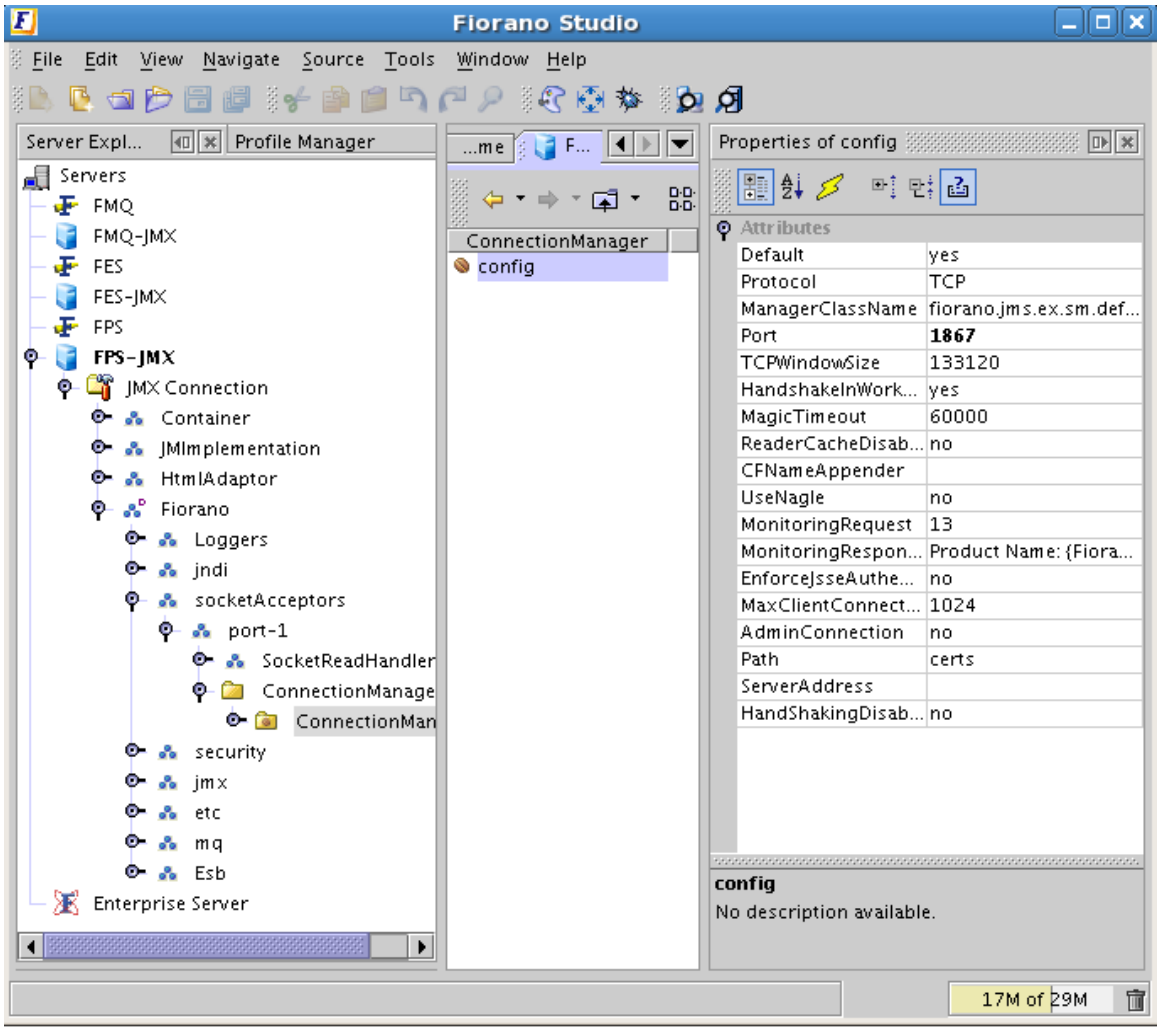


Figure 2.4.6: JMX Explorer

3. The server needs to be restarted after the value is set. A message dialog box appears with instructions for properties that require the server be restarted, as shown in Figure 2.4.7.

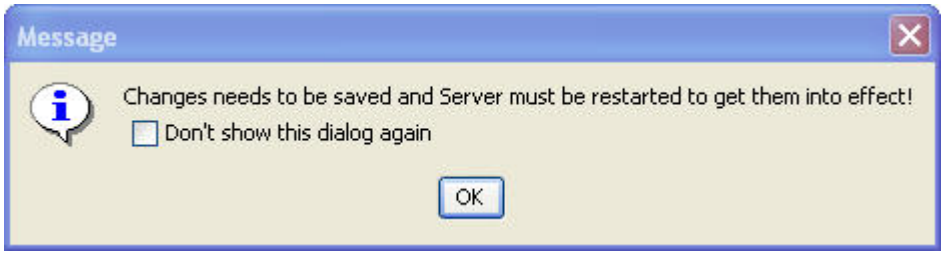


Figure 2.4.7: Message dialog box

4. Click the **OK** button to save the configurations. These configurations are applied after the server is restarted.

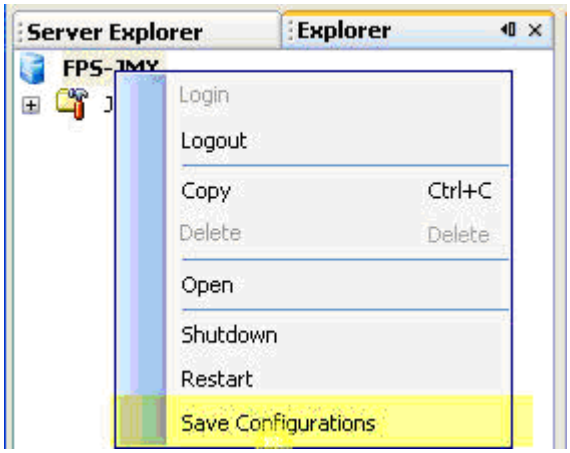


Figure 2.4.8: Save Configurations

2.4.4.1.2 RMI Ports

Ports that are opened to facilitate the communication between FPS with JMX clients are known as RMI Server Ports.

Configuration Steps

RMI ports can be configured in either offline or online mode.

Configuration of RMI Server Ports Using the Offline Mode

1. Open the server profile in Studio (tools → configure profile) as explained in section Server Ports.
2. Select the **RMIBasedJMXConnector** from FPS → Fiorano → JMX -> Connector. In the **Properties of RMIBasedJMXConnector** panel displayed on the right-hand side, change the **RMIServerPort** property.

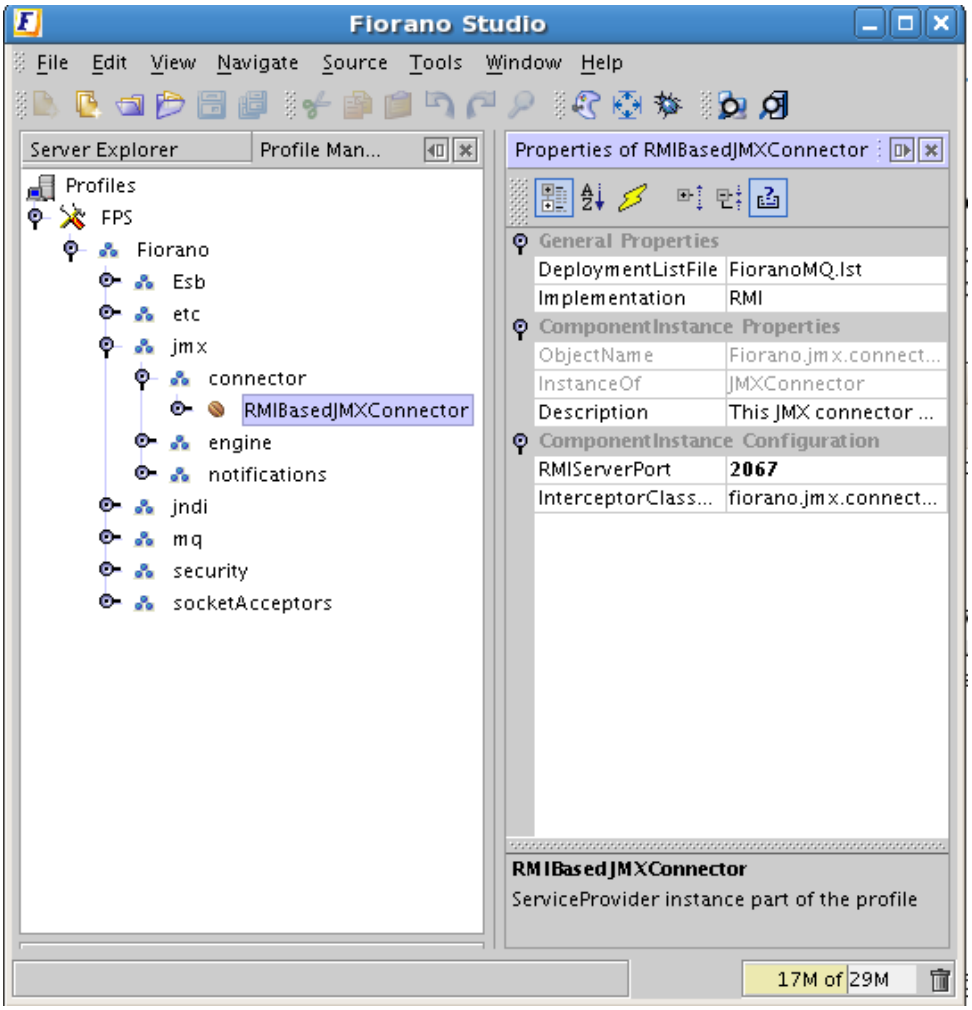


Figure 2.4.9: FPS RMI Port

3. Save the profile as explained in section 2.4.4.1.1 Server Ports.

Configuration of RMI Server Ports Using the Online Mode

1. Connect to the Fiorano Peer Server's JMX interface through Fiorano Studio as explained in section 2.4.4.1.1 Server Ports.
2. Select JMX Connection → Fiorano → jmx → connector → JMXConnector → RMI → RMIBasedJMXConnector. In the **Properties of Config** panel displayed on the right hand side, change the **RMIServerPort** properties.

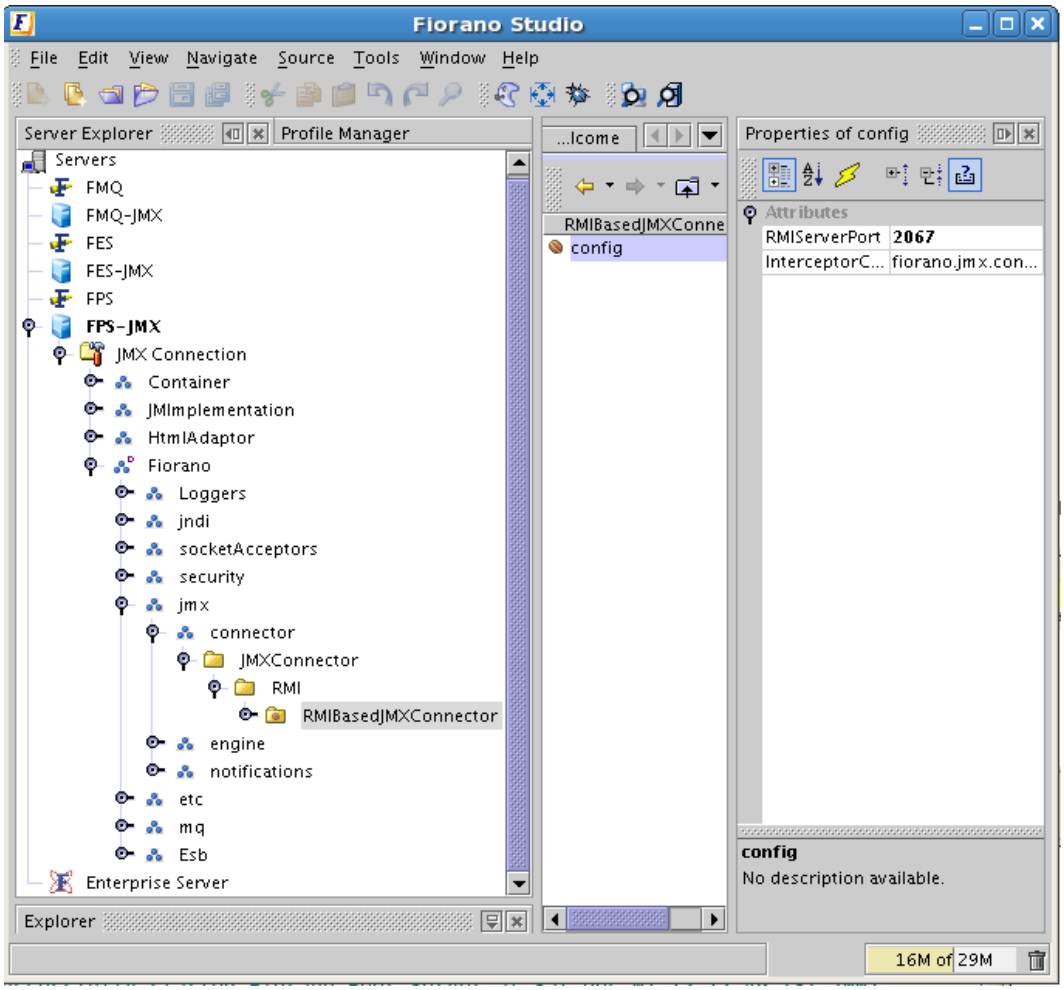


Figure 2.4.10: JMX Explorer

3. The server must be restarted after the value is set. A message dialog box is displayed with instruction for properties that require a server restart as explained in section 2.4.4.1.1 Server Ports.
4. Click the **OK** button to save the configurations. These configurations are reflected after the server is restarted as explained in section 2.4.4.1.1 Server Ports.

2.4.4.2 Memory Configurations

Better server performance is possible with proper configuration of JVM parameters, particularly those related to memory usage. The allocation of memory for the JVM is specified using -X options when starting the server.

JVM option	Meaning	Default ESB Settings
-Xmx	Maximum heap size	512MB
-Xms	Initial heap size	256MB
-Xss	Stack size for each thread	JVM default (120k)

Note: The stack size limits the number of threads that can run on a given JVM; a large stack size may result in memory running short as each thread is usually allocated more memory than it needs.

2.4.4.2.1 Configuration Steps

1. Open the server.conf in %FIORANO_HOME%/esb/server/bin/ and change the values for –Xms and –Xmx argument under <jvm.arg> tag. If no value for –Xss is specified, the default value will be used.

   Note: This parameter is used by both FES and FPS servers. If settings different from the default settings are to be specified to the FPS, then the files server.conf and server.sh/.bat need to be copied and renamed them, so that both files have the same name (for example, serverFPS.conf and serverFPS.sh/.bat). Only when this is done, different memory setting in *server.conf* and *serverFPS.*conf files can be set. Save the file and restart the server.

2.4.4.3 Java Configurations

The Peer Server requires JRE version 1.5 or above to be used for successful operation. The JAVA_HOME setting can be configured for the Enterprise Server as follows:

On UNIX:

The Peer Server by default uses JAVA_HOME value set for the console. This can be overridden by specifying the JAVA_HOME value in **%FIORANO_HOME%/fiorano_vars.sh** file.

On Windows:

JAVA_HOME is by default set to %FIORANO_HOME%/jre1.5.0_16 (shipped with the product) in **%FIORANO_HOME%/fiorano_vars.bat** file. To make the server use a different JAVA_HOME, this setting needs to be modified.

2.4.4.4 Changing to different ESB Network

This section provides information on configuring a Peer Server onto a different ESB Network.

Offline Mode

- Open the FPS server profile in Studio (Tools->Configure Profile). This FPS server profile can be selected from the following location as explained in section 2.4.4.1.1 Server Ports.
- Configure the Enterprise Server properties from: Fiorano->Esb->Peer->Transport->EnterpriseBus->EnterpriseServer.

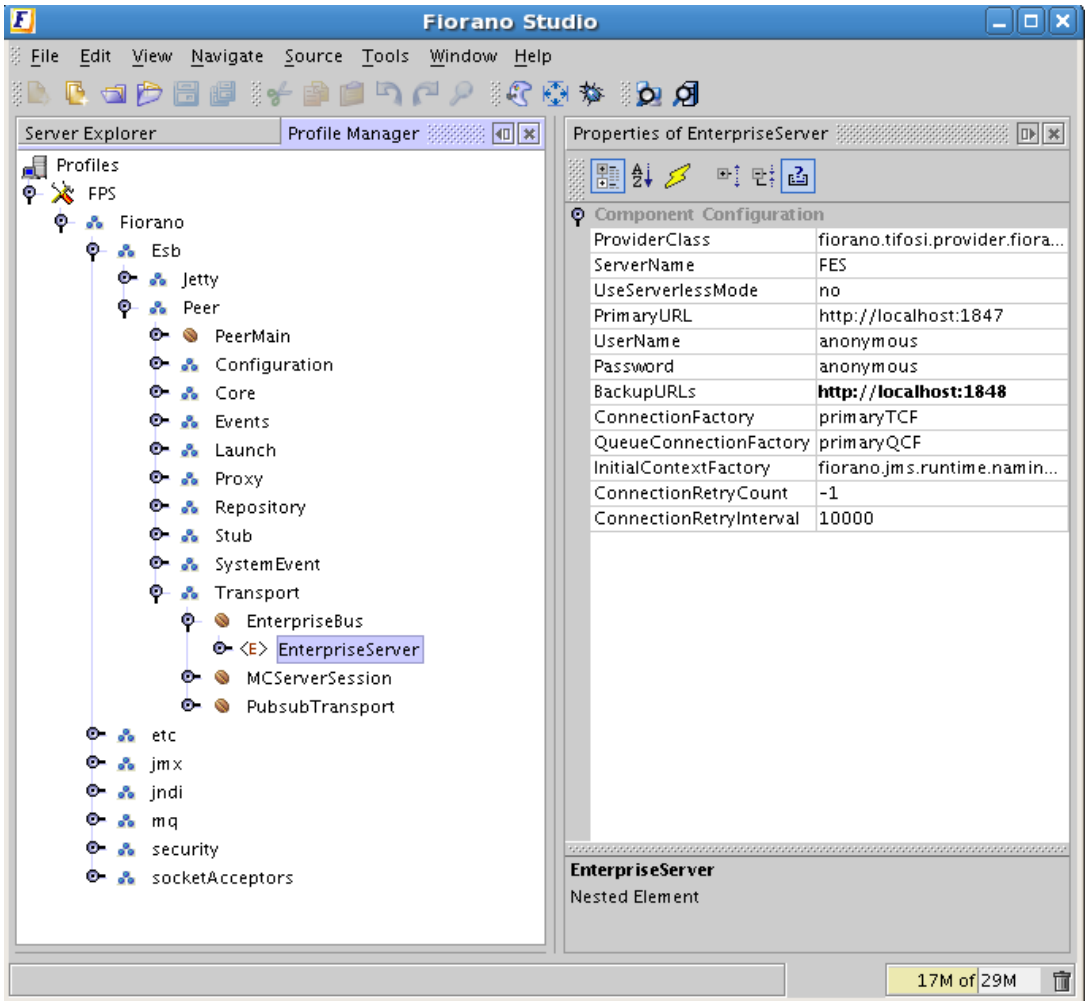


Figure 2.4.11: Configuration of Enterprise Server properties

The following table provides the details of the transport configuration attributes:

Attribute Name	Description	Default Value
ServerName	Name of the Fiorano Enterprise Server to connect to	FES
PrimaryURL	Primary URL of the FES server	http://<FES Primary IP>:1847
BackupURLs	Backup URLs separated by semi-colons are used when FES at the primary URL is not available.	http://<FES Secondary IP>:1848
Username	Username to be used to create a connection with the FES	Anonymous
Password	Password to be used while connecting to the FES	Anonymous
ConnectionFactory	TopicConnectionFactory to be used to create the JMS Connection with the Enterprise Server	primaryTCF
QueueConnectionFactory	QueueConnectionFactory to be used to create the JMS Connection with the Enterprise Server	primaryQCF
ConnectionRetryCount	Number of times the Peer should try to connect to the FES if the connection is not available	1 implies infinite retries - -1

2.4.5 Adding New Peer Server

To add a new Peer Server to the Fiorano Network, follow the steps below:

1. Open the default FPS profile.
2. To connect to the **Enterprise Server** on the same machine, the profile name and the port numbers of the **JMS** port, **RMI** port, and **Jetty** port should be changed from the default profile port to the ports which are not used by any other servers or processes.
3. To change the profile name, change the **Profile Name** property as shown in the Figure 2.4.12.

   Note: The profile name is valid if all of its letters satisfy any or all of the following criteria:

a. A letter

b. A currency symbol (such as '$')

c. A digit

d. A numeric letter (such as a Roman numeral character)

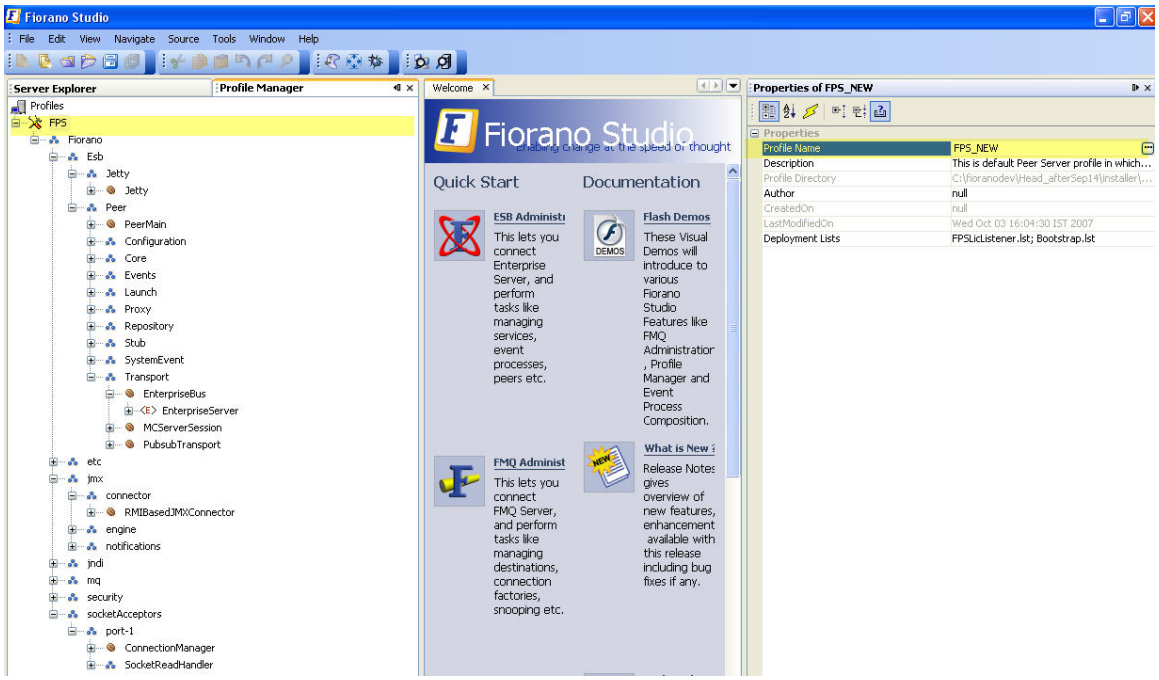


Figure 2.4.12: Profile Name property

4. To change the **JMS** port, refer to the Figure 2.4.13 and change the Port property in Fiorano → Socket Adapters → port-1 → ConnectionManager.

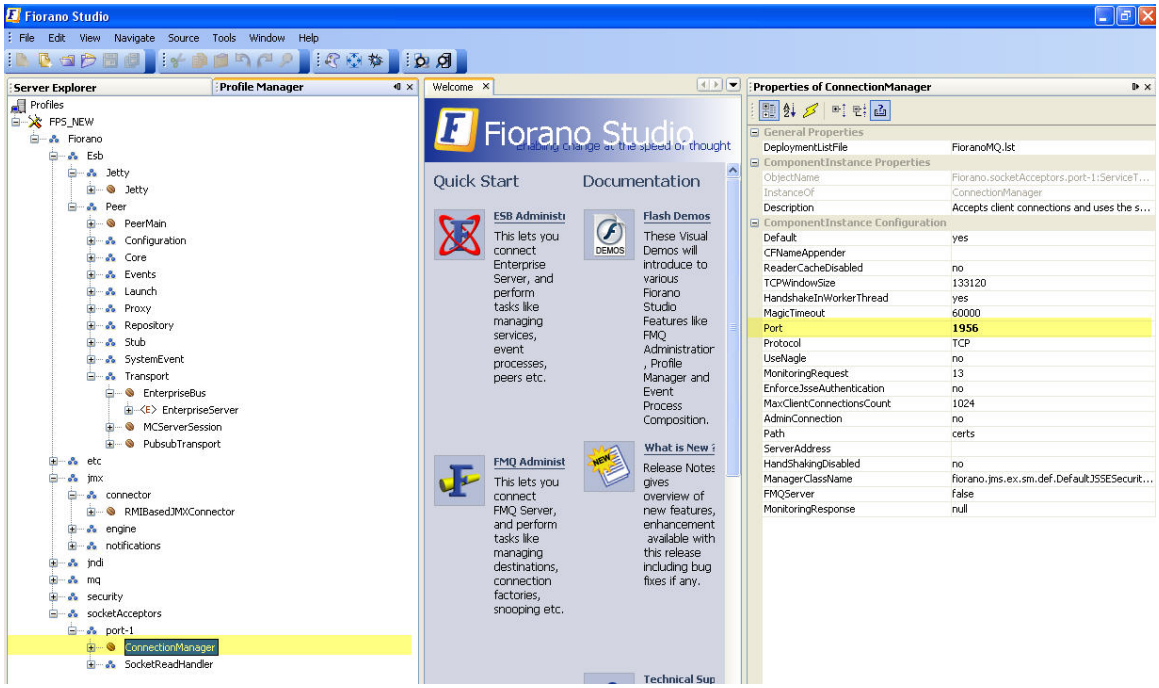


Figure 2.4.13: ConnectionManager

5. To Change the **RMI** port, refer to the image 2.4.14 and change the RMIServerPort property in Fiorano → jmx → connector → RMIBasedJMXConnector.

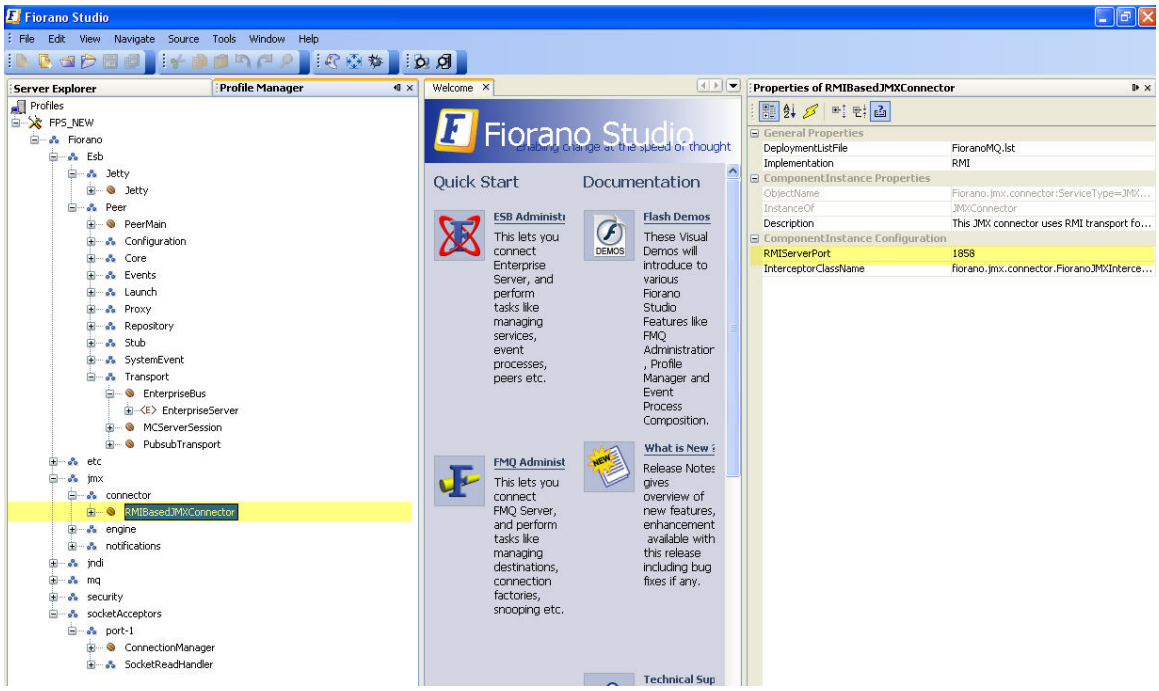


Figure 2.4.14: Changing RMI port

6. To change the **Jetty** port, refer the Figure 2.4.15 and change the PortNumber property in Fiorano → Esb→ Jetty → Jetty.

Figure 2.4.15: Changing Jetty port

7. If the profile is to be connected to the Enterprise Server running on some other machine, in addition to the above properties the Enterprise Server properties should be changed in the profile, as shown in the Figure 2.4.16.

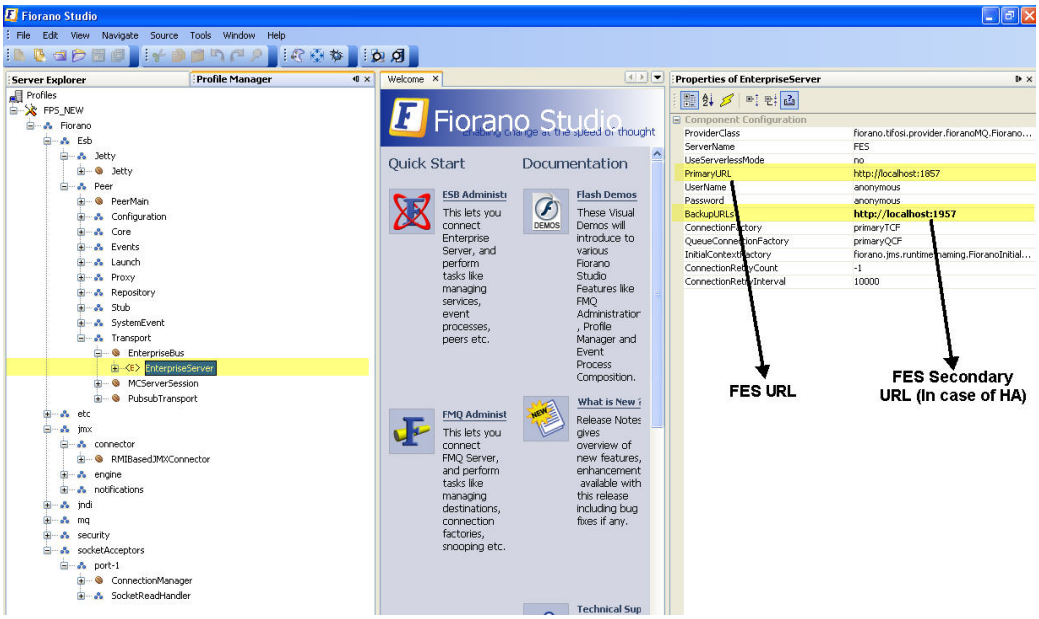


Figure 2.4.16: Enterprise server properties

8. After changing all the properties, save the profile with a new profile name. To save the profile, right-click on the profile and select the **Save As** option from the pop-up menu. The **Save As** dialog box appears with default save location as that of the profile which is being saved. You can choose to change the profile location to any location inside the profiles directory, that is, $FIORANO_HOME/esb/server/profiles. It is recommended that the new profile location must end with a directory named FPS (to indicate that it is a Peer Server profile). For example, a new profile can be saved as: $FIORANO_HOME/esb/server/profiles/newProfile/FPS.

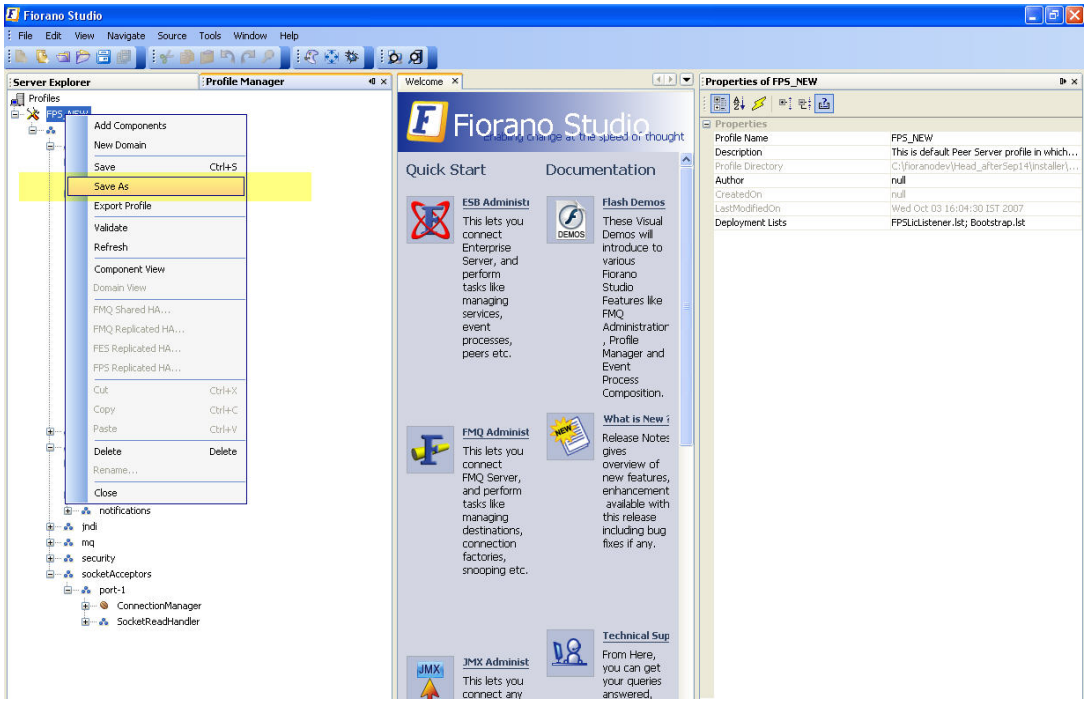


Figure 2.4.17: Saving the profile

2.4.6 Clearing Peer Server Database

To clear the Peer Server database of the default profile (that is profile1), run the script **clearDBServer.bat/.sh –mode fps** available under <fiorano_installation_dir>\esb\server\bin folder.

Note: The script by default clears the database of the Peer Server if an argument mode is not provided.

To clear the Peer Server database of a specific profile other than the default profile, run the script **clearDBServer.bat/.sh** available under <fiorano_installation_dir>\esb\server\bin folder with the profile option as shown below:

clearDBServer.bat/.sh –mode fps –profile <profilename>

The following operations are available when this script is executed.

Select the datastore to clear:

1. **File Based Datastore** – Clears the local cache of the Enterprise server including logs.
2. **Admin Datastore –** Clears the admin objects, that is, the JMS connection factories queues and topic destinations.
3. **Cached Component Store -** Clears all the cached components stored in the Peer Server's runtime storage.
4. **All** – Clears all four of the above.

This script can be executed in Quiet Mode as follows.

Example usage:

clearDBServer.bat -mode fes -profile profile1 -dbPath <DB Directory path> -q 1,2

- **-mode** - to clear fps or fes runtimedata
- **-dbPath** - runtime data directory for the profile
- **-profile** - profile name for which runtimedata is to be cleared
- **-q** - to run the script in quiet mode.

Note: Provide the comma separated option values to this argument. Absence of any argument will to the default option, option 4, **'ALL'**.

2.5 Fiorano Web Console

Fiorano Webcosnole provides a web based monitoring tool for the Fiorano ESB Network. It also provides support for launching, stopping and restarting an application using a web interface.

2.5.1 Login Page

Fiorano ESB Webconsole can be accessed by starting the Fiorano Enterprise Server (FES) and then opening http://localhost:1980/ESBDashboard on a web browser. Alternatively, the user can access the **Fiorano Web Container** from the link present in the welcome page at http://localhost:1980. The user can login onto the Fiorano ESB Webconsole by entering user name and password configured for the FES. The default **Username** is admin and the **Password** is passwd.

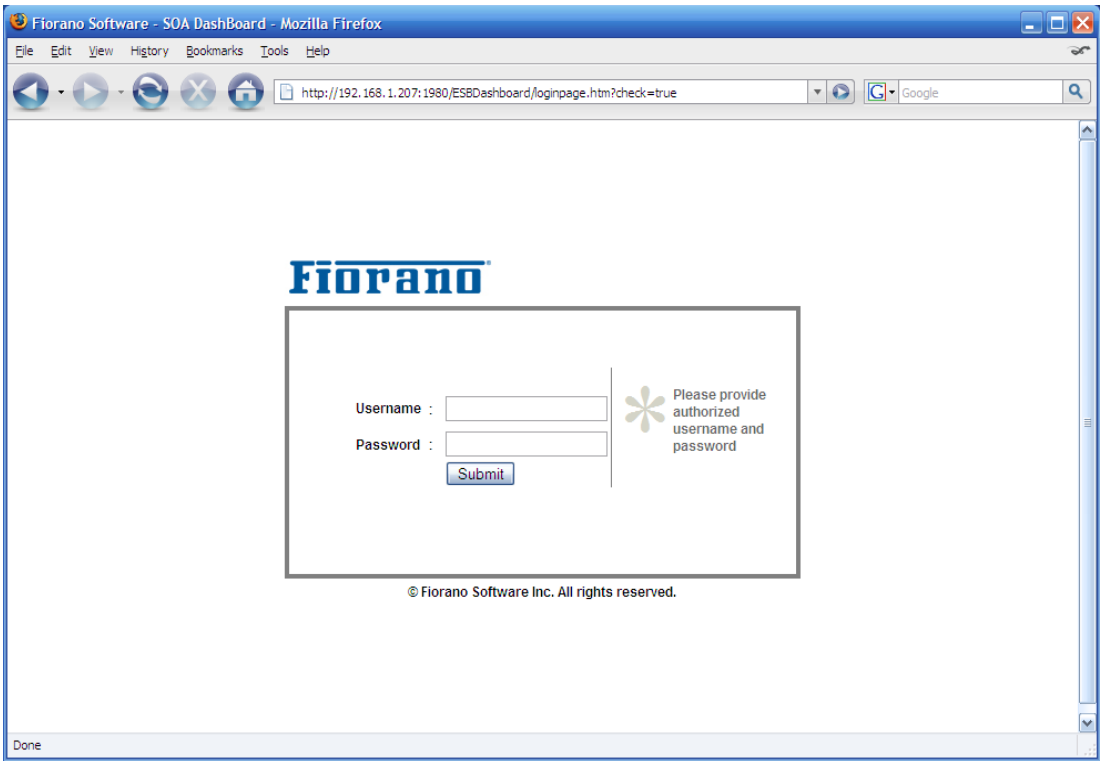


Figure 2.5.1: SOA Web Console Login Page

The Fiorano SOA Web Console has six different sections, grouped logically, based on data presented:

- Events
- Applications
- Server Status
- Document Tracking
- Web Services
- Resource Search

2.5.1.1 Events

The Events section gives the details of the events generated by both the Fiorano Enterprise Server (FES) and the Fiorano Peer Server (FPS). In addition, it also shows all the SBW exceptions that occurred while running various event processes.

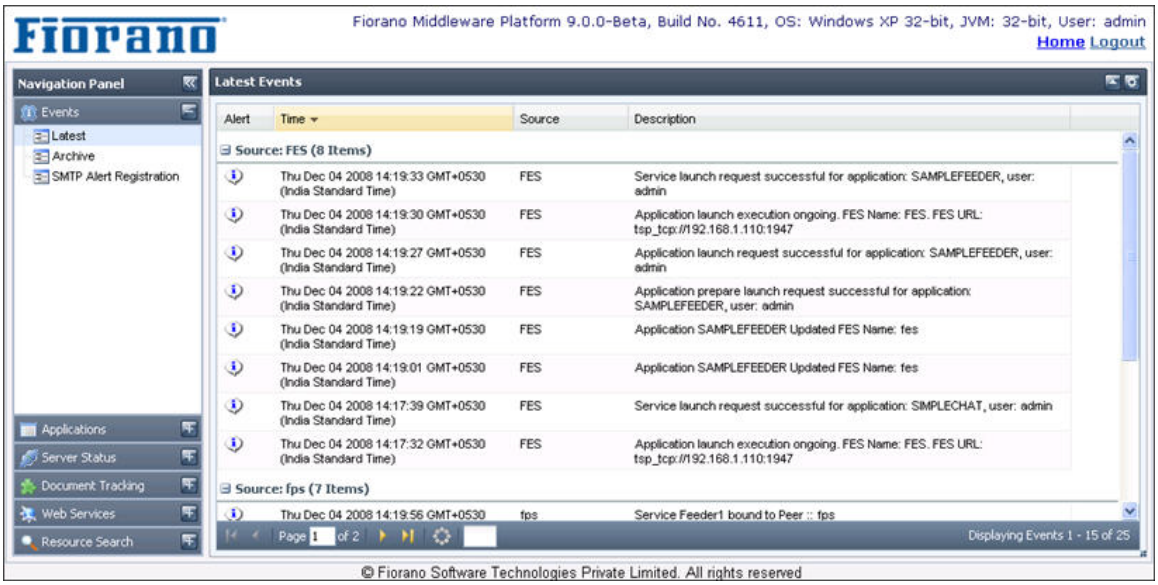


Figure 2.5.2: Events tab showing the latest events

The Event tab has three sections:

1. Latest
2. Archives
3. SMTP Alert Registration

Latest: This gives the list of the latest Events generated by FES and FPS. The visible Events can be filtered using the Event Type and Event Category options. Refer to Figure 2.5.3

To open Filter Event, perform the following:

1. Click the Events tab form the **Navigation Panel** and choose Latest option. The list of latest Events generated by FES and FPS **appears**.
2. Click the Filter Events button, as shown in Figure 2.5.3. The Set Event Filters dialog appears box appears (refer Figure 2.5.4).

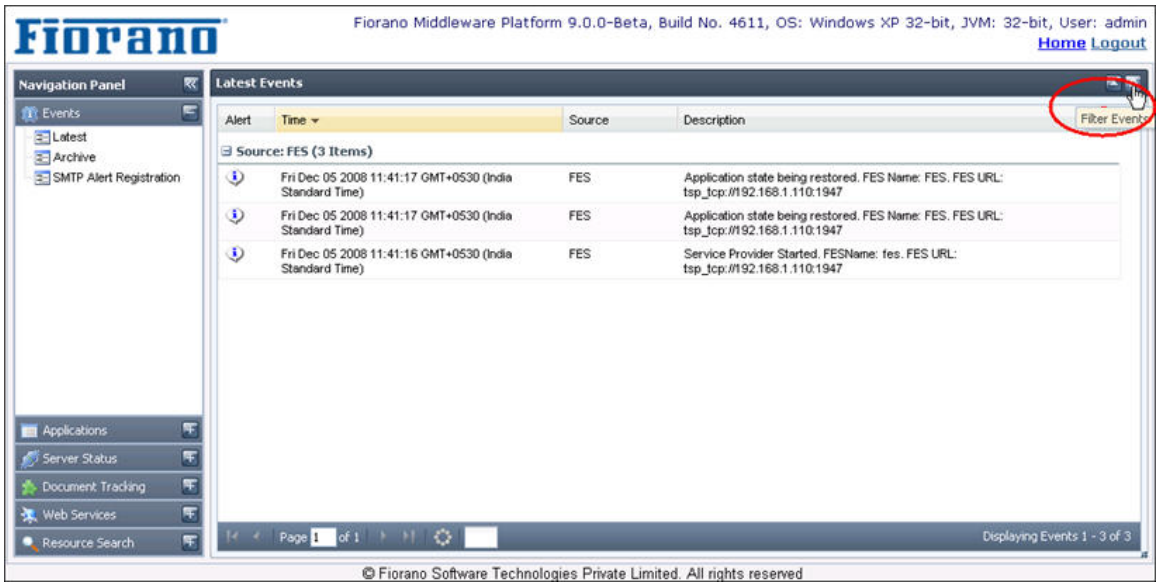


Figure 2.5.3: Filter Events button

3. Choose the Event Type and Event Category from the drop-down list and click the Done button. The result appears on the screen.

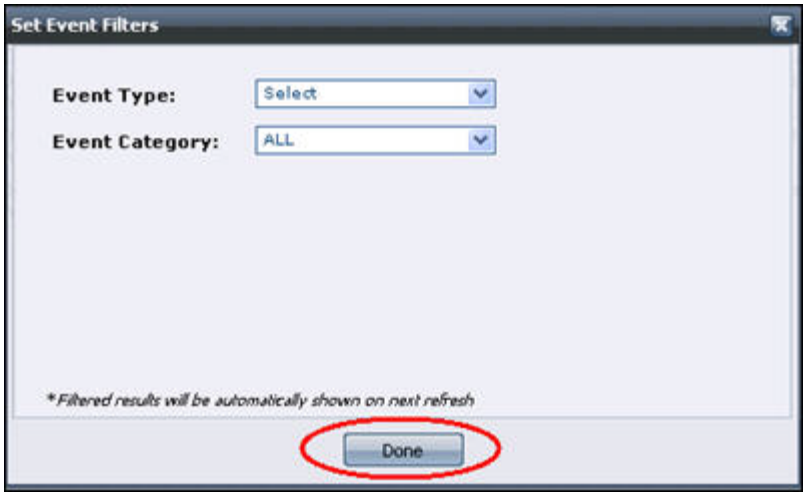


Figure 2.5.4: Set Events Filters dialog box

Archives: This section lets the User view Archived Events present in the database. The user has the option to choose Event Type, Event category, date and time range to view the corresponding Events.

To filter the events, perform the following:

1. Click the Events tab form the **Navigation Panel** and choose Archives option. The list of Archives Events **appears**.
2. Click the Filter Events button, as shown in Figure 2.5.5. The Set Event Filters dialog box appears (refer Figure 2.5.6).

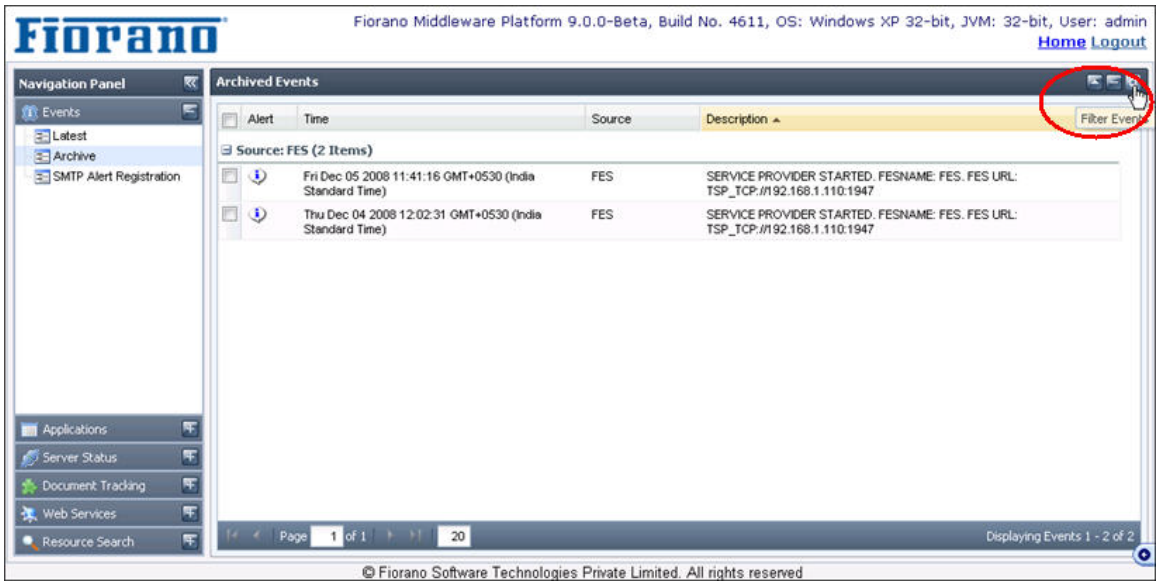


Figure 2.5.5: Filter Events button

3. Choose Event Type **and** Event Category from the drop-down list and click the Search button, as shown in Figure 2.5.6. The result appears on the screen.

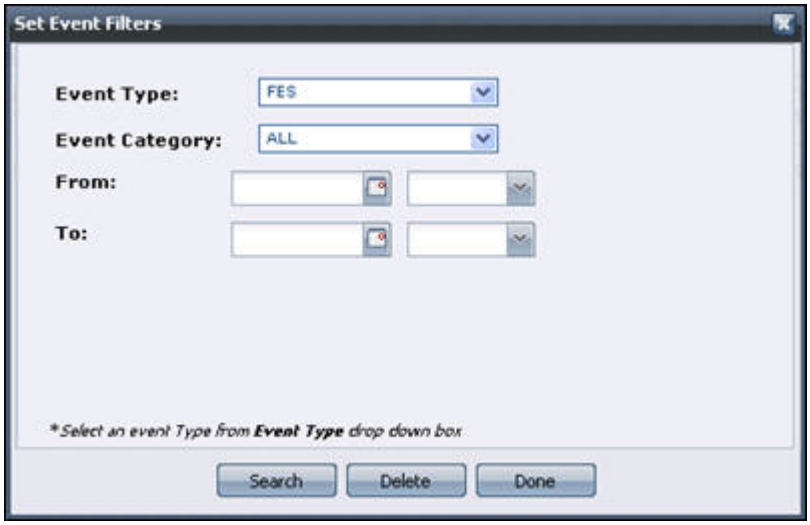


Figure 2.5.6: Set Event Filters dialog box

To Delete the Events, perform the following steps:

1. Click the Events tab form the **Navigation Panel** and choose the Archives option. The list of archives events **appears**.
2. Select the relevant Event and click the Delete selected records button.

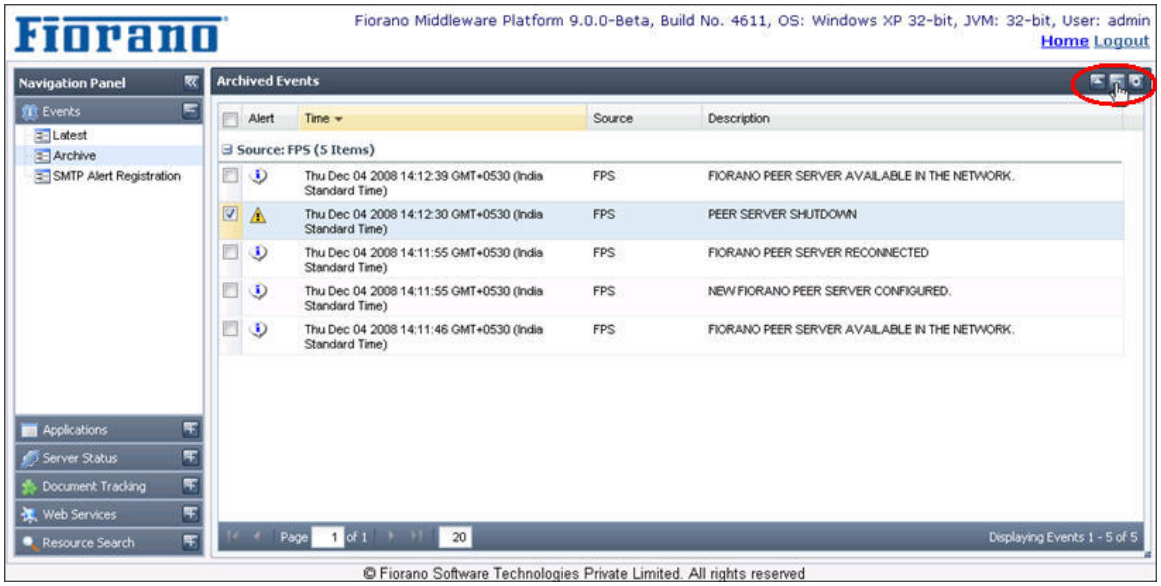


Figure 2.5.7: Delete selected records button

3. The Event Table Updated dialog box appears confirming the deletion, click the OK button. Refer to Figure 2.5.8.

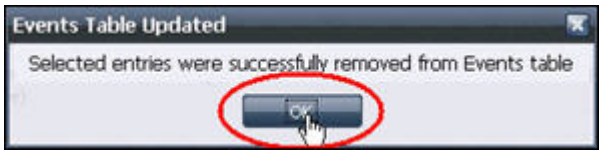


Figure 2.5.8: Event Table Updated dialog box

SMTP Alert Registration: No mail server is provided by default. A User can configure the mail server settings by going to the **Server Settings** button. This page also provides an option to specify usernames and passwords in case the mail server requires authentication of these in order to send e-mails.

To add an Alert Configuration, perform the following:

1. Click the Events tab form the **Navigation Panel** and choose SMTP Alert Registration option.
2. Click the Add an Alert Configuration button. The Configure SMTP Alert dialog box appears, as shown in Figure 2.5.8.

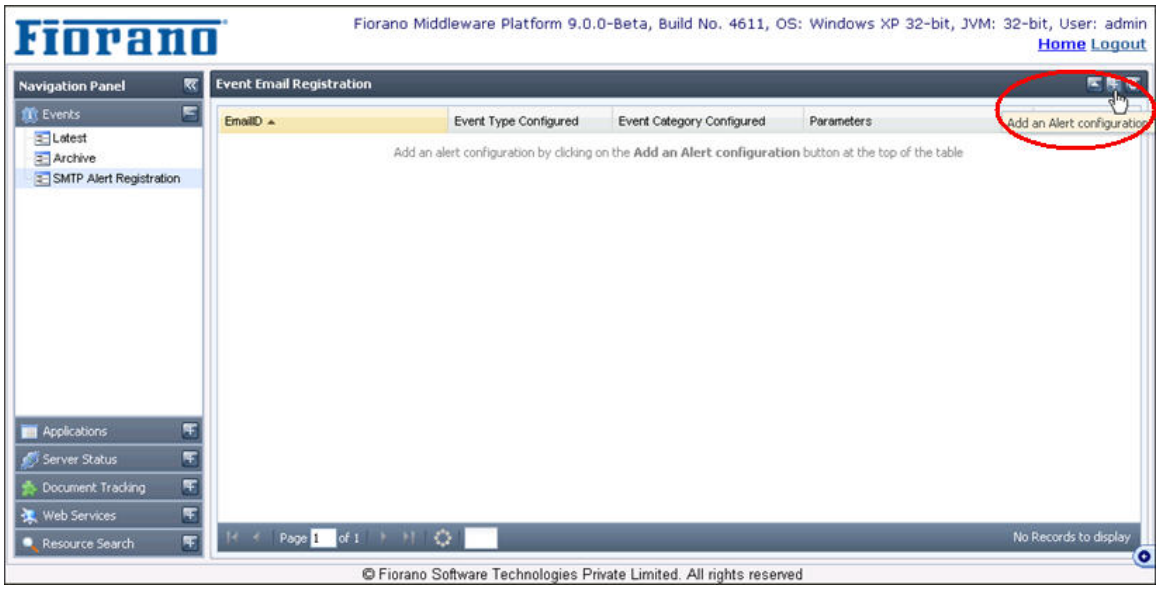


Figure 2.5.9: Add an Alert Configuration button

3. Choose the Event Type and Event Category from the drop-down list and enter the email ID in Enter email address field and click the Add button, as shown in Figure 2.5.10.

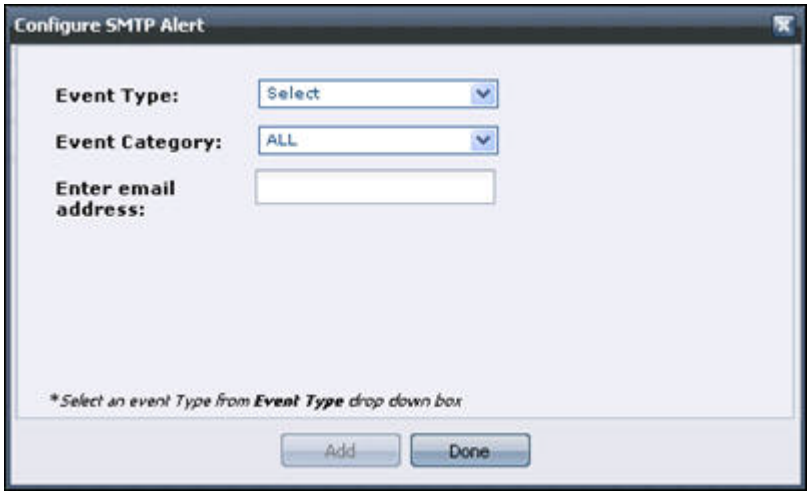


Figure 2.5.10: Configure SMTP Alert dialog box

4. Finally, click the **Done** button to apply the changes. The list of currently configured email alerts appears, as shown in Figure 2.5.11.

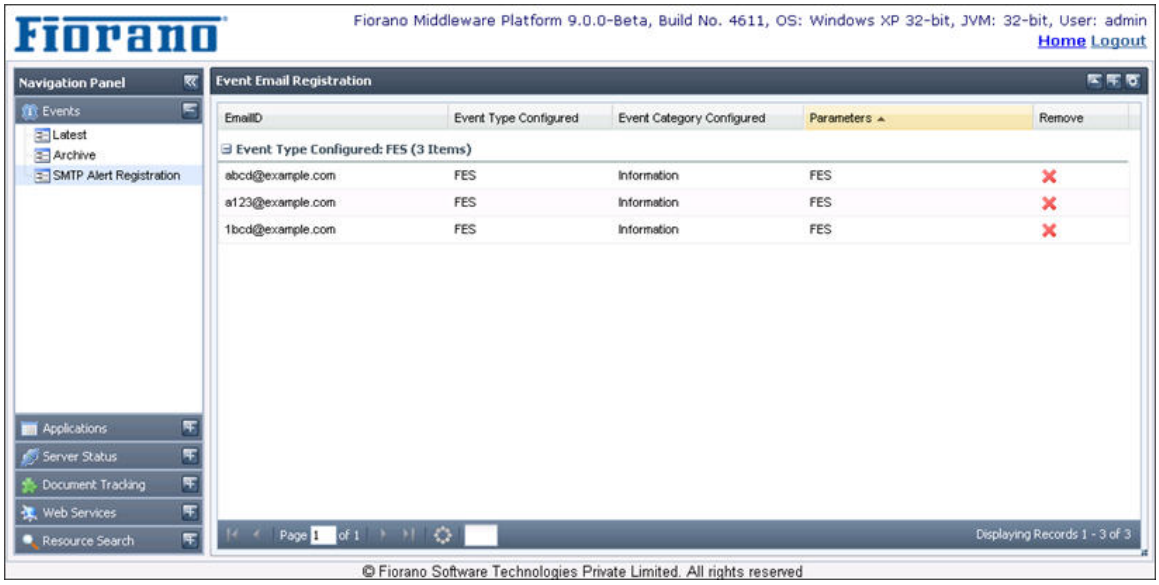


Figure 2.5.11: Event e-mail registration option

To configure SMTP server settings, perform the following:

1. Click the Events tab form the Navigation Panel and choose the SMTP Alert Registration option.
2. Click the Configure SMTP Server Settings button. The Configure SMTP Server Settings dialog box appears, as shown in Figure 2.5.12.
3. Enter the details and click the Save Configurations button to save the configurations and click the Done button after saving the configurations, as shown in Figure 2.5.12.

Figure 2.5.12: Configure SMTP Server Settings dialog box

2.5.1.2 Security

Security section gives the security actions that can be performed in the Fiorano Network. These include adding/ editing users, groups, application level or group level permissions and validation or strength rules for passwords. These also include the principal store synchronization.

The Security tab has six sections:

- Users
- Groups
- Global Permissions
- Application Permissions
- Principal Store Sync
- Security Datastore Reset
- Password Rules

Users

This tab shows the users that are currently registered in the Fiorano network. The actions that can be undertaken in this tab include creating new users, deleting an existing user or changing password for an existing user provided the user has proper permissions to do so.

Figure: Users

Groups

This tab shows the groups registered in the Fiorano network. Groups allow grouping users together and applying actions on them together. The actions that can be undertaken in this tab include creating new groups, deleting/editing an existing group provided the user has proper permissions to do so.

Figure: Groups

Global Permissions

This tab shows the global permissions that are currently available in the Fiorano network. This tab allows the user to edit or specify the users for whom these permissions will be applicable provided the user has proper permissions to do so.

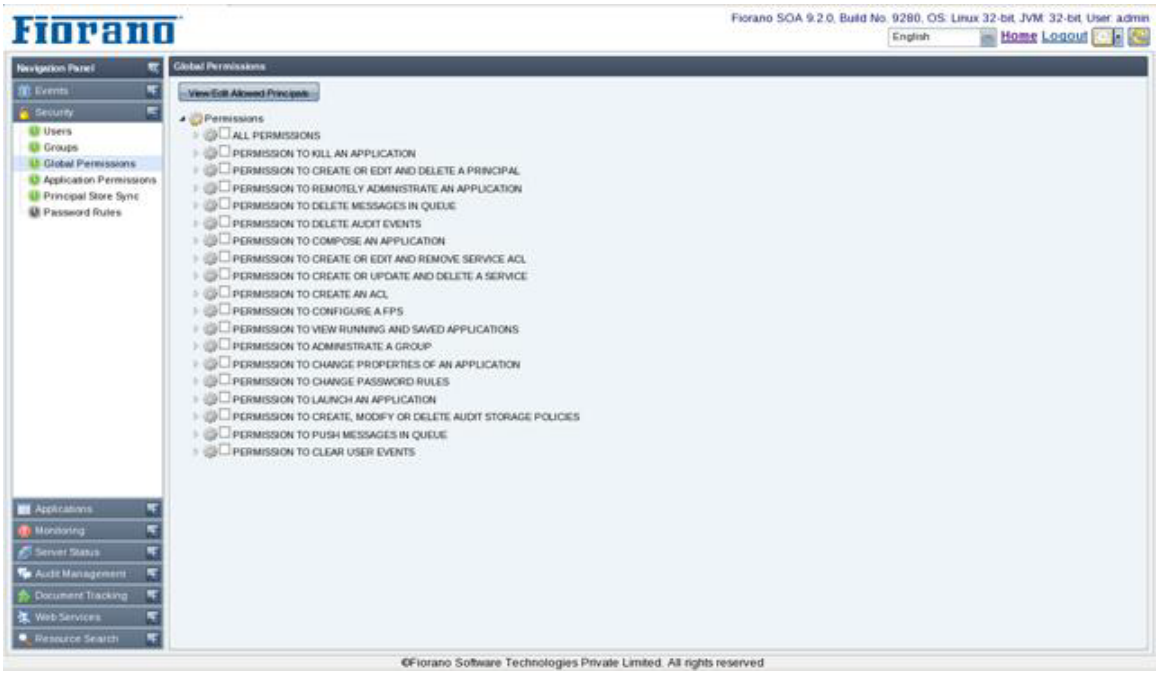


Figure: Global Permissions

Application Permissions

This tab shows the application permissions that are available in the Fiorano network. This tab allows the user to edit the application permissions for a user provided the user has proper permissions to do so.

Please note that application level permissions override global level permissions. For example, if a user is given launch permission for applications in global permissions, but for a particular application he/she is not granted the permission to launch that application, then the user will not be allowed to launch that particular application.

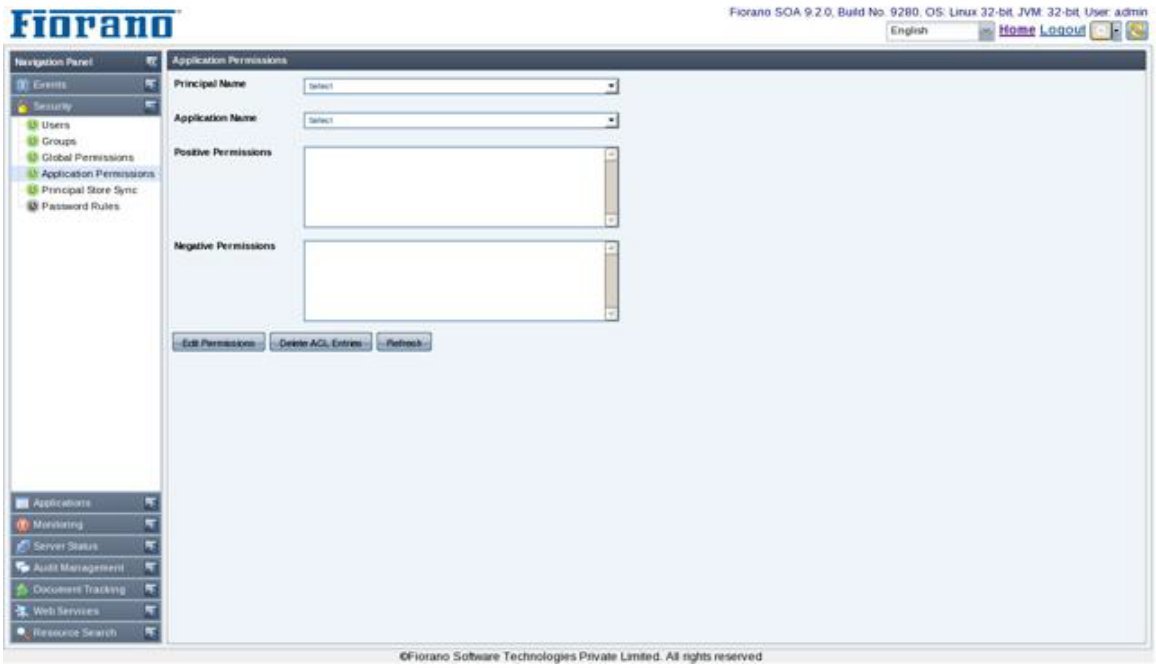


Figure: Application Permissions

Principal Store Sync

This tab shows the principal store synchronization status for the connected peers. This also allows the user to synchronize the store for a peer.

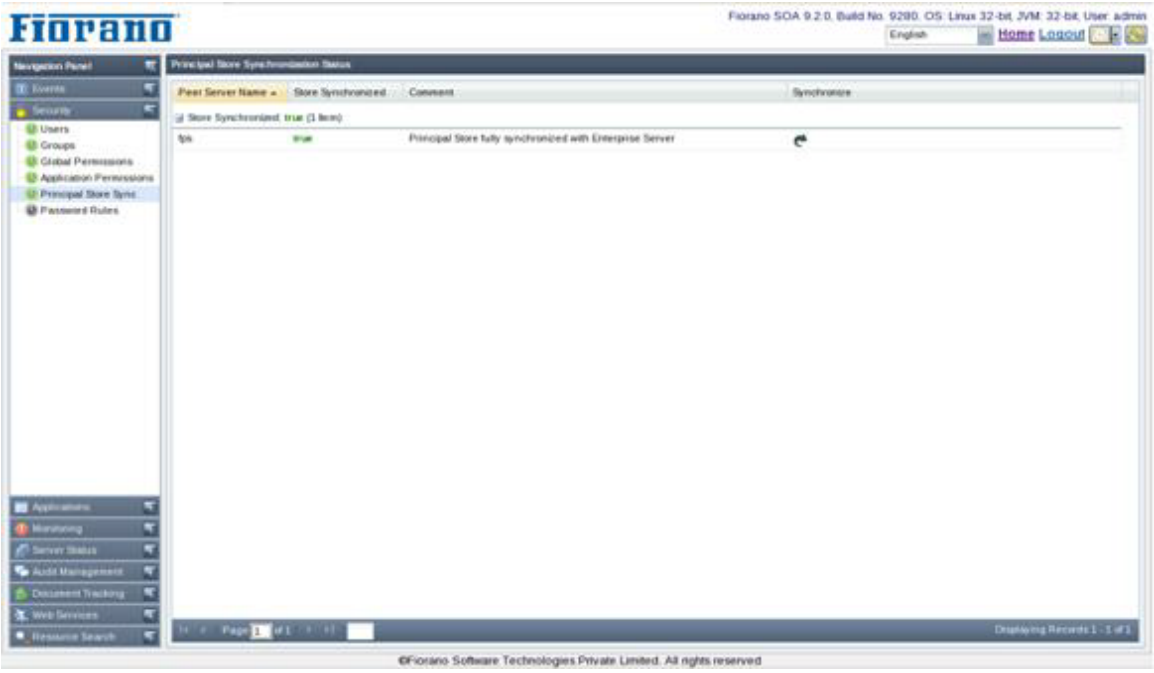
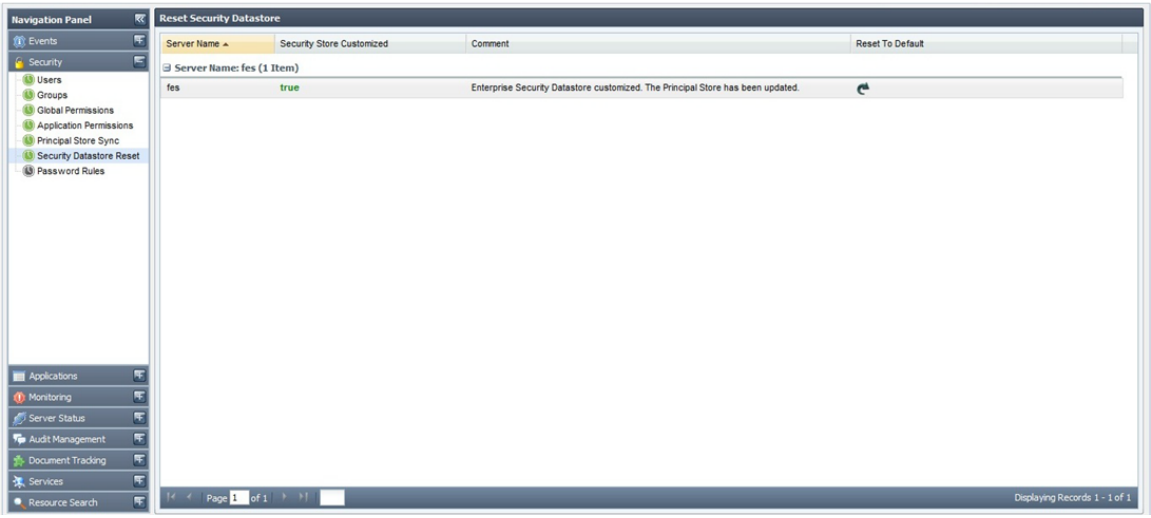


Figure: Principal Store Sync

Security Datastore Reset

This tab shows the Security Datastore of the Enterprise Network. It also allows the authorized user to reset the entire Security Datastore viz. the Principals and the ACLs to system default.



Password Rules

This tab shows the validation and strength rules that are used for matching the passwords or determining their strengths. This tab allows the user to add new rules or remove an existing one. By default, no validation rules are provided.

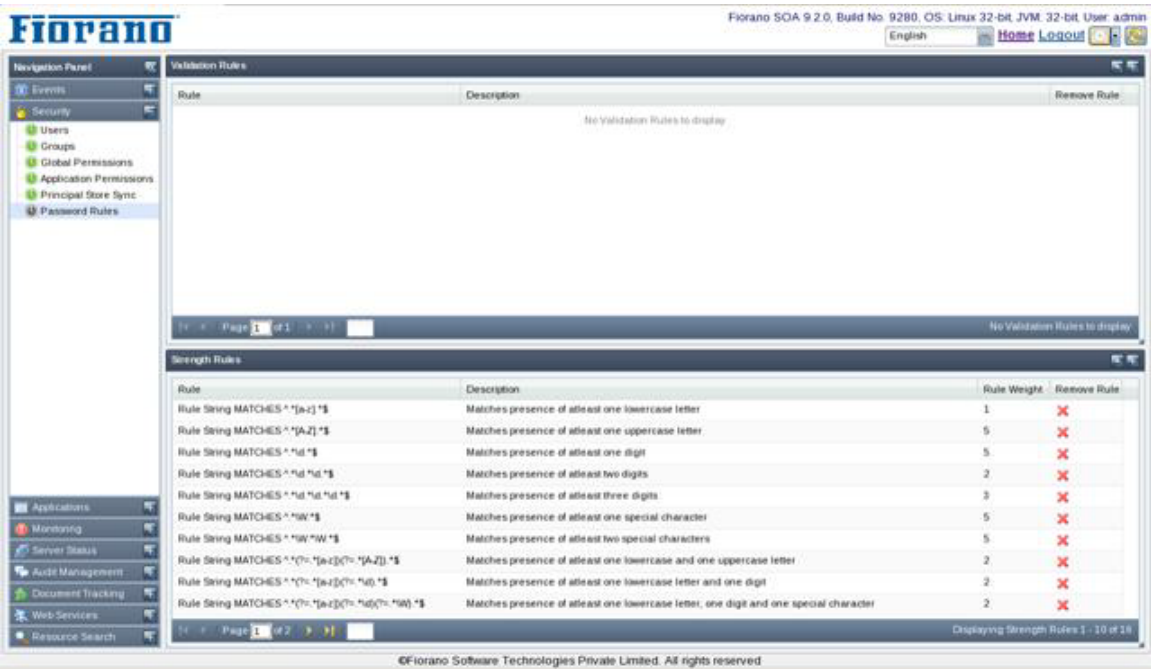


Figure: Password Rules

2.5.1.3 Applications

This section shows the details of the event processes running on the Fiorano Peer Server. The top view shows the list of event processes saved in the Fiorano Server. It also shows the details like running status, category and the Peer servers used. By clicking on the link for each event process, you get the details of the Service Instances running as part of it. This page also provides capabilities to launch, stop, restart an event process or its components, View Output and Error logs, and Export logs (These features is disabled if event process is in debug mode). The details for the services are displayed in the bottom view. This includes

- Service Instance Name – Service components in the event process
- Service GUID – Service GUID of the component
- Version – Version of the component
- Status – It tells whether the component is running or not
- Running on Peer – Name of the peer server on which the component is launched
- Launch Type – It tells whether the component is launched as a separate process, in-memory or manual.

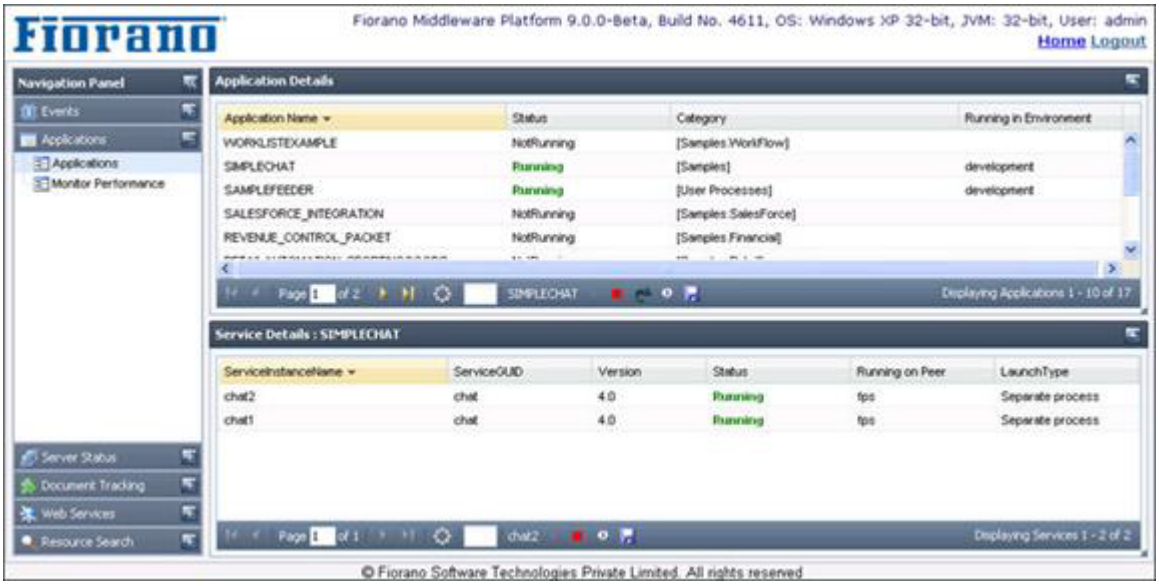


Figure 2.5.13: Applications Tab showing the details of the applications

Monitor Performance

The user can monitor the performance of services by enabling the monitor performance in component CPS. Performance statistics are shown in two views:

Data View

This view shows performance messages sent by the components in data form. User has the choice to select the components for which monitoring data should be displayed. Also, the time-interval for which monitoring data is displayed can be configured.

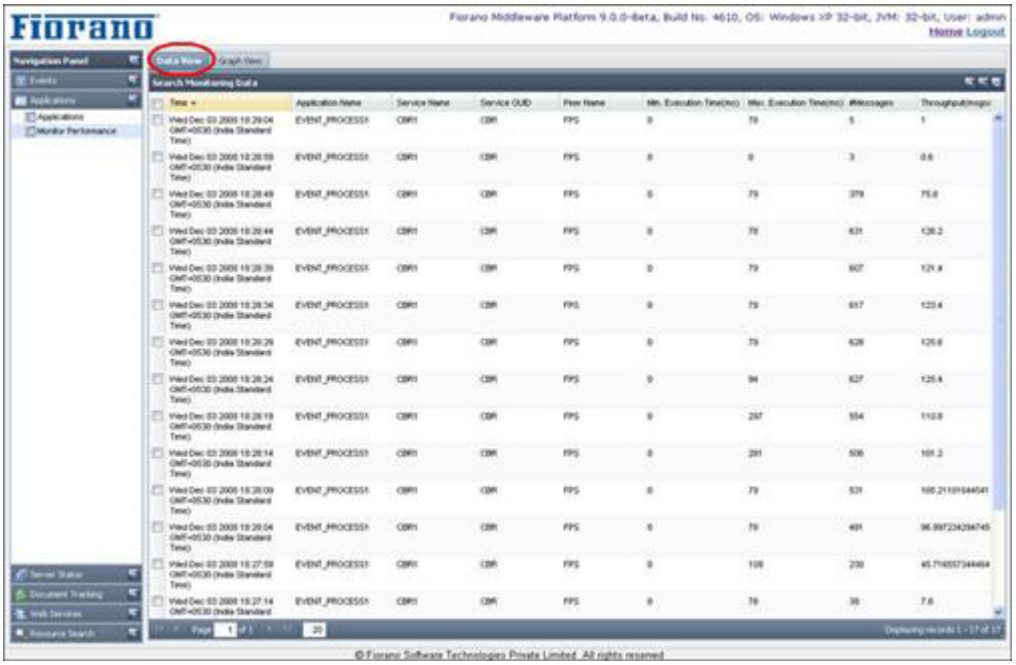


Figure 2.5.14: Data View tab

Graph View

Here the performance of the component is shown in the form of a graph. Note that only one component can be monitored at a single time in Graph View. Graphs can be monitored for archived performance data or for the latest data. To show latest performance data, select live graph option. Maximum number of points plotted on the graph can be configured by specifying desired value for the Max Points field. It is recommended using Firefox 2.0 or Internet Explorer to see Graph View.

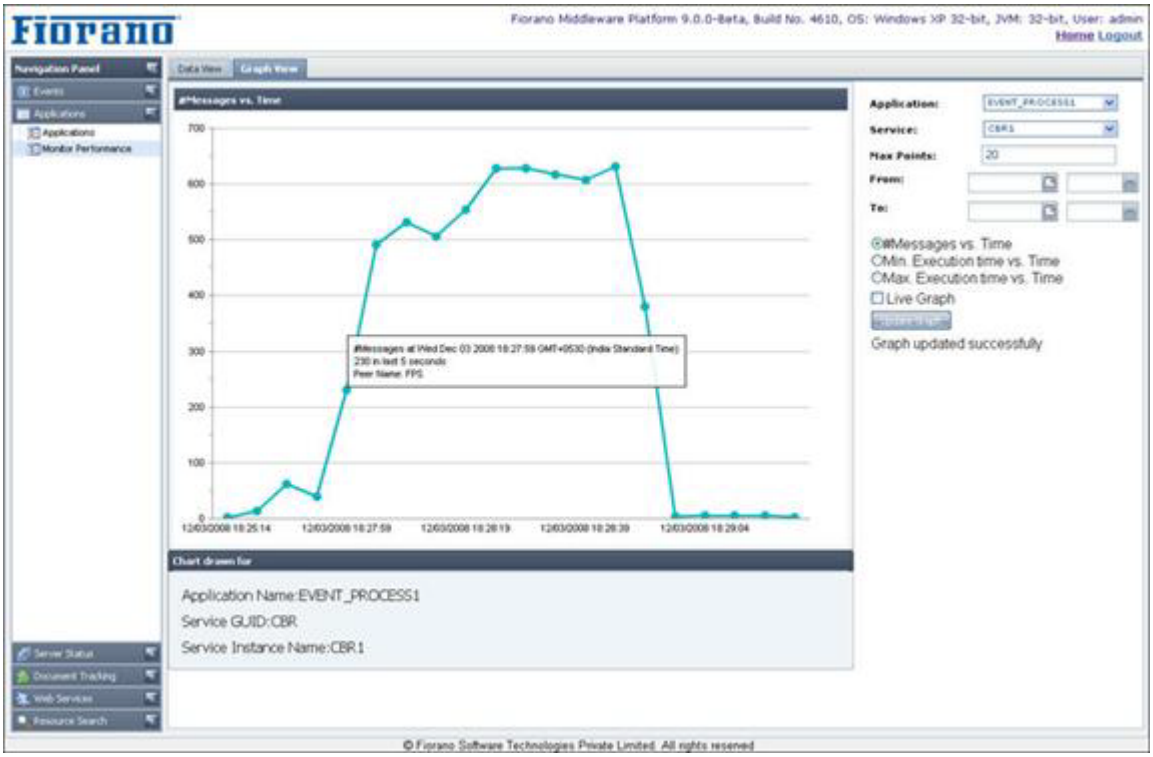


Figure 2.5.15: Graph View

2.5.1.4 Monitoring

The monitoring section helps the user to add policies or alerts to monitor the Fiorano Network. Refer to Chapter 11 for detailed description on Backlog Monitoring feature.

The monitoring section helps the user to add policies or alerts to monitor the Fiorano Network. The Monitoring tab has two sections:

- Policy Manager
- Alert Manager

Policy Manager

This tab shows all the policies that are present in the Fiorano Network. This tab allows the user to add new policies, edit or delete an existing policy, apply or suspend existing policies. A user can add two types of policies

- Backlog Policy
- LowMemory Policy

Figure: Policy Manager

Alert Manager

This tab shows all the alerts that are present in the Fiorano Network. This tab allows the user to add new alerts, edit or delete an existing alert. A user can add two types of alerts:

- SMTP Alert
- JMS Alert

Figure: Alert Manager

This tab has two sub-tabs. These allow to specify the transport settings that will be used for sending the alerts. The settings can be specified for SMTP and JMS transports.

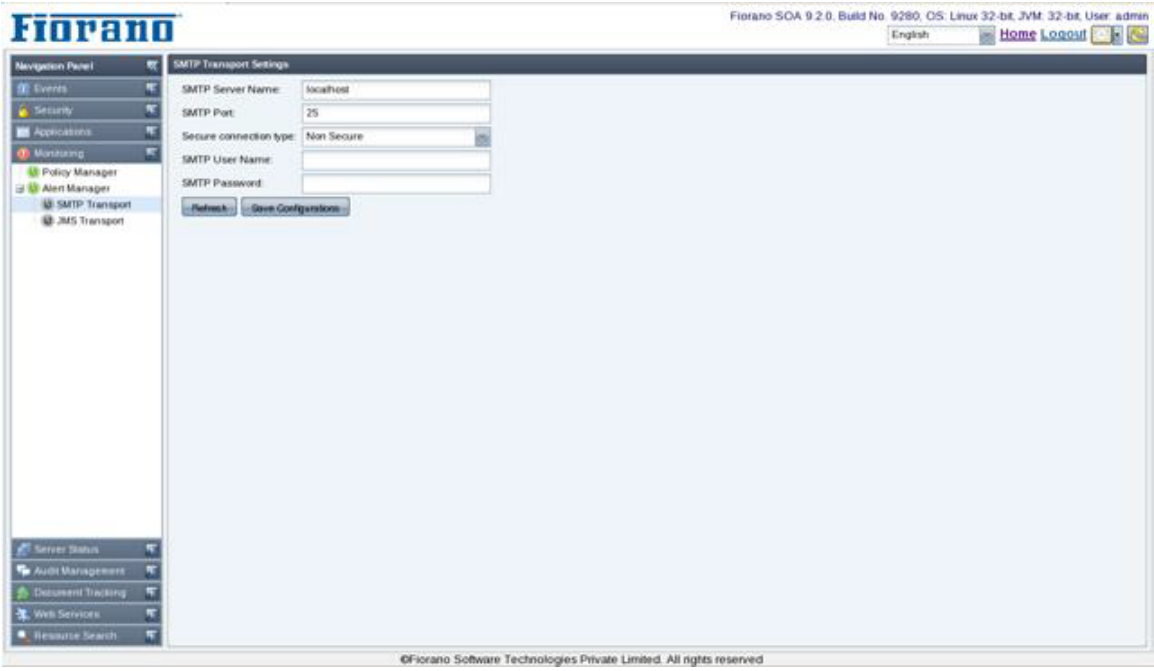


Figure: SMTP Transport

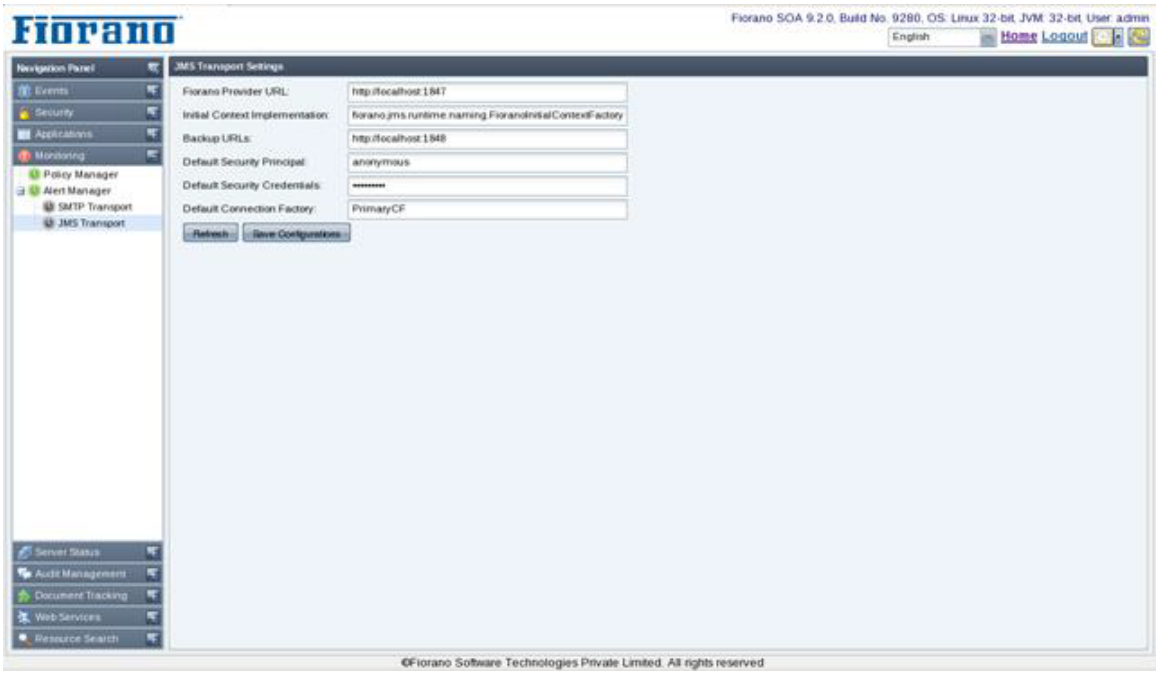


Figure: JMS Transport

2.5.1.5 Server Status

Server Status tab shows the details of the available Fiorano Servers. The top view shows the running status as well as the memory usage. Further details are available on clicking the server links, which loads the bottom view with the following details:

- **System Details** – O/S and JVM statistics of the server
- **Topics, Queues and Connections** – List of JMS topics and queues present in the server and the connections created by the server
- **Out and Error logs** – Displays the server logs

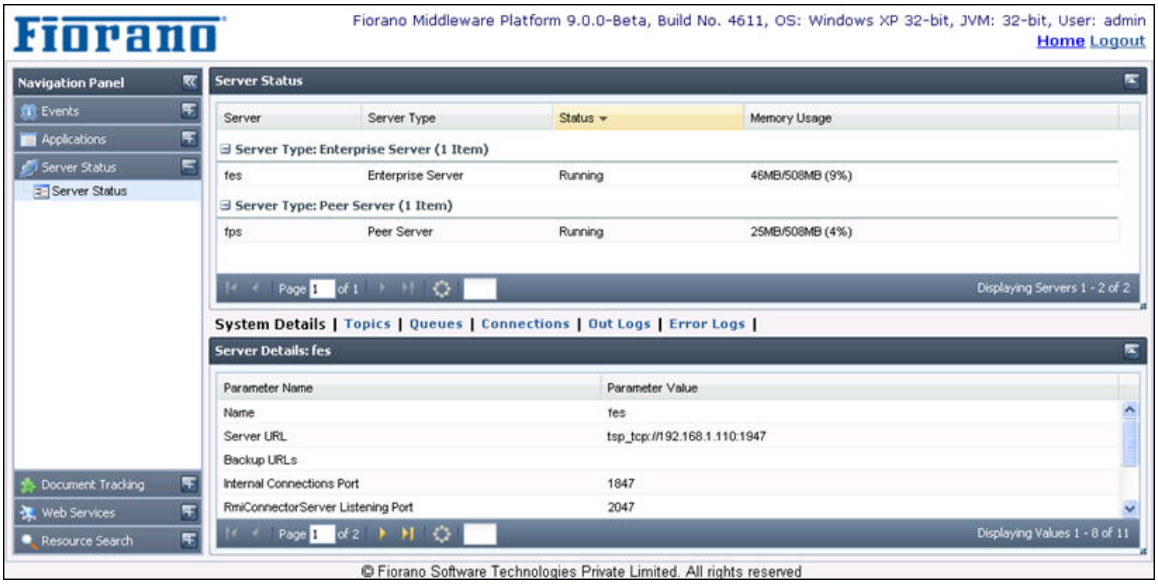


Figure 2.5.16: Server Status tab showing System details

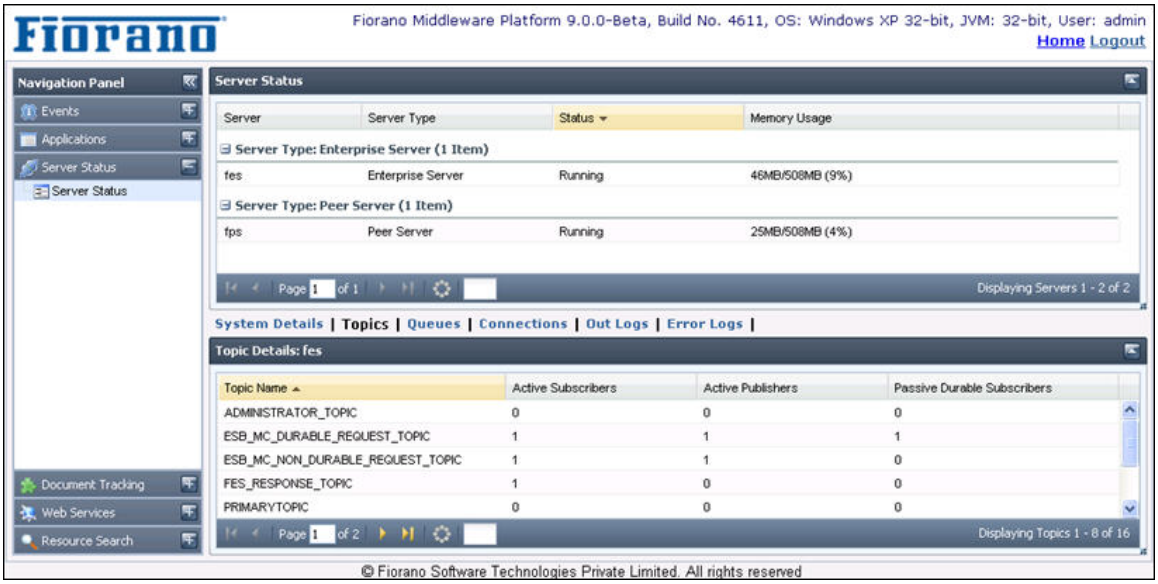


Figure 2.5.17: Server Status tab showing topics created by FES

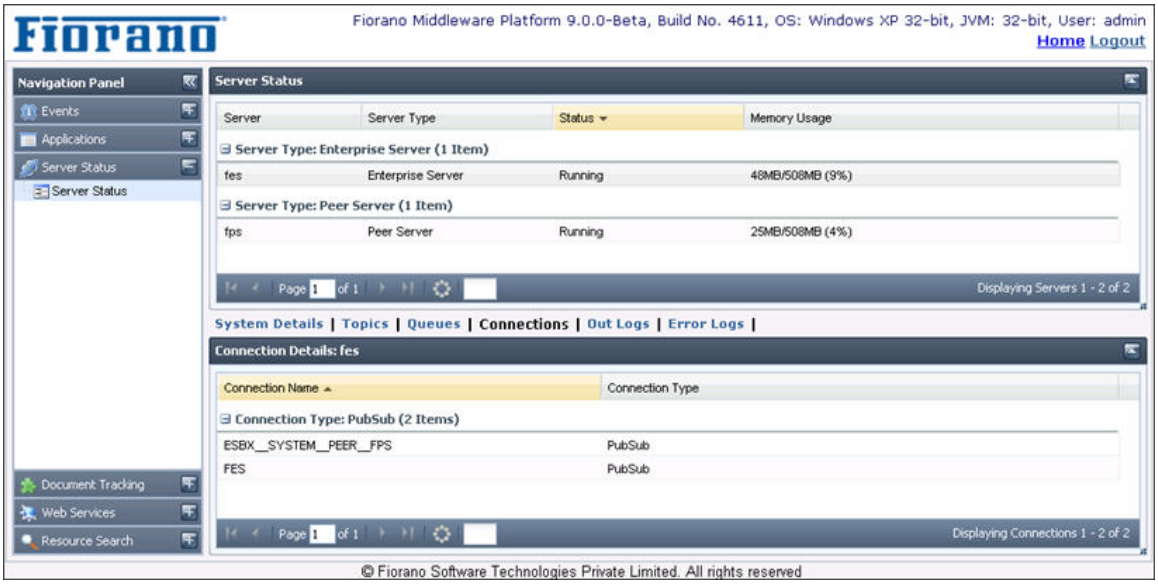


Figure 2.5.18: Server Status tab showing connections created on FES

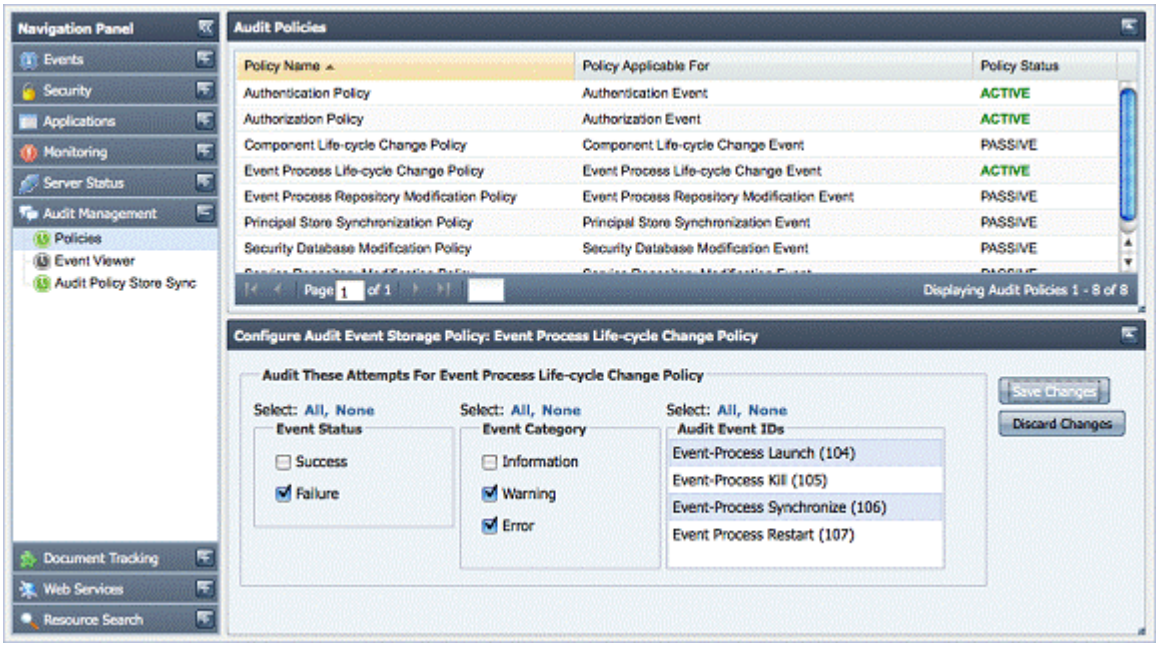
2.5.1.6 Audit Management

Audit Management section in web console allows the user to define audit policies and search for audit events that user is interested in. Refer to Chapter 12 for detailed information about Fiorano Audit Management feature. This section has been divided into 3 sub-sections.

- Policies
- Event Viewer
- Audit Policy Store Synchronization

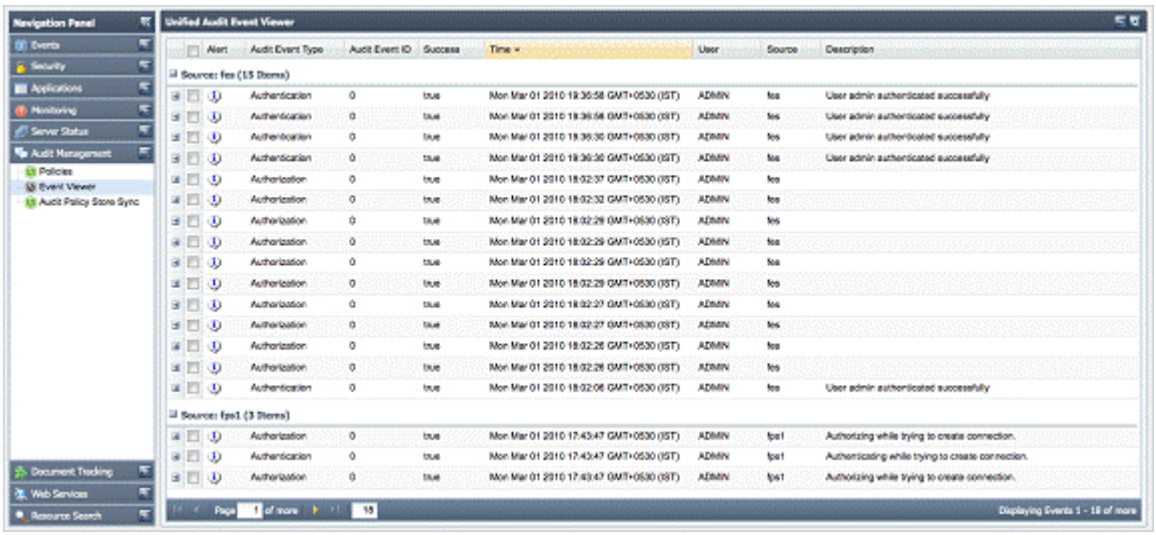
Policies

In this view, users can view/edit available audit policies. These policies define the actions that will be audited by the system. In a fresh installation, all policies are in passive state, meaning none of the action is audited. Users can selectively enable the actions that they want to audit.



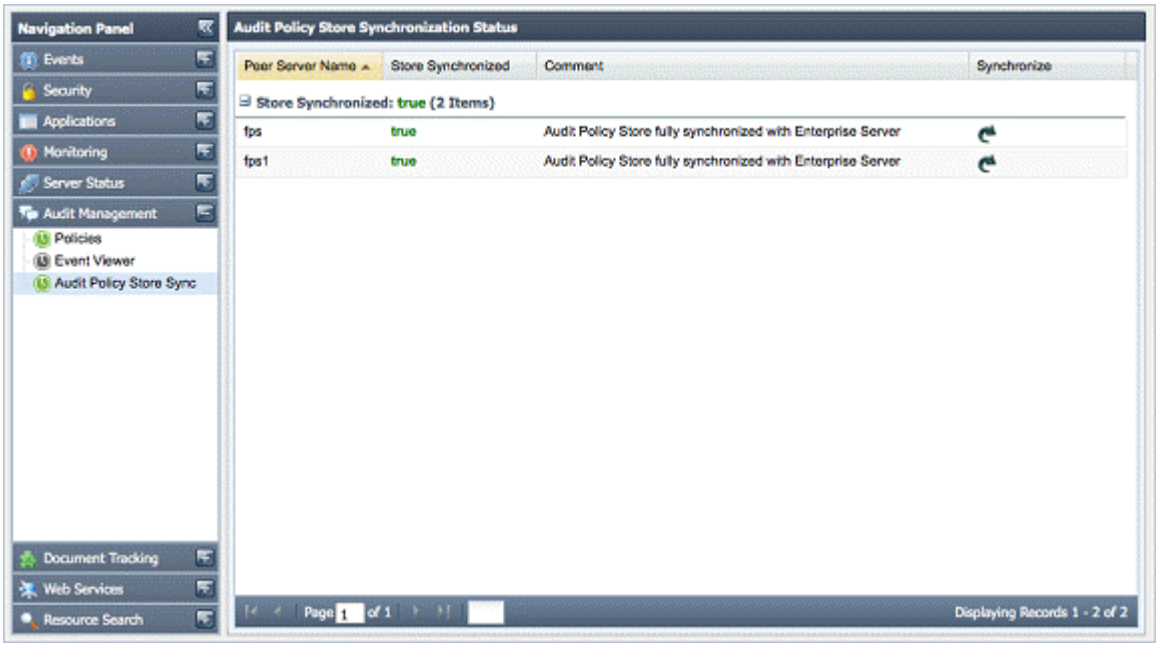
Event Viewer

This view shows audit events based on the search criteria's specified by the user. A comprehensive list of audit event filters is available to refine the audit events that user is interested in. This view also allows saving search preferences for later use, thus avoiding the painful task of creating audit filters each time user wants to search (See Chapter 12.5 for more information). Below is a snapshot of Event Viewer page rendered after user searched for audit events.



Audit Policy Store Synchronization

This view shows the result of synchronization of audit policy store operation (See image below). A synchronized audit policy store means that the same audit policies (as in Enterprise Server) are active in peer servers as well. If stores are not synchronized, the status field will display 'false' and the reason for which the synchronization could not be achieved will be displayed under Comment column, on which user will have to take corrective action. User can force re-synchronization of audit policies with a particular peer server by clicking on the image shown under 'Synchronize' column.



2.5.1.7 Document Tracking

This section shows all the tracked documents in Fiorano Event Processes along with all details of the tracked document.

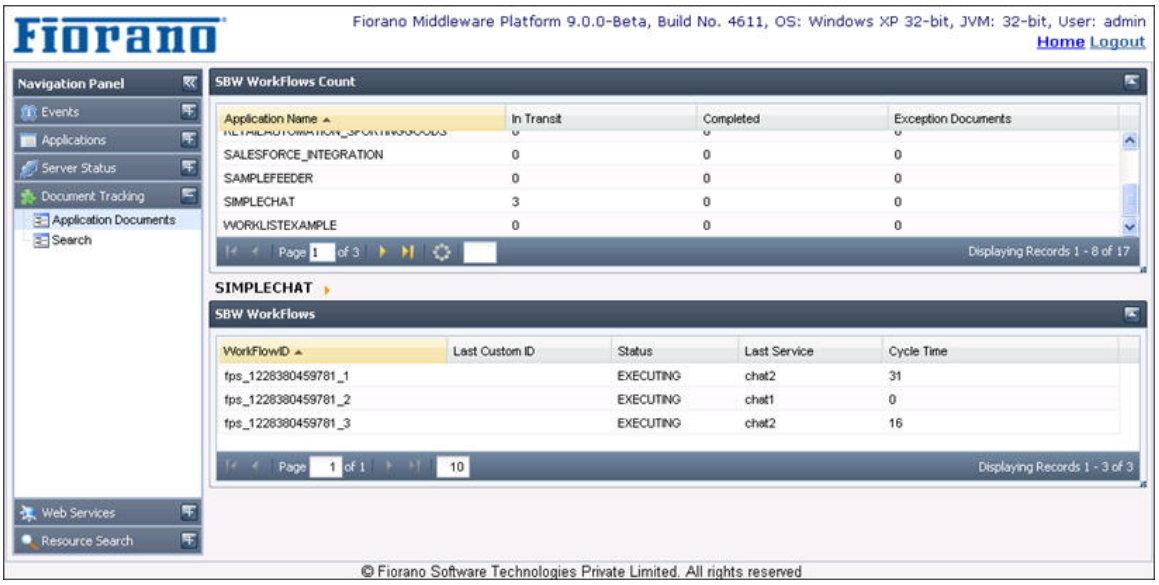


Figure 2.5.19: Document tracking tab showing tracked documents

The details of each tracked documents can be seen by clicking the particular document. This shows the document details such as the component processing it, time stamps, document IDs and the originating port of the tracked document.

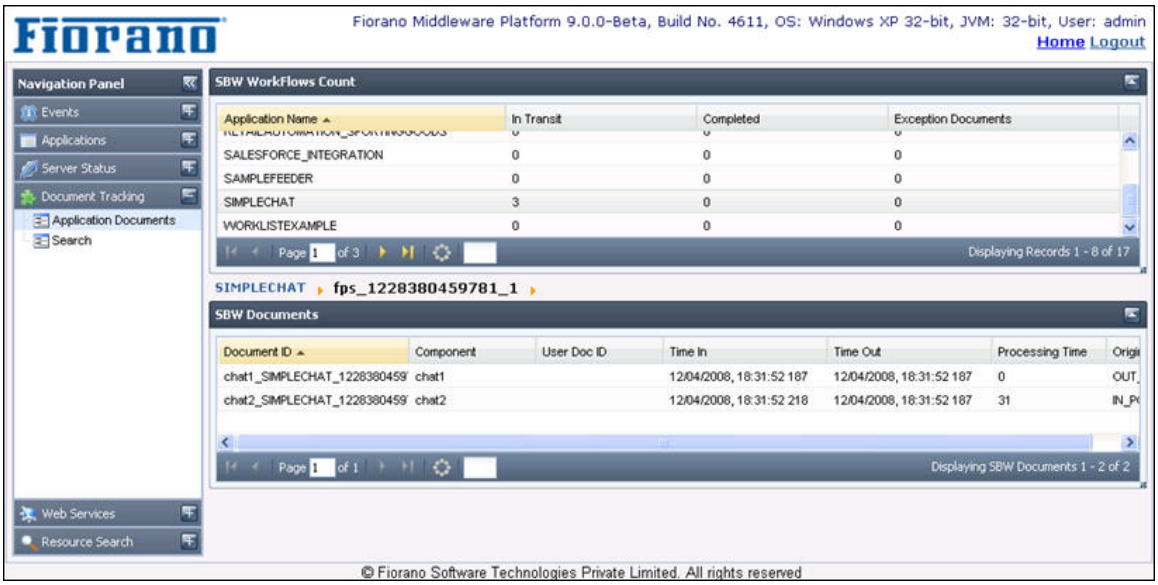


Figure 2.5.20: Details of the tracked document

The tracked document's properties can be seen by clicking on a particular Document ID. This shows the tracked document message properties, details of attachments, application context, message body and other general properties.

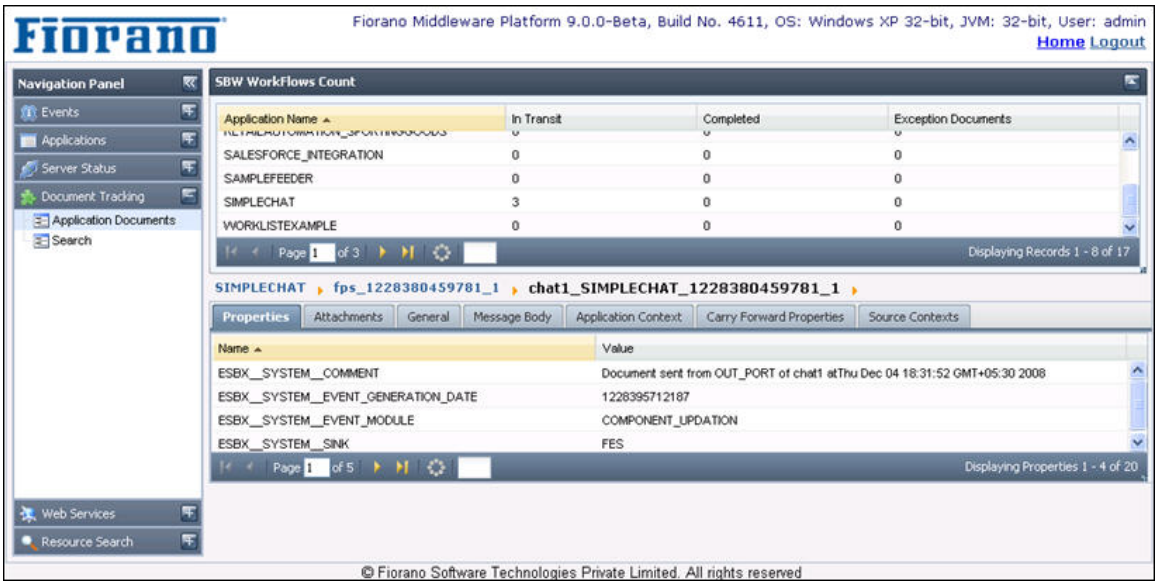
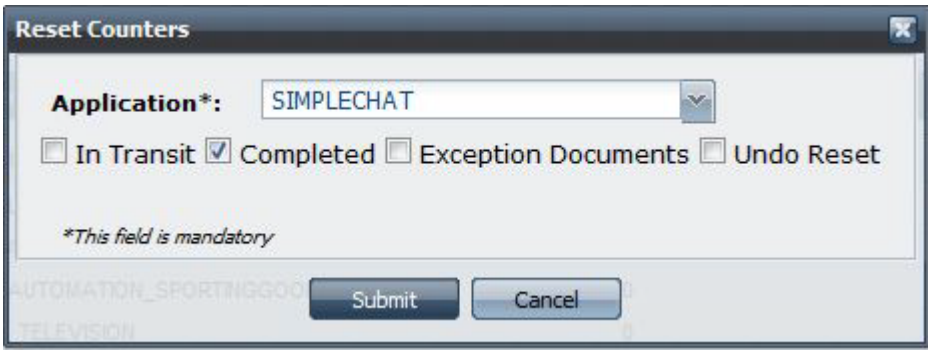


Figure 2.5.21: Tracked document

The User can also reset the counters of Workflows and Exception Documents as visible in the Application Documents tab. The counters to be reset can be selected by ticking the appropriate checkbox. The reset action can also be undone later.



The Dashboard supports searching for tracked documents based on many criteria such as, Application Name, Peer Server Name, Document Status that is, EXECUTED or EXECUTING, Service Instance Name, and Port Name. In addition, the documents can be searched based on their date they were generated.

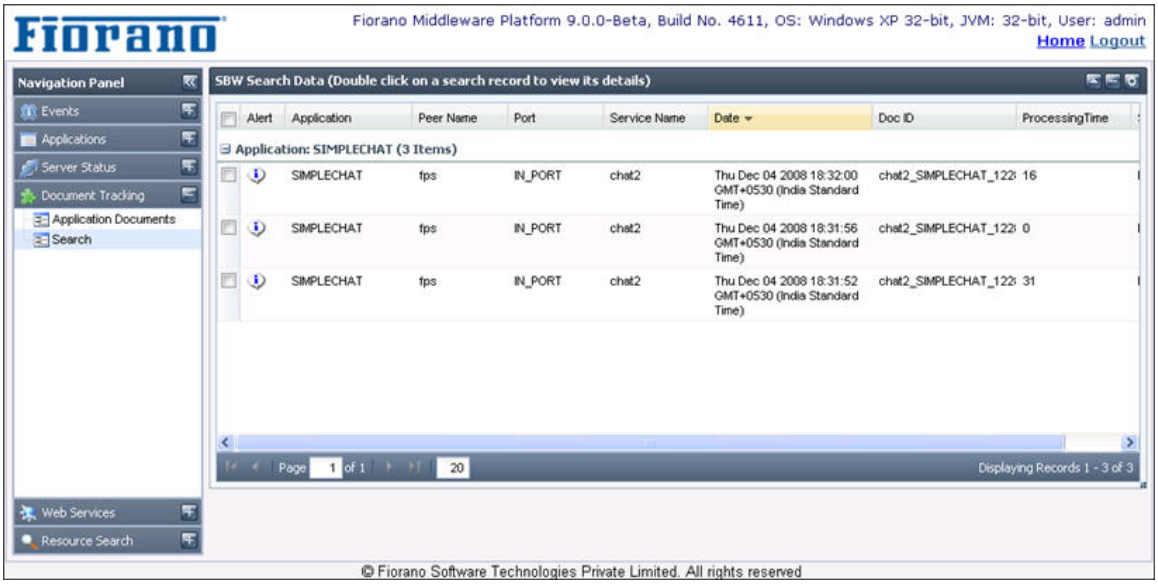


Figure 2.5.22: Searching tracked documents

2.5.1.8 Services

The Services tab shows the details of the event processes deployed as Web Services. The User can view the status of web service either online or offline and has the option of enabling or disabling this option. The User can also test RESTful deployed from the dashboard.

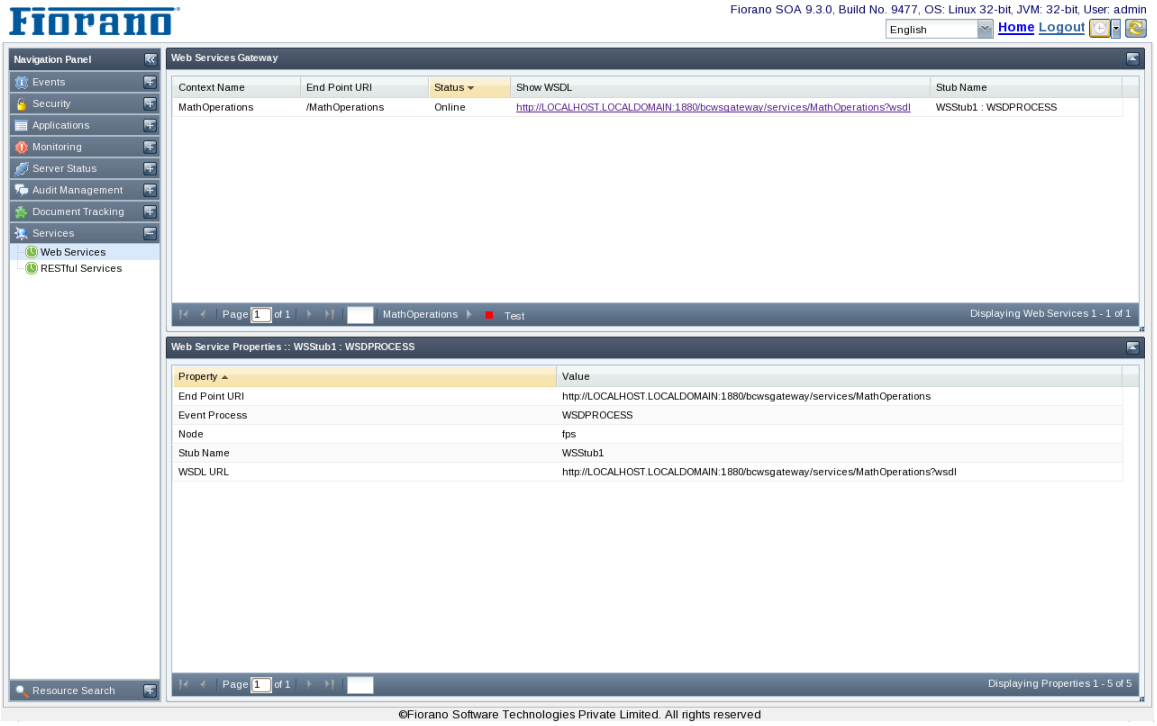


Figure 2.5.23: Web Services – Event process

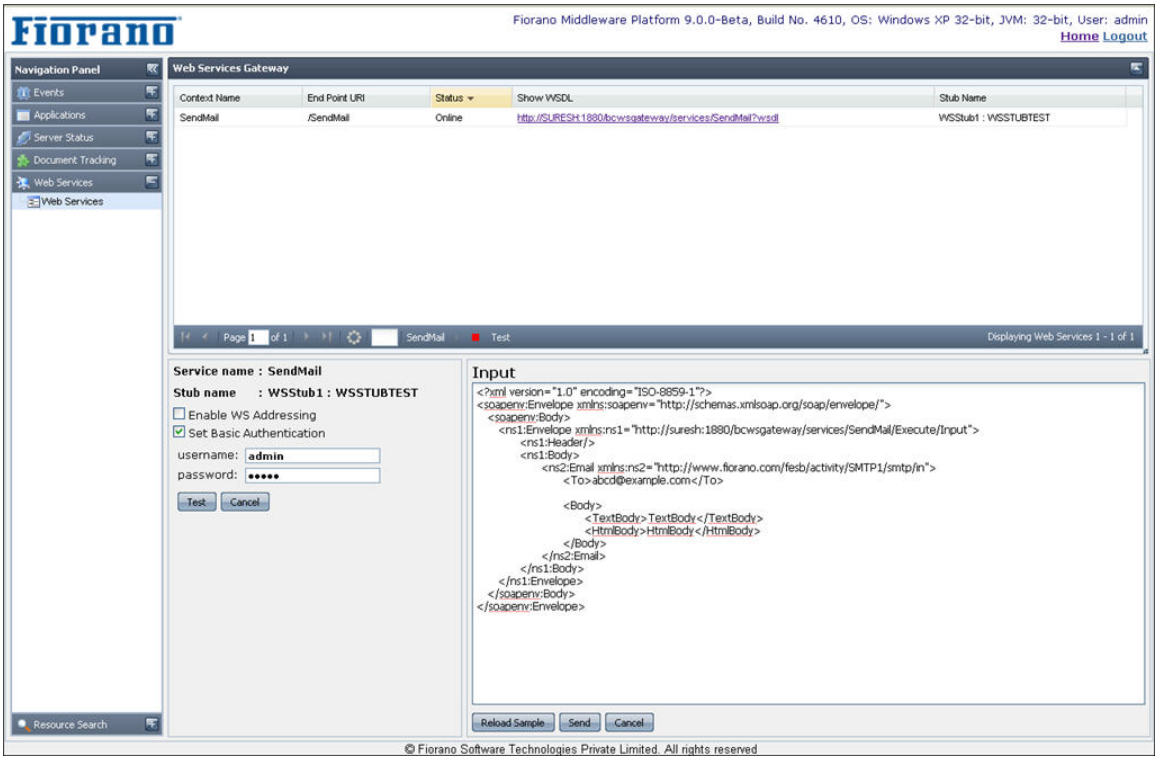


Figure 2.5.24: Web Services tab

The details shown for the Event Process deployed as web services are:

- **Context Name** - Name of the context for the web service deployed
- **End Point URL** - Effective End Point URL is http://<peerserverip>:<httpport>/<rootContext>/ContextName
- **Status** - Shows whether the web service is online or offline
- **Show WSDL -** Gives the link to show WSDL
- **Stub Name** - Name of Stub for the deployed Event Process as web service

2.5.1.9 Resource Search

This section allows the User to search for different resources that have been configured to be used by Fiorano Event Processes. The search for the resources can be performed based on three types:

1. Application View
2. Component View
3. Resource View

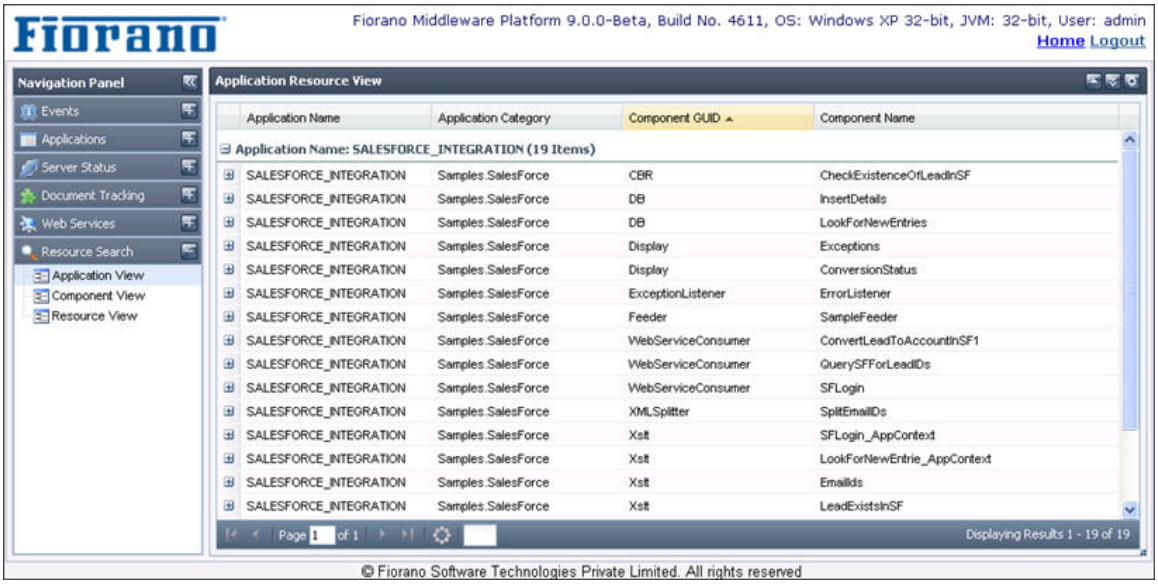


Figure 2.5.25: Searching configured resources based on application view

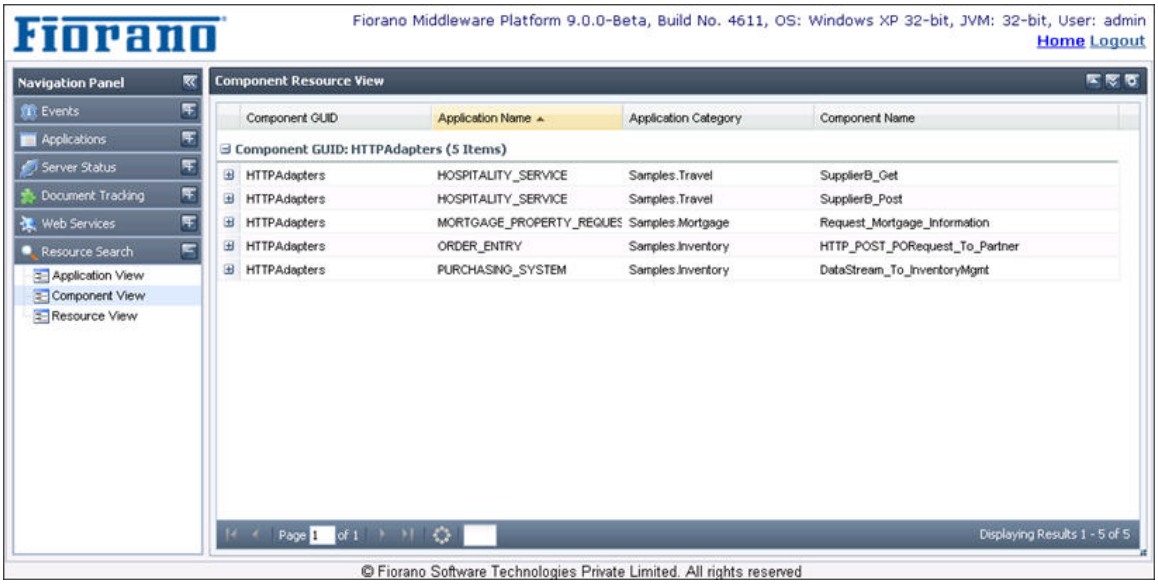


Figure 2.5.26: Searching configured resources based on component view

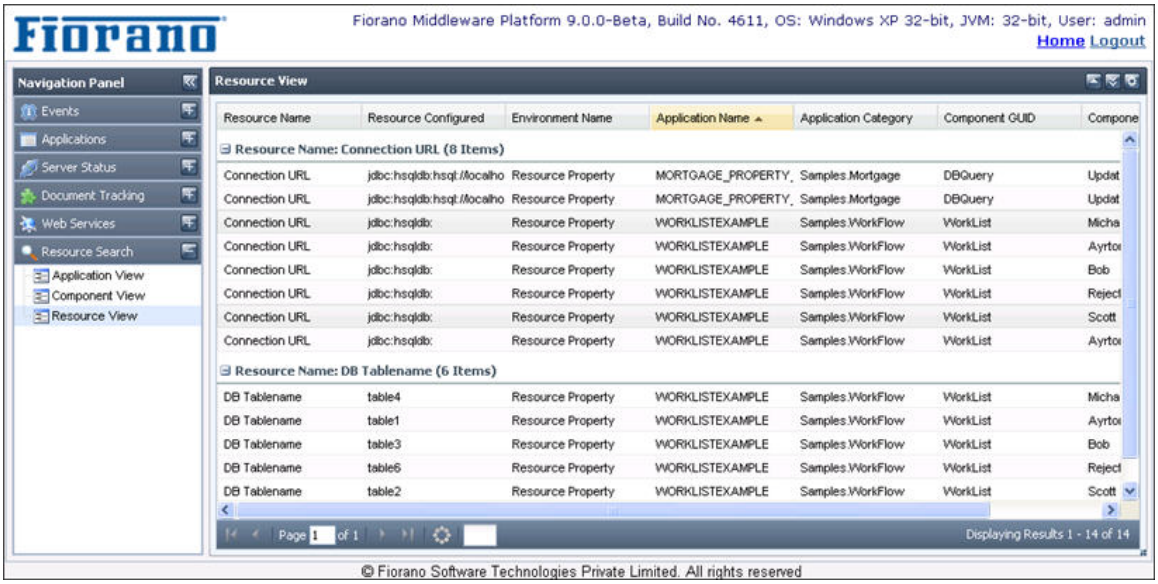


Figure 2.5.27: Searching configured resources based on resource view

2.5.2 Enabling Fiorano Web Console

The Fiorano Web Console can be enabled in an Online and Offline mode.

To enable Fiorano Web Console in online mode (server running):

1. Login to Peer server via JMX(FPS-JMX)
2. Navigate to Fiorano>etc>Jetty Server>JettyServer>config and set property EnableStart to yes.
3. Right-click on **FPS-JMX** and click **Save Configurations** option.
4. Shut down both Peer and Enterprise servers.

To enable Fiorano Web Console in offline mode (server not running):

1. Open profile in studio, browse to Fiorano>etc>Jetty Server>JettyServer>config and set property EnableStart to yes.
2. Save profile.

After editing profile (online or offline)

1. Clear peer repository, (using clearDBServer.sh -mode fes –profile <fes_profile_Name> and then choose option to clear "Peer Repository")
2. Restart the servers.

While starting peer server the port for WMT will be listed as the Dashboard Listening Port.

2.6 Configuring Servers and Tools

This section explains the new scripts introduced in SOA 2007 to manage and configure Fiorano Servers and Fiorano Tools.

2.6.1 Configuration File

Each script is associated with a specific configuration file (conf) ideally in the same location as the script file, with the name as that of the script file followed with .conf as the extension. This configuration file provides various configuration properties of server/tools as shown below:

Config Property/Block	Usage
<java.classpath>	Specify any additional jar files required to be in classpath in a separate line at the end of this block.
<java.endorsed.dirs>	Specify the jars to be considered other than the default jars in a separate line at the end of this block.
<java.ext.dirs>	Specify the external jar files to be loaded along with the default system jars, in a separate line as the end of the block.
<java.library.path>	Specify the folders containing dll/so files that are to be loaded, in separate line at the end of the block.
<java.system.props>	Specify any additional system properties, in separate a line at the end of the block.
<jvm.args>	Specify any arguments to JVM like memory settings; debug info, in a separate line at the end of the block.

Note following points about configuration file:

1. Comments can be written in conf file. A line starting with '#' is treated as comment. For example, the above sample conf file has some comments highlighted in green color
2. A .conf file can have empty lines. These empty lines are simply ignored by the launcher.
3. Environment variables can be used in conf files. (Using environment variables makes your conf file platform dependent)
4. Wildcards are not supported. That is, you should not write lib/*.jar

2.6.2 Reference Matrix

The following table summarize all the scripts that are changed in SOA 2007 for **Windows**:

Functionality	Old Script (Before SOA 2007 SP2)	Old Script (From SOA 2007 SP2 to SP4)	New Script (From SOA 2007 SP5 and further releases)
FES Server			
Memory settings	/fiorano_vars.bat	/esb/fes/bin/fes.conf	/esb/server/bin/server.conf
External jar files	/esb/fes/bin/runContainer.bat	/esb/fes/bin/fes.conf	/esb/server/bin/server.conf
Startup	/esb/fes/bin/runContainer.bat	/esb/fes/bin/fes.bat	/esb/server/bin/server.bat -mode fes
Shutdown	/esb/fes/bin/shutdownFES.bat	/esb/fes/bin/shutdown-fes.bat	/esb/server/bin/shutdown-server.bat -mode fes
Clear Database	/esb/fes/bin/clearDB.bat	/esb/fes/bin/clearDB.bat	/esb/server/bin/clearDBServer.bat -mode fes
Clear FPS repository	/esb/fes/bin/clearPeerRepository.bat	/esb/fps/bin/clearDB.bat	/esb/server/bin/clearDBServer.bat -mode fps
FES Server as Windows Service			
Install	/WinService/bin/InstallFES-NT.bat	/esb/fes/bin/service/install-fes.service.bat	/esb/server/bin/service/install-server.service.bat -mode fes
Uninstall	/WinService/bin/UnInstallFES-NT.bat	/esb/fes/bin/service/uninstall-fes.service.bat	/esb/server/bin/service/uninstall-server.service.bat -mode fes
Install a profile	Edit the following line in the file /WinService/conf/fes.conf wrapper.app.parameter.3=-fiorano.profile FES	/esb/fes/bin/service/install-fes.service.bat –profile <profile_name>	/esb/server/bin/service/install-server.service.bat -mode fes –profile <profile_name>
Uninstall a profile		/esb/fes/bin/service/uninstall-fes.service.bat –profile <profile_name>	/esb/server/bin/service/uninstall-server.service.bat -mode fes –profile <profile_name>
Configuration		Uses /esb/fes/bin/fes.conf	Uses /esb/server/bin/server.conf
Default log location	/WinService/logs	$user.dir/EnterpriseServers/$Profile/run/logs directory.	$user.dir/EnterpriseServers/$Profile/FES/run/logs directory.
FPS Server			
Memory settings	/fiorano_vars.bat	/esb/fps/bin/fps.conf	/esb/server/bin/server.conf
External jar files	/esb/fps/bin/runContainer.bat	/esb/fps/bin/fps.conf	/esb/server/bin/server.conf
Startup	/esb/fps/bin/runContainer.bat	/esb/fps/bin/fps.bat	/esb/server/bin/server.bat -mode fps
Shutdown FPS using FES	/esb/fps/bin/shutdownFPS.bat	/esb/fes/bin/shutdown-fps.bat	/esb/server/bin/shutdown-server.bat -mode fps
Shutdown FPS directly	/esb/fps/bin/shutdown.bat	/esb/fps/bin/shutdown-fps.bat	/esb/server/bin/shutdown-fps.bat
Clear DB	/esb/fps/bin/clearDB.bat	/esb/fps/bin/clearDB.bat	/esb/server/bin/clearDBServer.bat -mode fps
FPS Server as Windows Service			
Install	/WinService/bin/InstallFPS-NT.bat	/esb/fps/bin/service/install-fps-service.bat	/esb/server/bin/service/install-server.service.bat -mode fps
Uninstall	/WinService/bin/UnInstallFPS-NT.bat	/esb/fps/bin/service/uninstall-fps-service.bat	/esb/server/bin/service/uninstall-server.service.bat -mode fps
Install a profile	Edit the following line in the file /WinService/conf/fes.conf wrapper.app.parameter.3=-fiorano.profile FPS	/esb/fps/bin/service/install-fps-service.bat –profile <profile_name>	/esb/server/bin/service/install-server.service.bat -mode fps –profile <profile_name>
Uninstall a profile		/esb/fps/bin/service/uninstall-fps-service.bat –profile <profile_name>	/esb/server/bin/service/uninstall-server.service.bat -mode fps –profile <profile_name>
Configuration		Uses /esb/fps/bin/fps.conf	Uses /esb/server/bin/server.conf
Default log location	/WinService/logs	In respective $Profiles_dir/$Profiles/service directory.	$user.dir/EnterpriseServers/$Profile/FPS/run/logs directory.
Tools			
Fiorano Studio	/Studio/bin/Studio.exe	/Studio/bin/Studio.exe	/Studio/bin/Studio.exe
Fiorano Deployment Manager	/esb/tools/dm/bin/runDM.bat	/esb/tools/dm/bin/dm.bat	/esb/tools/dm/bin/dm.bat

Fiorano Mapper	/esb/tools/mapper/bin/runMapper.bat	/esb/tools/mapper/bin/mapper.bat	/esb/tools/mapper/bin/mapper.bat
FSSM	/esb/tools/fssm/bin/runFSSM.bat	/esb/tools/fssm/bin/fssm.bat	/esb/tools/fssm/bin/fssm.bat
Admin tool (NAT)	/esb/tools/fnat/bin/runNAT.bat	/esb/tools/fnat/bin/fnat.bat	/esb/tools/fnat/bin/fnat.bat
License Manager	/framework/tools/LicenseManager/bin/runLM.bat	/framework/tools/LicenseManager/bin/lm.bat	/framework/tools/LicenseManager/bin/lm.bat

The following table summarizes all the scripts that are changed in SOA 2007 for **UNIX**:

Functionality	Old Script (Before SOA 2007 SP2)	Old Script (From SOA 2007 SP2 to SP4)	New Script (From SOA 2007 SP5 and further releases)
FES Server			
Memory settings	/fiorano_vars.sh	/esb/fes/bin/fes.conf	/esb/server/bin/server.conf
External jar files	/esb/fes/bin/runContainer.sh	/esb/fes/bin/fes.conf	/esb/server/bin/server.conf
Startup	/esb/fes/bin/runContainer.sh	/esb/fes/bin/fes.sh	/esb/server/bin/server.sh -mode fes
Shutdown	/esb/fes/bin/shutdownFES.sh	/esb/fes/bin/shutdown-fes.sh	/esb/server/bin/shutdown-server.sh -mode fes
Clear Database	/esb/fes/bin/clearDB.sh	/esb/fes/bin/clearDB.sh	mode fes
Clear FPS repository	/esb/fes/bin/clearPeerRepository.sh	/esb/fps/bin/clearDB.sh	mode fps
FPS Server			
Memory settings	/fiorano_vars.sh	/esb/fps/bin/fps.conf	/esb/server/bin/server.conf
External jar files	/esb/fps/bin/runContainer.sh	/esb/fps/bin/fps.conf	/esb/server/bin/server.conf
Startup	/esb/fps/bin/runContainer.sh	/esb/fps/bin/fps.sh	/esb/server/bin/server.sh -mode fps
Shutdown FPS using FES	/esb/fps/bin/shutdownFPS.sh	/esb/fes/bin/shutdown-fps.sh	/esb/server/bin/shutdown-server.sh -mode fps
Shutdown FPS directly	/esb/fps/bin/shutdown.sh	/esb/fps/bin/shutdown-fps.sh	/esb/server/bin/shutdown-fps.sh
Clear DB	/esb/fps/bin/clearDB.sh	/esb/fps/bin/clearDB.sh	/esb/server/bin/clearDBServer.sh -mode fps
Tools			
Fiorano Studio	/Studio/bin/Studio.sh	/Studio/bin/Studio.sh	/Studio/bin/Studio.sh
Fiorano Deployment Manager	/esb/tools/dm/bin/runDM.sh	/esb/tools/dm/bin/dm.sh	/esb/tools/dm/bin/dm.sh
Fiorano Mapper	/esb/tools/mapper/bin/runMapper.sh	/esb/tools/mapper/bin/mapper.sh	/esb/tools/mapper/bin/mapper.sh
FSSM	/esb/tools/fssm/bin/runFSSM.sh	/esb/tools/fssm/bin/fssm.sh	/esb/tools/fssm/bin/fssm.sh
Admin tool (NAT)	/esb/tools/fnat/bin/runNAT.sh	/esb/tools/fnat/bin/fnat.sh	/esb/tools/fnat/bin/fnat.sh
License Manager	/framework/tools/LicenseManager/bin/runLM.sh	/framework/tools/LicenseManager/bin/lm.sh	/framework/tools/LicenseManager/bin/lm.sh

2.6.3 Configuring the Jetty Server with SSL Support

SSL support for Jetty has been provided. The User can configure the SSL parameters for Jetty, running with FES/FPS by editing the corresponding profiles.

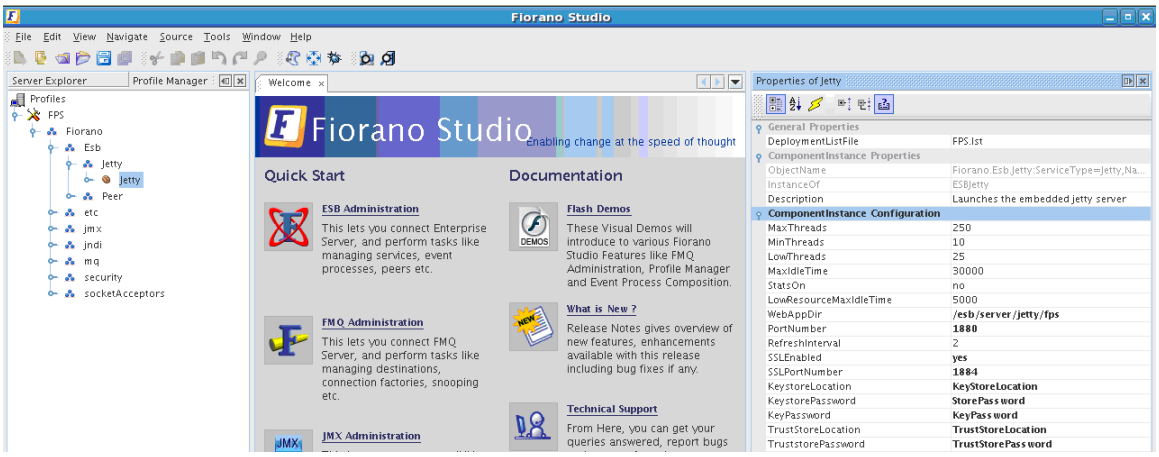


Figure 2.6.3: Configuring Jetty for FPS

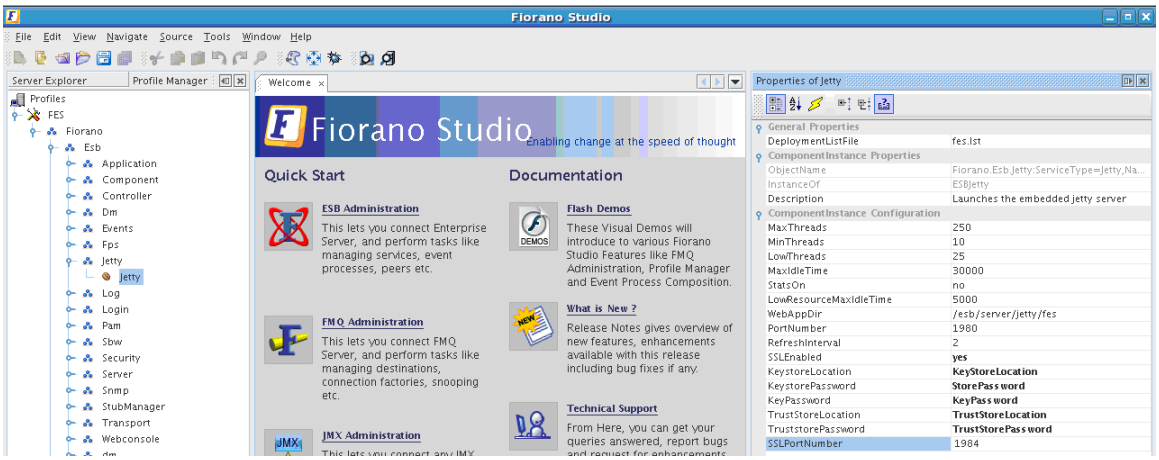


Figure 2.6.4: Configuring Jetty for FES

2.6.3.1 SSL Configuration for Jetty

You can configure the SSL parameters for Jetty which is running with FES/FPS by editing the corresponding profiles. Configuration of SSL support for WSStub and HttpStub are described in the following section.

2.6.3.1.1 Configuring SSL parameters for Jetty

1. By default the SSL property for Jetty is disabled. To enable this property, open FES/FPS profile in Studio esb-> Jetty, change **EnableSSL** to **Yes**.
2. Specify the values for KeyStoreLocation, KeyStorePassword, KeyPassword, TrustStore, TrustStorePasswd and save the FES/FPS profile.

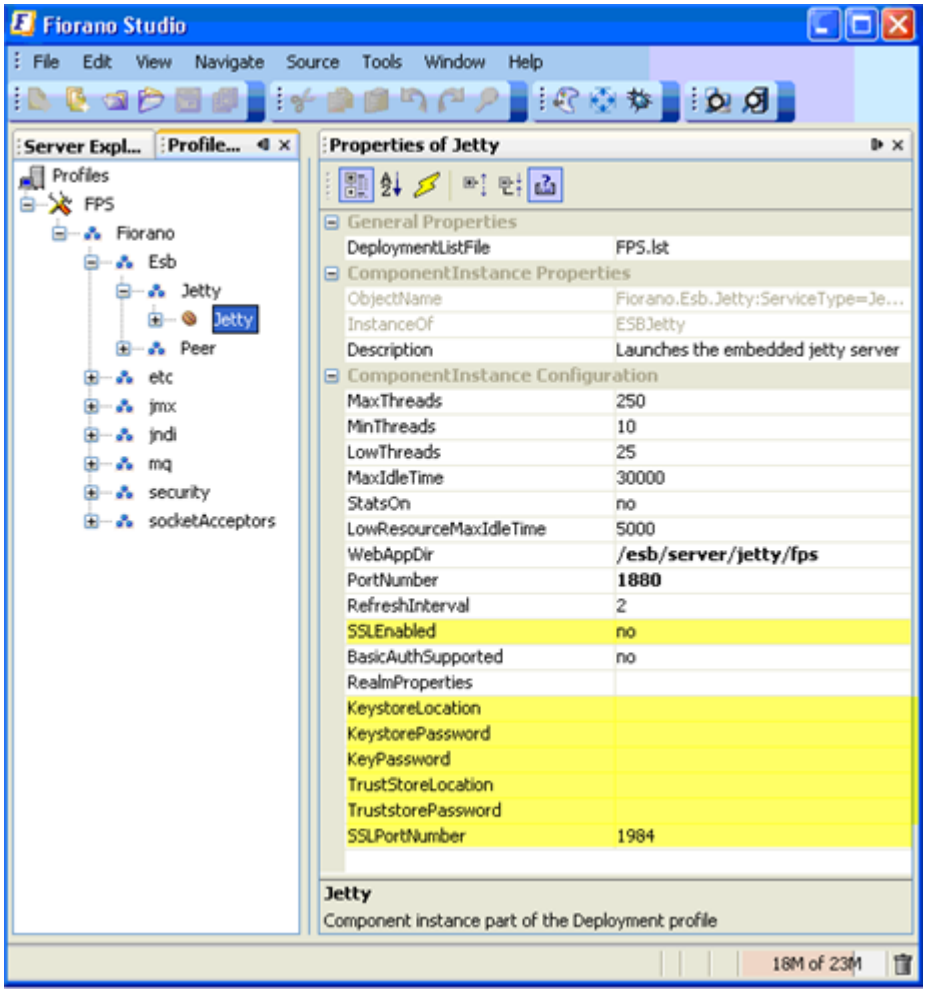


Figure 2.6.5: Values for FES/FPS profile

3. Start the servers. Jetty is started with the SSL enabled.

2.6.3.1.2 Configuring SSL support for WSStub and HttpStub

To use WSStub/HttpStub with SSL Support, the Jetty Server running in FPS needs to be started with SSL support. To check if Jetty is started with SSL support, check the URL https://<IP Address>:<Port Number>/bcwsgateway. If the URL is working, this means the Jetty with FPS is started with SSL Support.

SSL properties can be configured for WSStub and HttpStub through CPS.

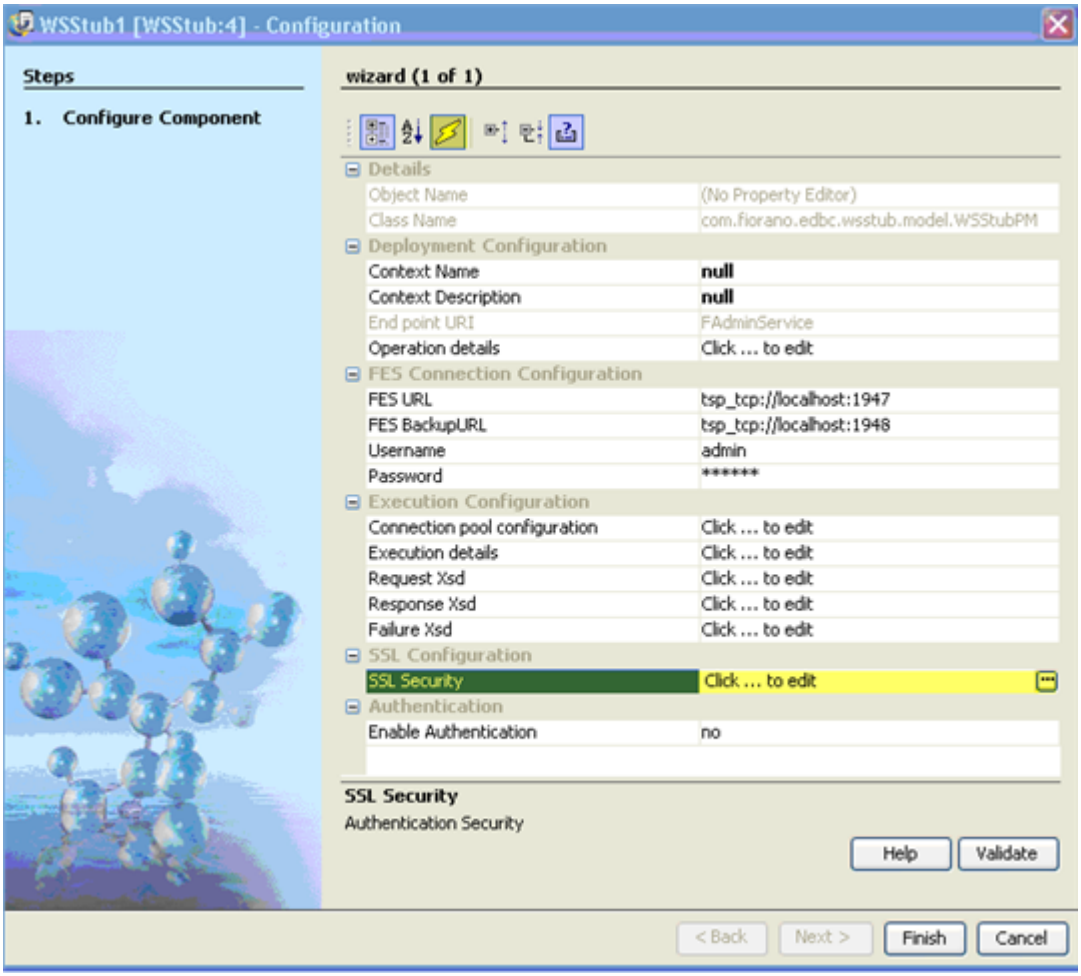


Figure 2.6.6: Properties of SSL

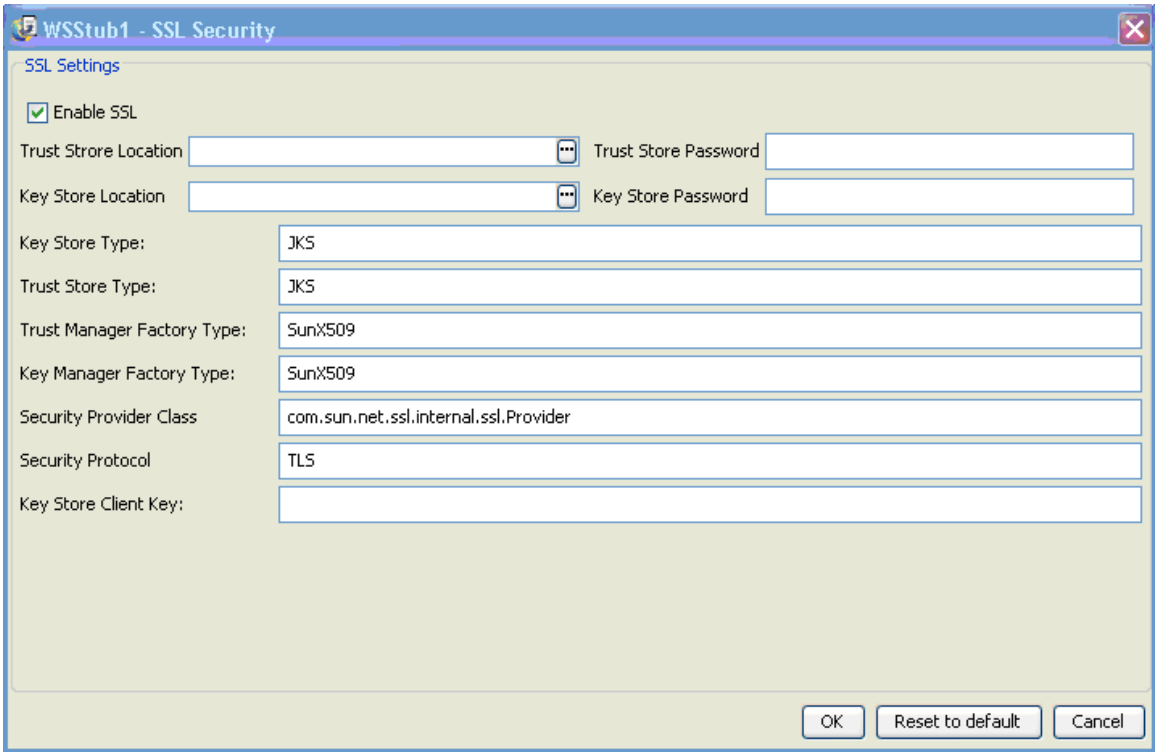


Figure 2.6.7: SSL Security dialog box

After configuring the SSL parameters through CPS, launch the Event Process. If WSStub does not start properly, check the SSL configuration with WSStub CPS.

2.6.3.1.3 Testing Web Service from Dashboard

Web service can be tested from dashboard by clicking the **Test** button and giving the input parameters.

2.6.3.1.4 Testing Web Service from WebServiceConsumer

1. After Launching the WSStub, get the WSDL URL (right-click on stub →view WSDL.) Open the CPS of WSConsumer and provide the URL.

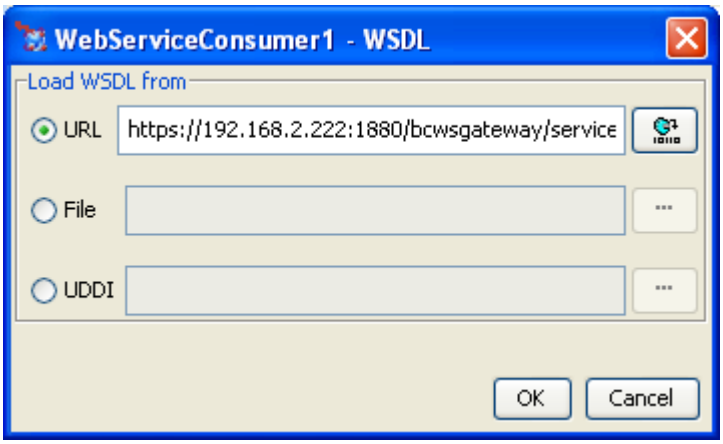


Figure 2.6.8: WebServiceConsumer1 - WSDL dialog box

2. Configure WSConsumer for SSL through CPS.

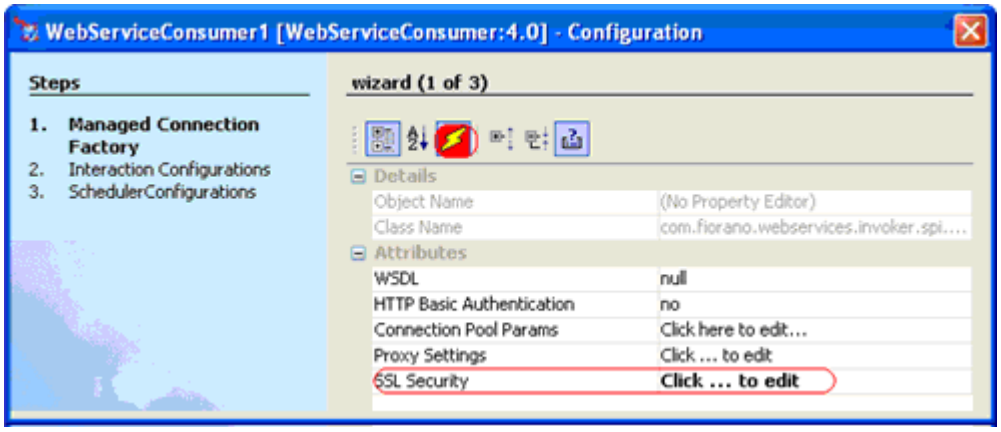


Figure 2.6.9: Configure WSConsumer

3. Enable SSL and provide TrustStore location, TrustStorePassword, KeyStoreLocation and KeyStorePassword.

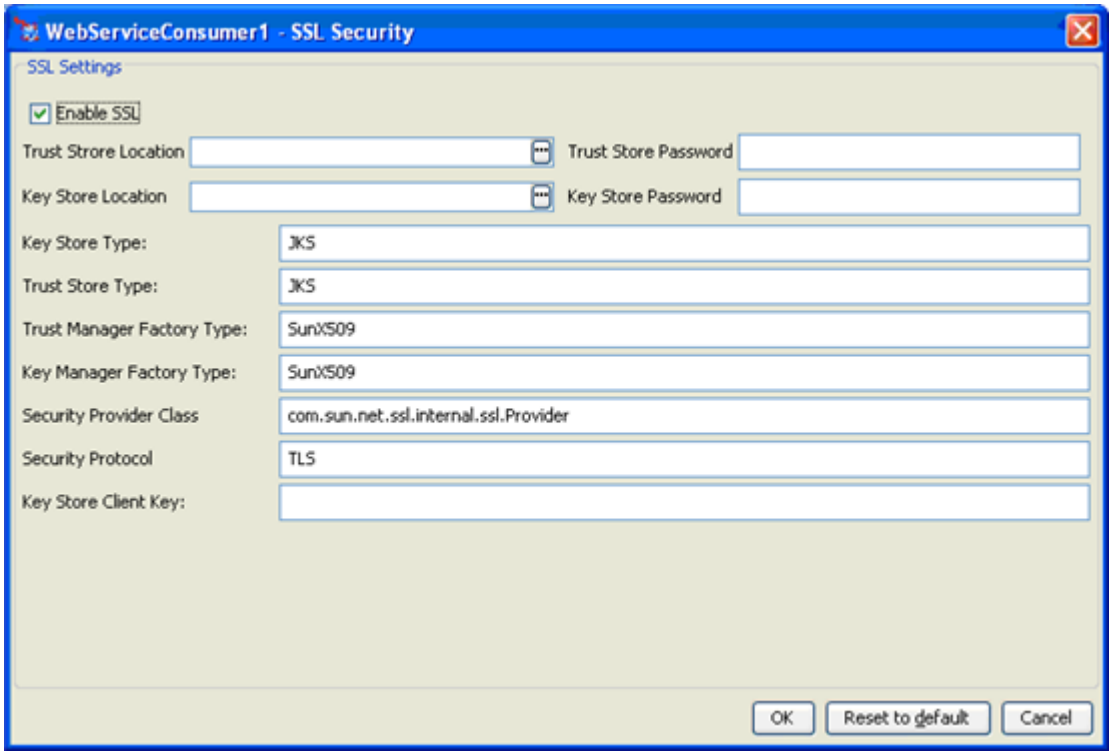


Figure 2.6.10: Enabling SSL option

4. The Web Servie configured for SSL from WebServiceConsumer can now be invoked.

2.6.4 Using Basic Authentication with the Jetty Server

2.6.4.1 Configuring Jetty Server

In FPS Jetty Server, basic Authentication needs to be enabled as the Stub component is running in the FPS server.

1. Before starting the FPS server, start the Studio and open the FPS profile from profile management.
2. Go to FPS->Fiorano->Esb-Jetty. In the properties of Jetty, set Basic Authentication to **yes** and give the fully qualified path of the Realm.properties file.

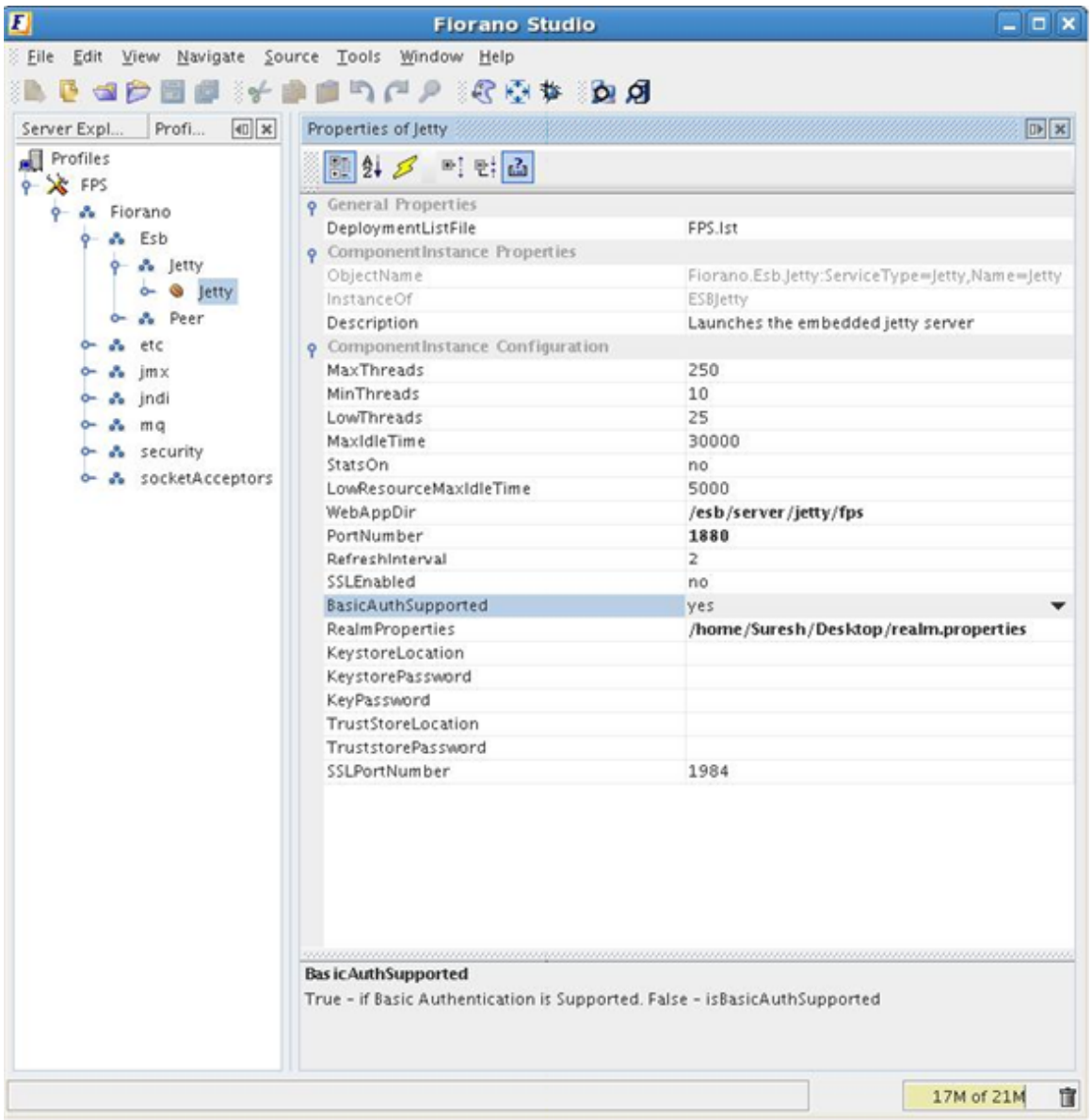


Figure 2.6.11: Enabling Basic Authentication

3. Save the profile and Close.

2.6.4.2 Enabling Basic Authentication with bcwsgateway

1. Open Web.xml in %FIORANO_HOME%/esb/server/jetty/fps/webapps/bcwsgateway/WEB- INF
2. Uncomment the security-constraint and login-config tags. Save this and then close.
3. Start the Server and login onto the Studio. Configure WSStub.

2.6.4.3 Enabling Basic Authentication with WSStub

1. In Transport Properties, set HTTP Authentication to yes and select the type as BASIC. Give the username and password that are present in Realm.properties.

Note: Refer 4.14.1.1.7.2 section from WSStub documentation for more information.

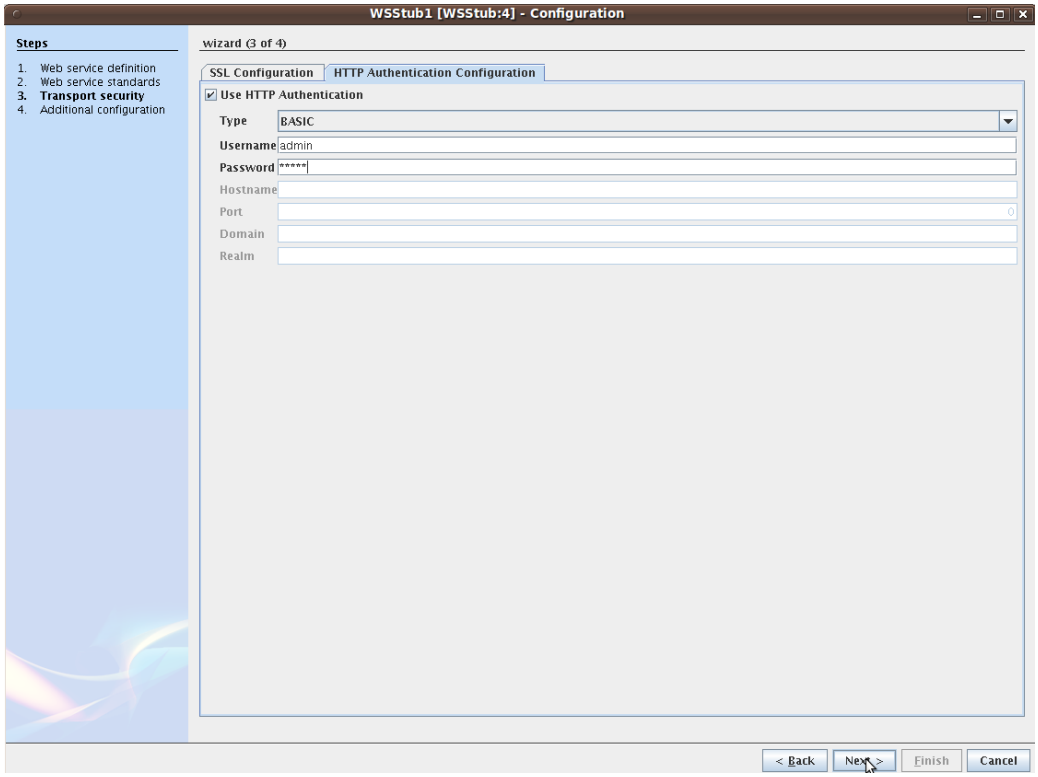


Figure 2.6.12: Enabling Expert Properties

2. Launch the flow

2.6.4.4 Testing Services from Dashboard

1. Go to the Services tab in Dashboard. Click the **Test** link to display the Test dialog box.
2. Enable **Set Basic Authentication** and enter Username and Password in the WSStub configuration and click the Test button to perform the test.

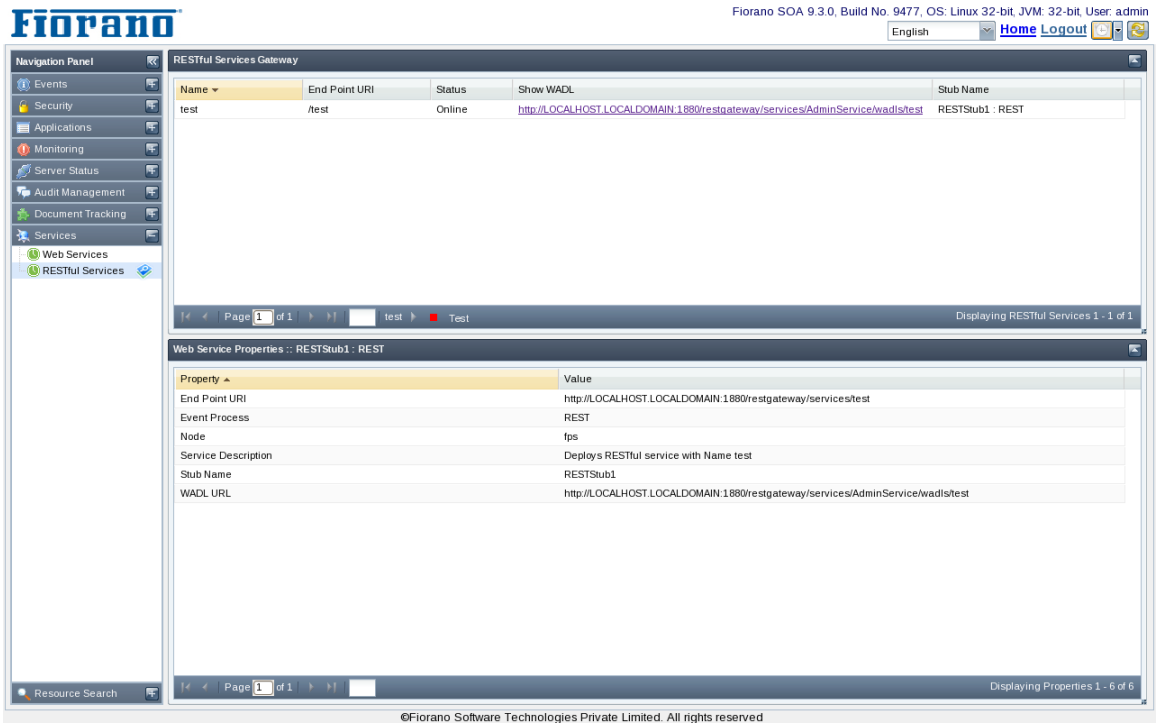


Figure 2.6.13: Enabling Set Basic Authentication

2.6.4.5 Testing Services from Web Service Consumer

1. Configure the WSC 4.0. Set **Http Basic Authentication** to **yes** and enter the user name and password in the WSStub. Specify the WSDL URL and click the **Next** button.

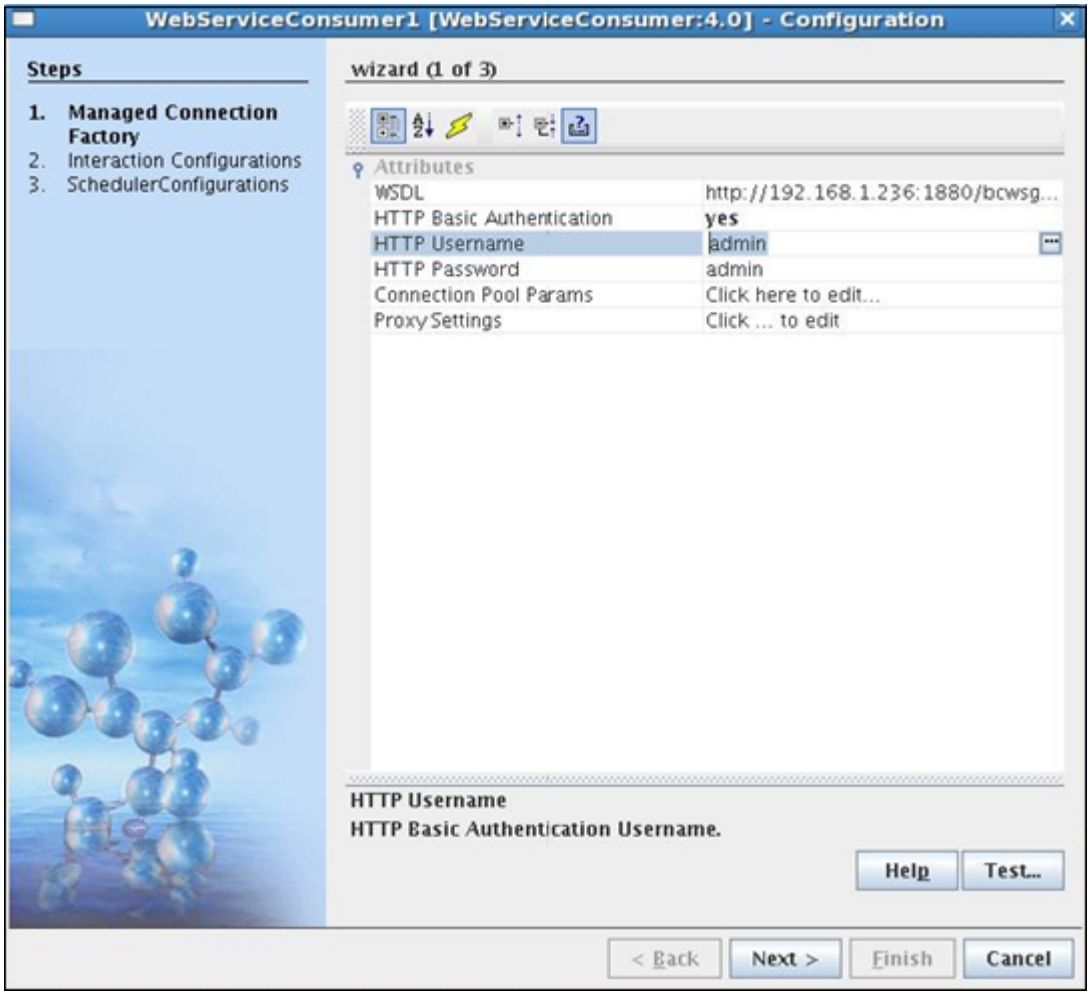


Figure 2.6.14: Enabling Http Basic Authentication

2. Click on the **Call Properties**, add the javax.xml.rpc.security.auth.username and javax.xml.rpc.security.auth.password properties, and then enter the values. Finally perform the test.

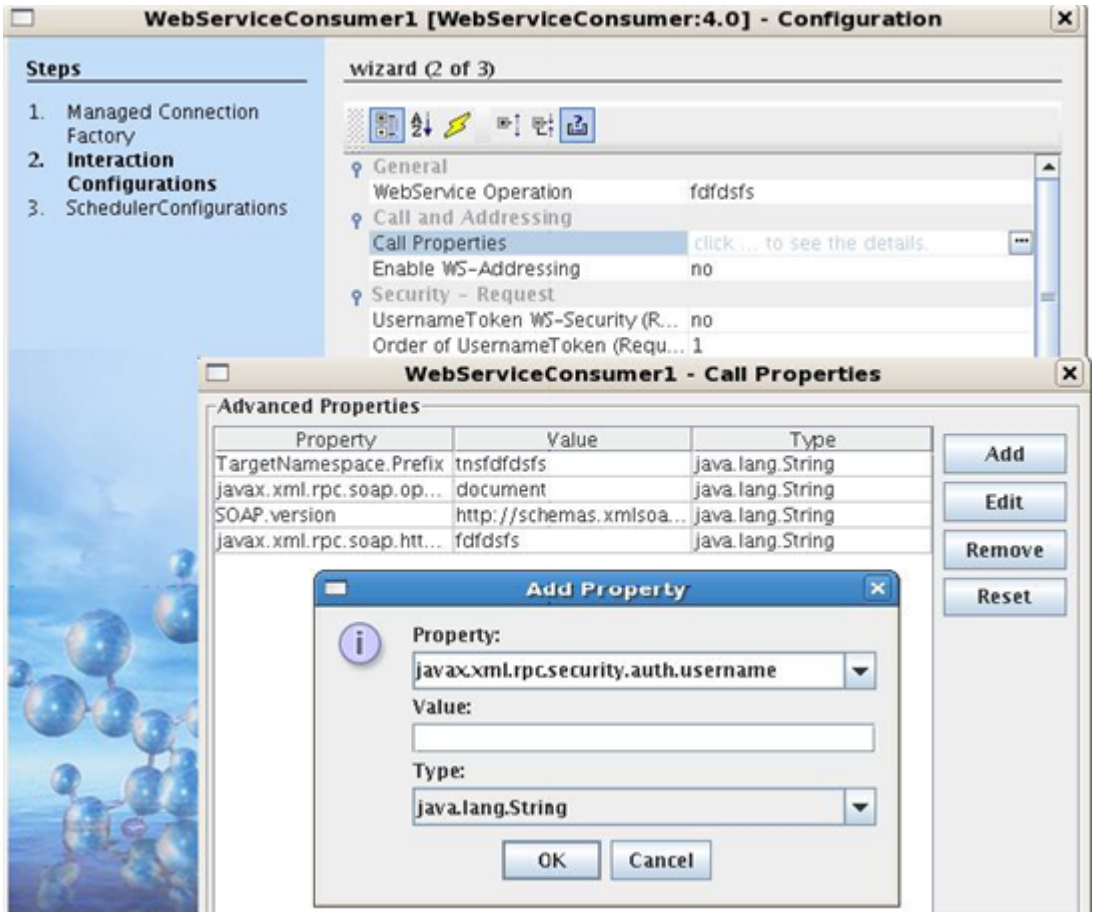


Figure 2.6.15: Add Properties dialog box

2.6.5 Adding Additional Port for Peer to Peer Communication

The Fiorano Peer Server accepts internal connections from other Peer Servers as well as from Fiorano Adapters at a single port. The Fiorano Peer Server default ports for profile1 and profile2 are 1867 and 1877 respectively.

An additional socket acceptor can be added for exclusive communication between Fiorano Peer Servers using Fiorano Studio.

Note: This configuration is done in offline mode

1. Launch the Fiorano Studio. Open the **Profile Manager** and open the profile in which the Socket Acceptor is to be added.
2. In the profile, select the domain to which the new Socket Acceptor is to be added. The default Fiorano Peer Server profile has socket acceptors at the following node: in the tree Fiorano -> socketAcceptors. Fiorano recommends adding a new sub-domain (for instance, port-2) in this domain and also adding a new socket acceptor.
3. Right-click on the desired domain and select **Add Components**. The **Add Components to Profile** dialog box appears. Navigate to Fiorano -> FioranoFw -> Services and select the component Connection Manager1 from the new dialog box.

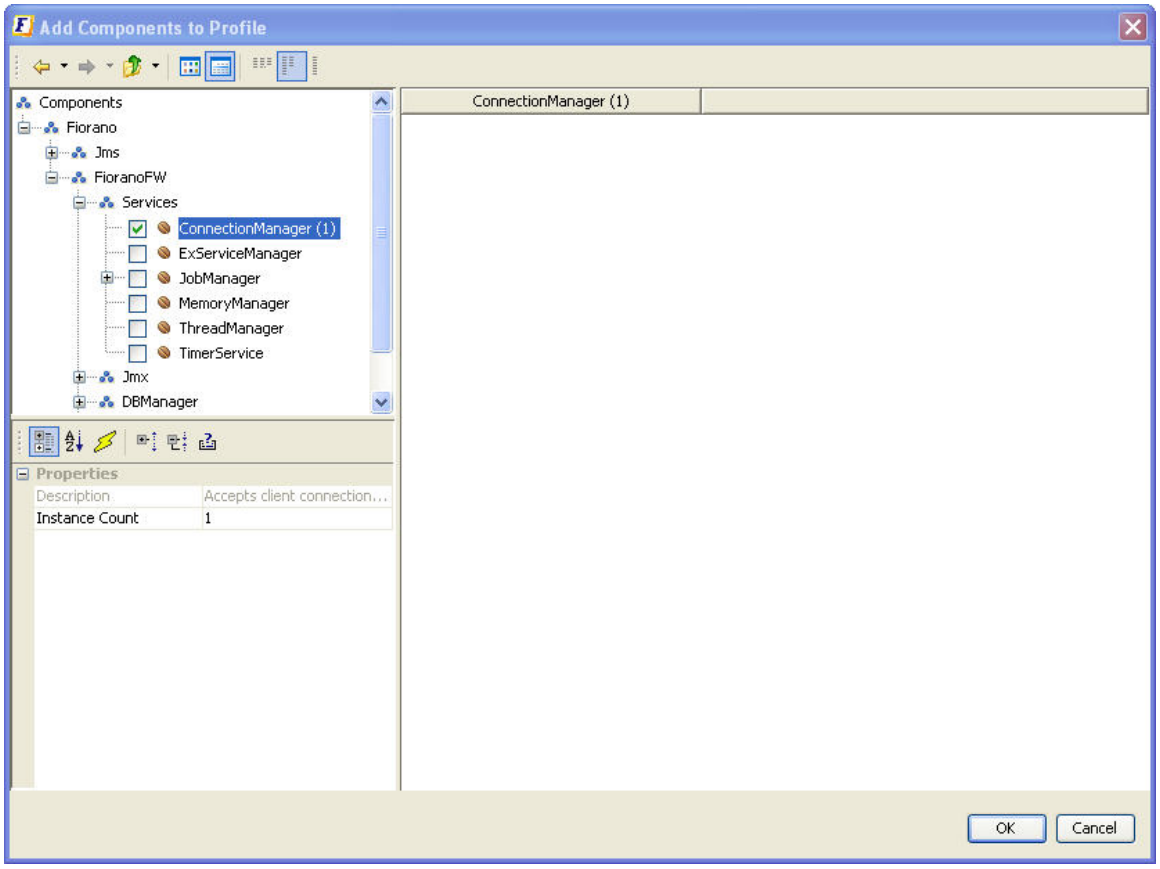


Figure 2.6.16: Adding Components

4. Click the **OK** button to add the selected instance(s) to the profile. The dependencies of the newly added component(s) have to be resolved. All un-resolved dependencies are marked with a red error icon.

Note: Besides the connection Manager instance, an instance of the SocketReadHandler gets included into the profile.

5. To resolve dependencies, open the **DependsOn** property of the newly added Connection Manager and the associated SocketReadHandler. For each dependency marked with a red icon signifying an error, select the desired instance from the drop-down list shown in the properties field.

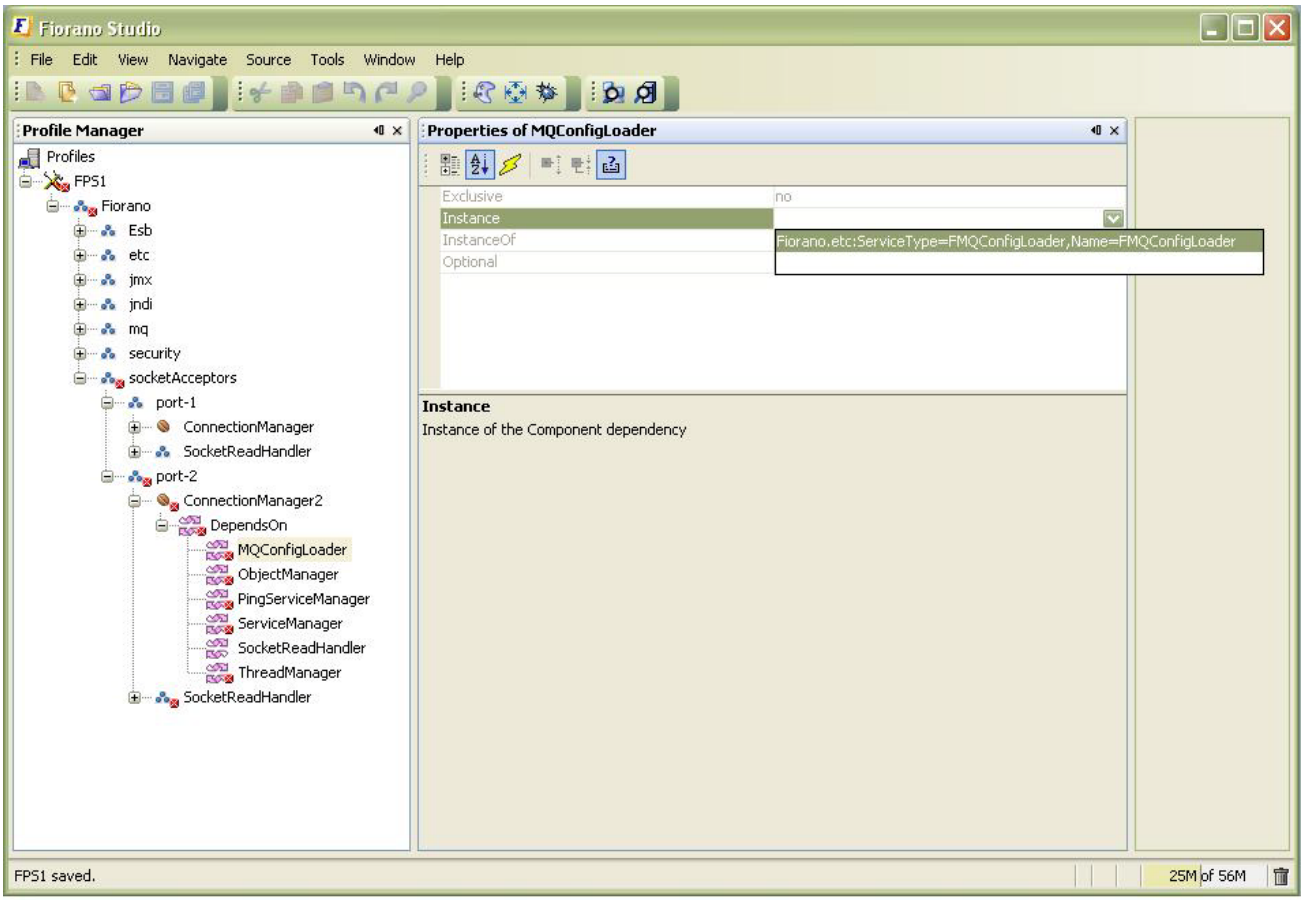


Figure 2.6.17: Adding Components

Note: Any existing instance for a dependency can be used to resolve it.

6. Right-click on the profile root node and select **Validate** to ensure that all dependencies have been resolved.
7. Navigate to the node Fiorano -> socketAcceptors -> port-2 -> ConnectionManager2. In the properties window, change the default port number displayed to the desired value and set the **UseForPeerToPeerCommunication** property to **yes**. Additionally, set the properties **Default** and **FMQServer** properties to **no**.

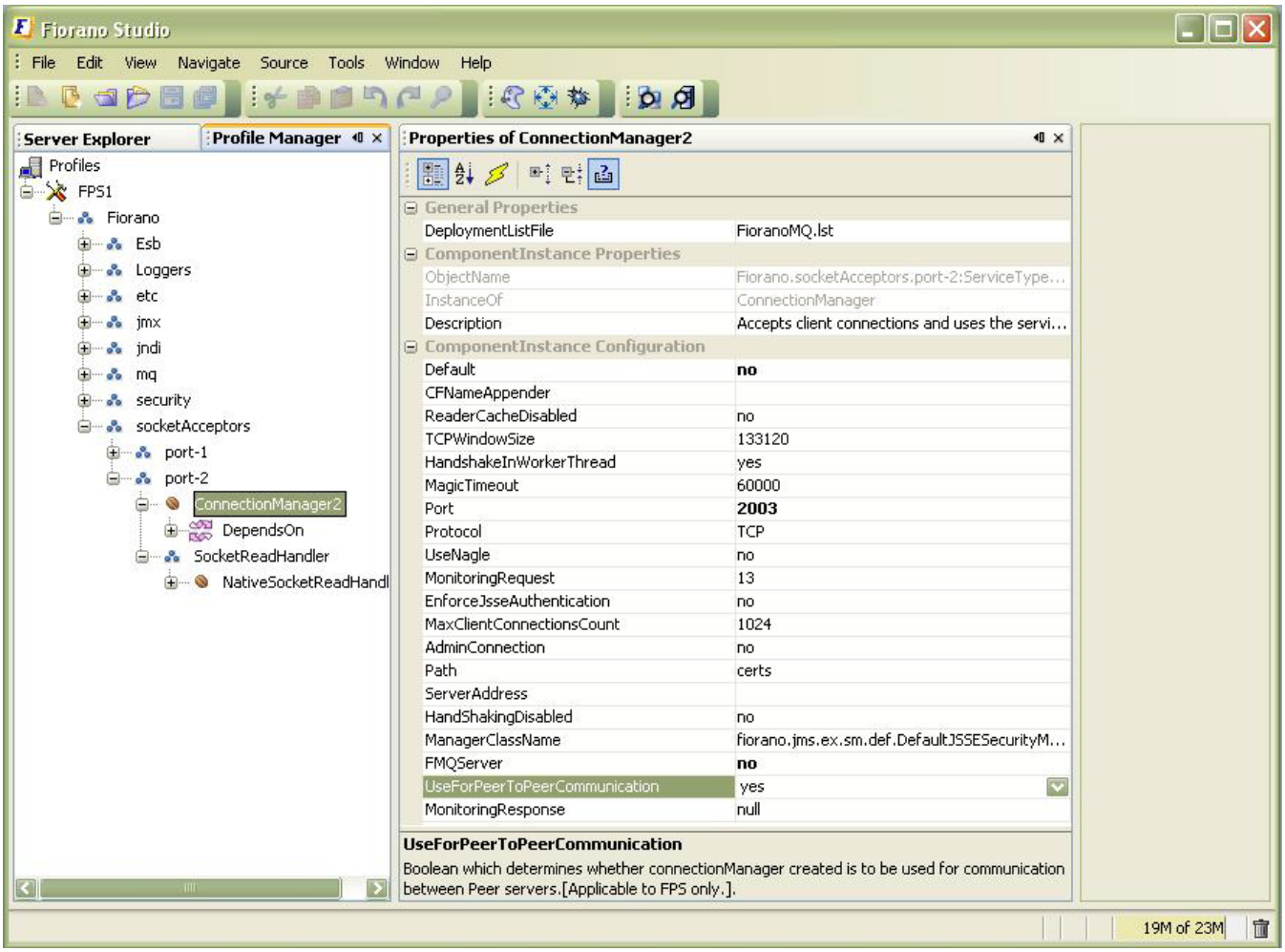


Figure 2.6.18: Adding Components

8. Save the profile.

Note: If the protocol of the Connection Manager is set to **SUN_SSL** or **HTTPS_SSL**, an additional **FMQConfigLoader** should be added to the profile. This can be done by performing the following steps:

1. Navigate to the node Fiorano -> etc. Right-click on **Fiorano** node and select **Add Components**. The **Add Components to Profile** dialog box appears.
2. Navigate to Fiorano -> Jms -> Services and select the **FMQConfigLoader (1)** from the new dialog box and click **OK**.

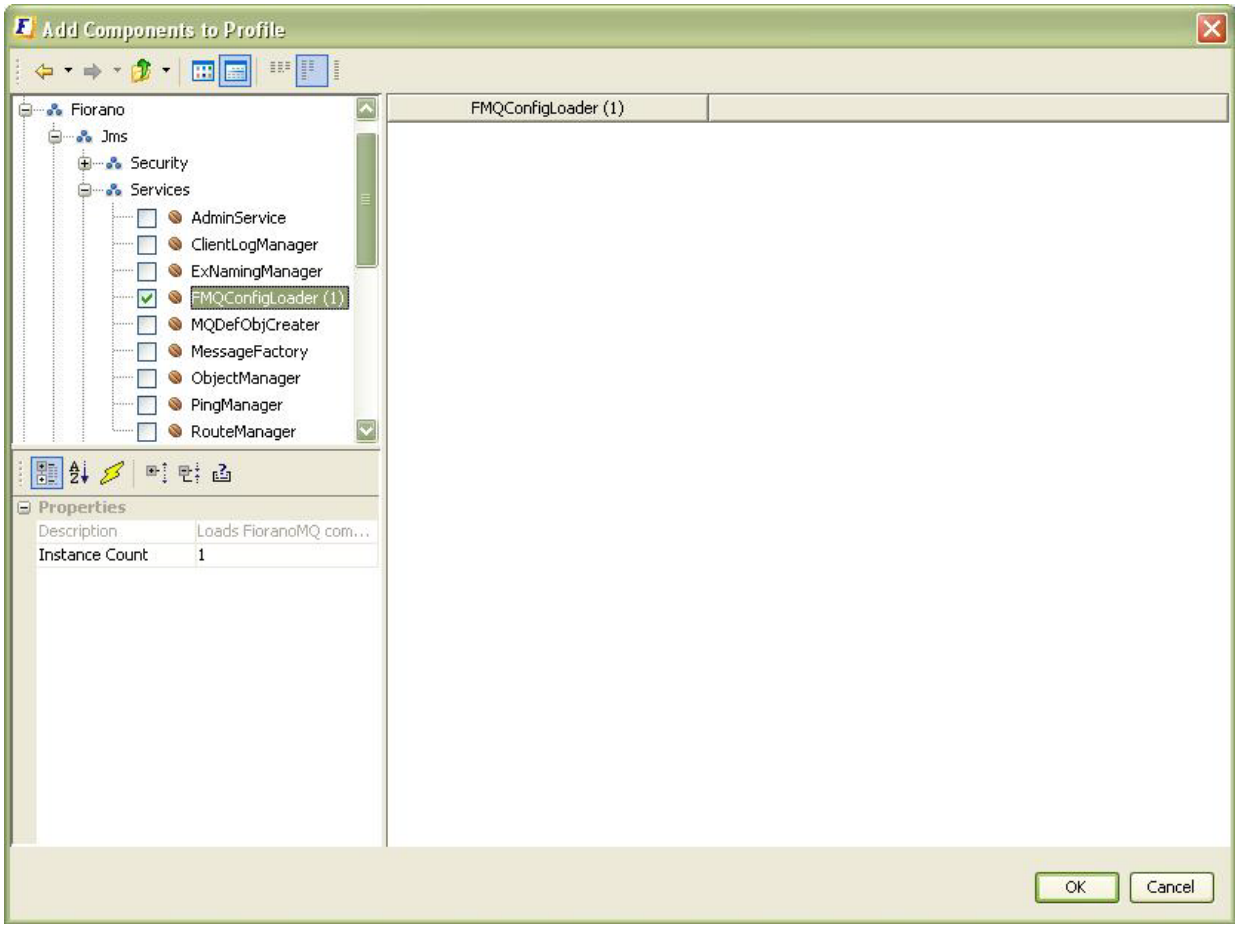


Figure 2.6.19: Adding Components to profile dialog box

3. Select the newly created FMQConfigLoader node under Fiorano -> etc and change the **SSL Enabled** property to **yes** in the properties pane.

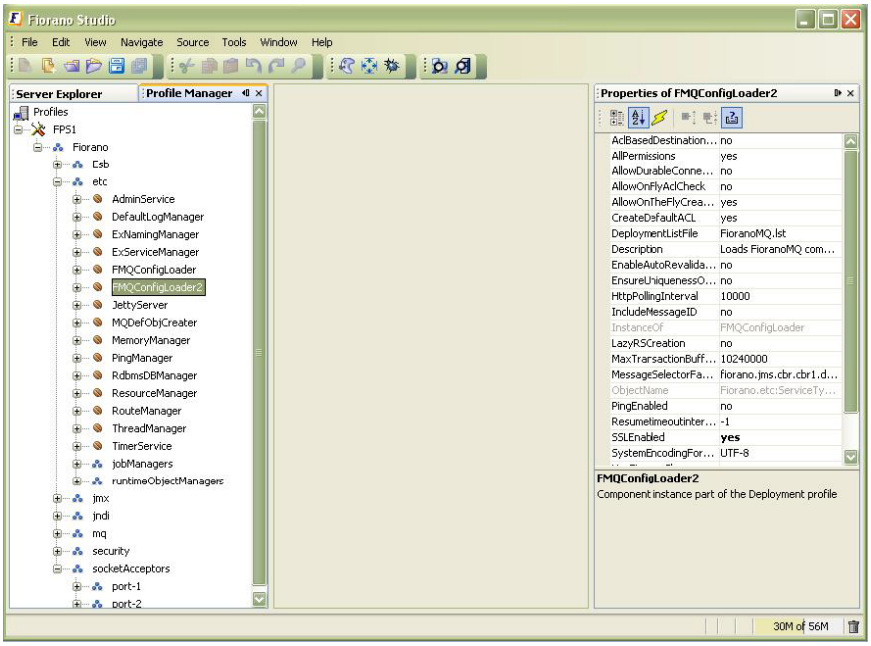


Figure 2.6.20: Enabling SSL

4. Navigate to Fiorano -> Socket Acceptors -> port-2 -> Connection Manager2. Under the **Depends On** node, select the dependency **MQConfigLoader** and select its instance as the newly added FMQConfigLoader (that is, FMQConfigLoader2 in this case).

5. Save the Profile.

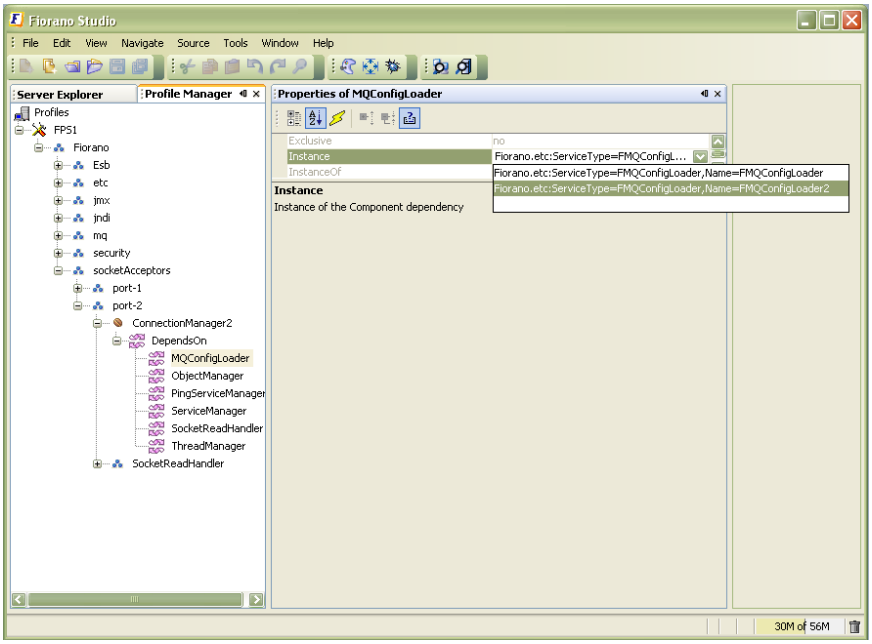


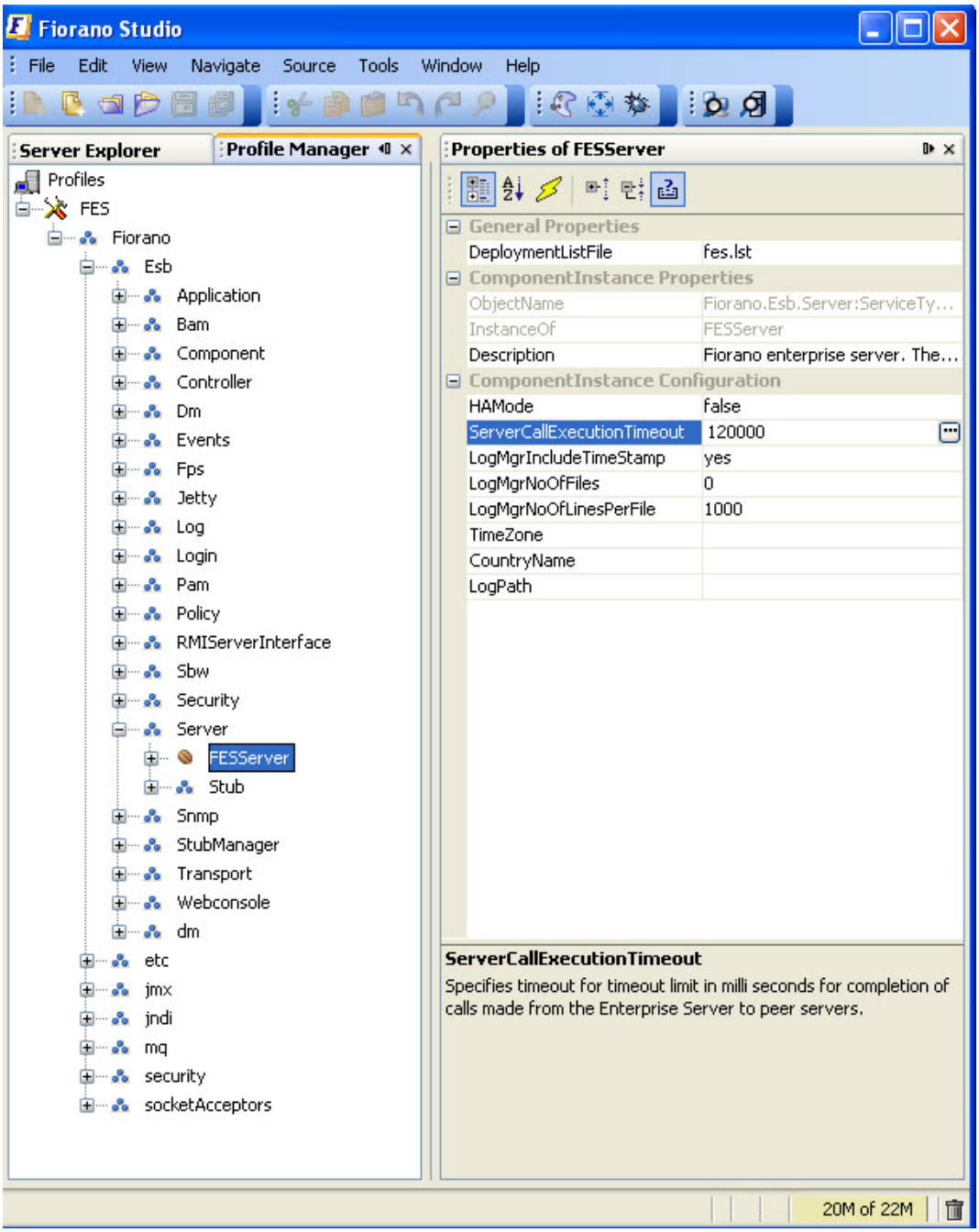
Figure 2.6.21: Selection of Instance

2.6.6 Configuring Server Execution Timeout

When running Fiorano event processes with many components, some calls may get timed out. This means that the communication between the Enterprise Server and Peer Server had taken more time than the default time. The default value is 2 minutes.

The timeout value can be configured from FES profiles. Open the FES profile from Studio. Navigate to FES->Fiorano->ESB->Server->FESServer.

Edit the ServerCallExecutionTimeout property. Specify the value in milliseconds. If the property value is 0, the timeout is infinite.



2.6.7 SNMP Configuration

The Fiorano network can be monitored through SNMP Managers. The Fiorano Enterprise Server has an SNMP Agent that handles MIB requests and sends traps carrying status information about all the Peer Servers, Event Processes, and Business Components running within a Fiorano network.

The MIB file, fiorano_mib.txt, that an SNMP Manager needs for monitoring a Fiorano network is available in $FIORANO_HOME\ esb\server\profiles\<profile_name>\FES\conf. This file contains the Fiorano SOA Platform MIB tree required for monitoring Peer Servers, Event Processes, and Business Components.

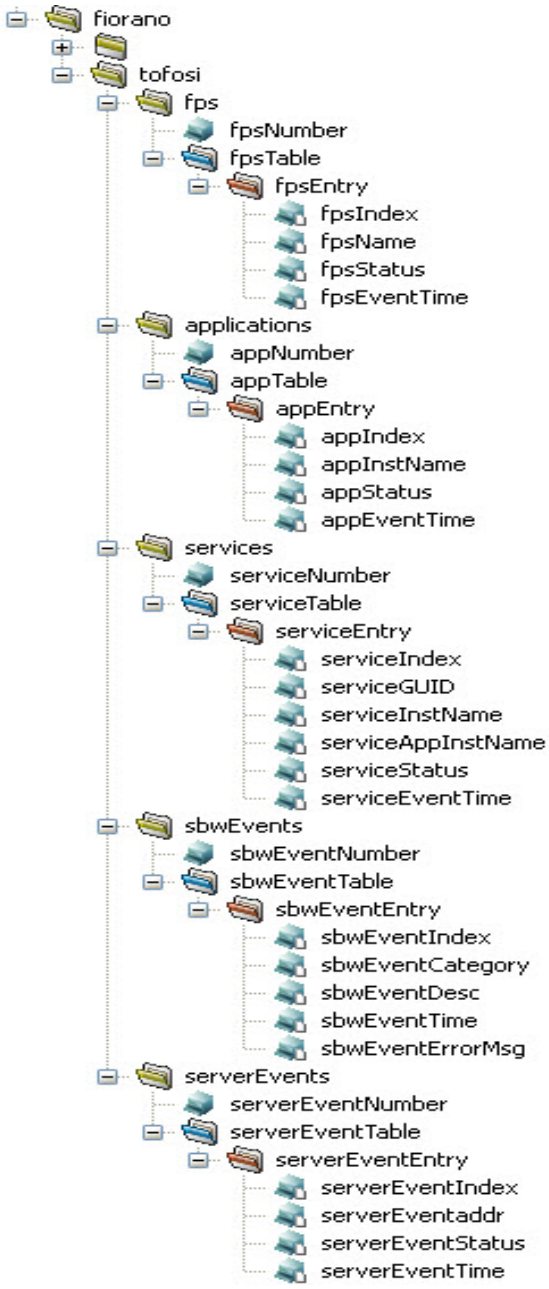


Figure 2.6.22: Fiorano SOA Platform MIB tree

2.6.7.1 Fiorano SOA 9 Platform MIB

As discussed in the previous section, MIB declares the information the agent can provide. The Fiorano SOA Platform MIB gives an idea about the information that can be extracted from the Fiorano SOA Platform Agent. The present implementation is SNMPv1 compliant. The Fiorano SOA Platform MIB is the first sub-tree in the Fiorano sub-tree. The structure of the MIB is shown in the Figure 2.6.22.

2.6.7.2 Fiorano SOA 9 Platform Agent

The Fiorano SOA Platform Agent is an implementation that provides the values for the objects defined in the MIB above. It maintains its own database of values, which is updated when Fiorano SOA Platform performs events like launch or the shutdown of a FPS occurs. The Agent is programmed to respond to SNMP queries from the manager. Currently the data is in read-only form for the manager and setting values by the SET operation is not permitted. Additionally, the Agent can be configured to send trap messages to the manager on occurrence of alerts and/or events.

2.6.7.3 How to receive SNMP Traps

The Fiorano network can be monitored through SNMP Managers. The Fiorano Enterprise Server has an SNMP Agent that handles MIB requests and sends traps carrying status information about all the Peer Servers, Event Processes, and Business Components running within a Fiorano network. The MIB file, fiorano_mib.txt, that SNMP Manager needs for monitoring a Fiorano network is available in $FIORANO_HOME\ esb\server\profiles\<profile_name>\FES\conf. This file contains the Fiorano SOA Platform MIB tree required for monitoring Peer Servers, Event Processes, and Business Components.

2.6.7.4 Getting the information

In order to utilize the SNMP aware Fiorano SOA Platform system, all you need to do is configure a SNMP manager (popular ones include HP OpenView and IBM Tivoli NetView) of your choice by loading the Fiorano MIB into it. The manager can now be used to view the data. The agent provides information via six tables-

1. The FPS table (OID = 1.3.6.1.4.1.7163.1.1.2)

   This table provides the data about the Fiorano Peer Servers configured with the Enterprise Server. It shows the following details:

   a. fpsname: This is the unique name of the Fiorano ESB Peer as configured in the Fiorano Network.

   b. fpsStatus: This field shows the last event received for this peer server.

   c. fpsEventTime: This represents the time at which the last event occurred.

2. The Applications table (OID = 1.3.6.1.4.1.7163.1.2.2)

This following show the data for the Event Processes that are currently or those that were running on the network. It provides the following information:

a. appInstanceName: The name of the Event Process instance.

b. appStatus: The status of the Event Process is the last event that was received with respect to the concerned Event Process.

c. appEventTime: The time at which the last event occurred.

3. The services table (OID = 1.3.6.1.4.1.7163.1.3.2)

   This table gives the information regarding the Business Components that are/were running on the Peer Servers that the managed Enterprise Server is connected to. The fields available are the following:

   a. serviceGUID: This is a unique string that identifies a Business Component available on a Fiorano ESB Server.

   b. serviceInstanceName: Since many instances on the Business Component may be running, a ComponentInstanceName is used to identify the particular instance of the concerned Business Component.

   c. c. serviceAppInstanceName: This identifies the name of the Event Process as a part of which this particular Business Component is/was running

   d. serviceStatus: The status of the Business Component that was last received by the Enterprise Server.

   e. serviceEventTime: The time at which the last event occurred.

4. The SBW events table (OID = 1.3.6.1.4.1.7163.1.4.2)

   This table gives the information regarding ON_EXCEPTION events of SBW tracked messages. The fields available are the following:

   a. sbwEventCategory: The event category is a Warning as only ON_EXCEPTION events are tracked.

   b. sbwEventDesc: This gives the details of the particular port of service on which the event is tracked as well as the Application Instance to which the service belongs to.

   c. sbwEventTime : The time at which the last event occurred

   d. sbwEventErrorMessage: This gives details such as error code and error message.

5. The Server Events(1.3.6.1.4.1.7163.1.5.2)

   This table gives the information regarding the status of the gateway server that is configured to run the Fiorano servers in HA mode. The event will be generated only when the server is running in HAMode.

   a. serverEventaddr: The gateway server machine address.

   b. serverEventStatus: This gives the details whether gateway server is running or not.

   c. serverEventTime: Time at which the last event occurred

6. The Ha Server and GateWay State Events (OID = 1.3.6.1.4.1.7163.1.6.2)

This table gives Ha Server and GateWay server state change Status Events.

a. haServerStateEventsaddr: The Ha Server and GateWay machine addresses.

b. haServerStateEventStatus: This gives the details of Ha server status and status whether GateWay server is running or not

c. haServerStateEventsTime: Time at which the last event occurred.

2.6.7.4 Trap Messages

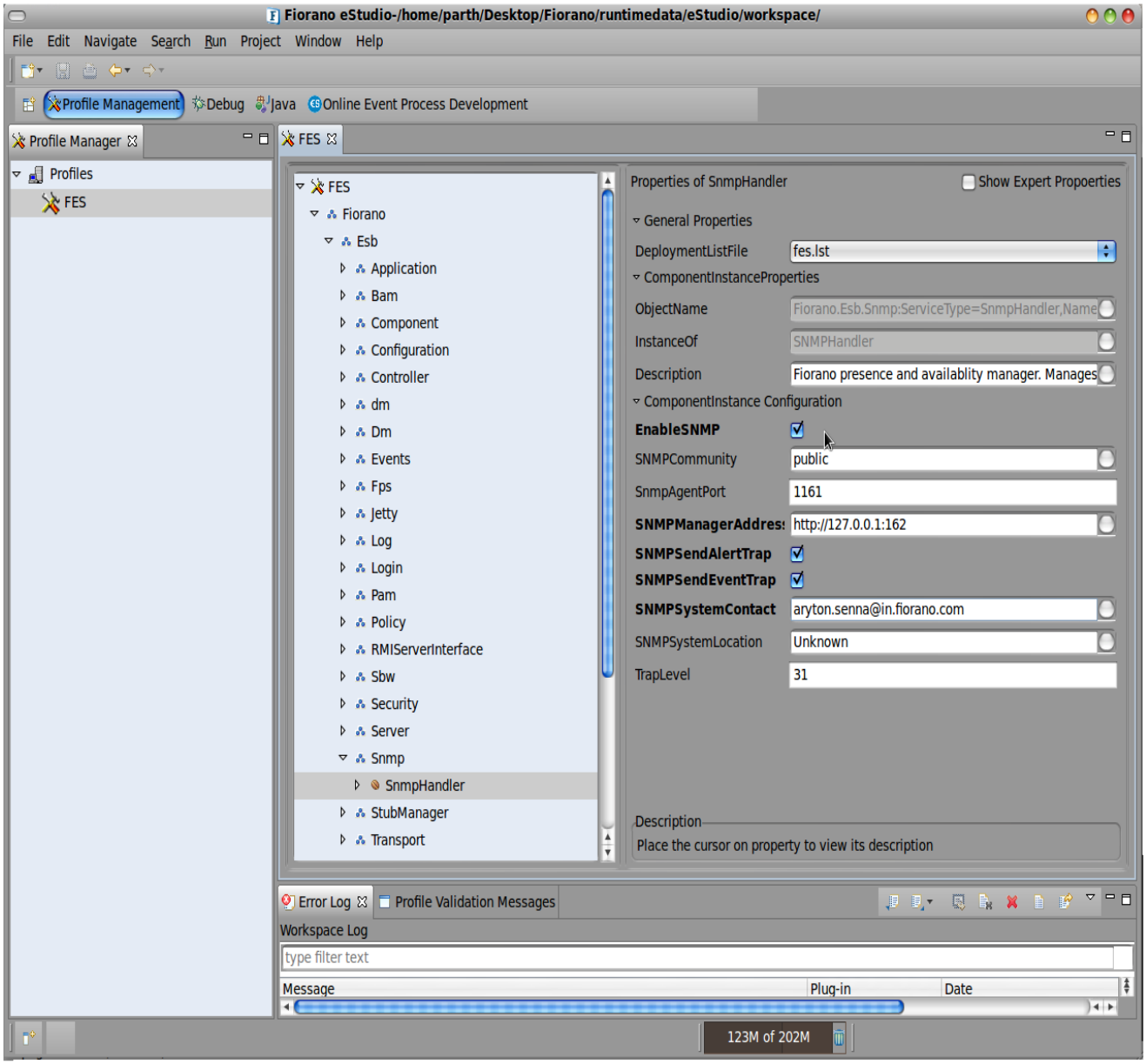
Fiorano SOA Platform can be configured to send trap messages on occurrence of alerts and events, if so desired. This information can be sent to a pre-configured manager address or broadcast on the network for all managers listening, so as to enable them to pick up this message. Trap messages can provide valuable and instant information about faults and failures. The following Fiorano SOA Platform specific trap types are presently configured on occurrence of events.

1. A FPS Event has occurred.
2. An Event Process Event has occurred.
3. A Business Component Event has occurred.
4. An SBW On Exception Event has occurred.
5. A server event has occurred.
6. A HA Server and GateWay State Events has occurred.

The above trap numbers can be configured as enterprise specific traps in the manager for appropriate action to be taken as when they are raised.

2.6.7.5 Configuring the SNMP parameters for the Fiorano ESB Server

This section deals with enabling and customizing the Fiorano SOA Platform agent. This can be done easily through the Profile Manager in Studio. Go to the profile intended to be changed->Fiorano->esb->Snmp



1. Enabling SNMP

   This can be done by adding the line ENABLE_SNMP=yes. The default value is false.

2. Setting the SNMP community

   This is done by setting the SNMP.COMMUNITY variable.

   For example, SNMP.COMMUNITY=tifosi. The default value is public.

3. SNMP Agent Port

   The port the SNMP agent will listen on. The default SNMP port number is 1161

4. SNMP manager URL

This URL specifies hostname and event port to which the SNMP traps are sent. By default all SNMP managers listen for traps on event port 162. The default URL is http://localhost:162.

For instance, if you need to send traps to a machine named network_manager on event port 162, you can set it as follows:

SNMP.MANAGER_ADDRESS=http://network_manager:162.

5. Enable sending traps on occurrence of alerts

   This tells the FES to send an SNMP trap message to the configured manager in case of alerts being raised. The default is disabled. To enable it, set the "SnmpSendAlertTrap" value to yes

6. Enable sending traps on occurrence of events

   This tells the FES to send an SNMP trap message to the configured manager whenever an event is received. The default is disabled. To enable it, set the "SnmpSendEventTrap" value to yes.

7. The system contact

   This variable should be set to the name of the individual responsible for maintaining the managed network. The default is support@fiorano.com. In order to set it, set the SNMPSystenContact value

8. The system location

   This is the physical location of the system. The default value is set to Unknown.

   Example: SnmpSystemLocation: Fourth Floor, Room No. 123

9. Trap Level

   This is used to set the Trap Levels for sending the traps to the SNMP Manager.

   Quite : 0 (00000)
   FioranoPeerServer : 1 (00001)
   Gatewayserver : 2 (00010)
   Application : 4 (00100)
   Service : 8 (01000)
   SBW : 16 (10000)

   Set the trap level to 0 for No traps and to 31 for all types of traps to be sent to the SNMP manager.

   For any combination of these traps, a value which sets the bits at those positions has to be provided.
   Example: For receiving both Application and Service level traps, the trap level will be 12 (01100)

Next, we have described how to configure iReasoning MIB tool to view and receive SNMP traps.

2.6.7.7 Using iReasoning MIB tool

Here we will show how to configure iReasoning MIB tool to view and receive SNMP traps and events.

Download

Download the iReasoning MIB browser from the link given below. Click on the link below to go to the webpage and click on "Accept".

http://ireasoning.com/downloadmibbrowserlicense.shtml

There are two installers provided. One for each Linux and Windows. After downloading the installer, for Windows, double-click on the .exe to install the tool and run it. For Linux, simply extract the .zip file.

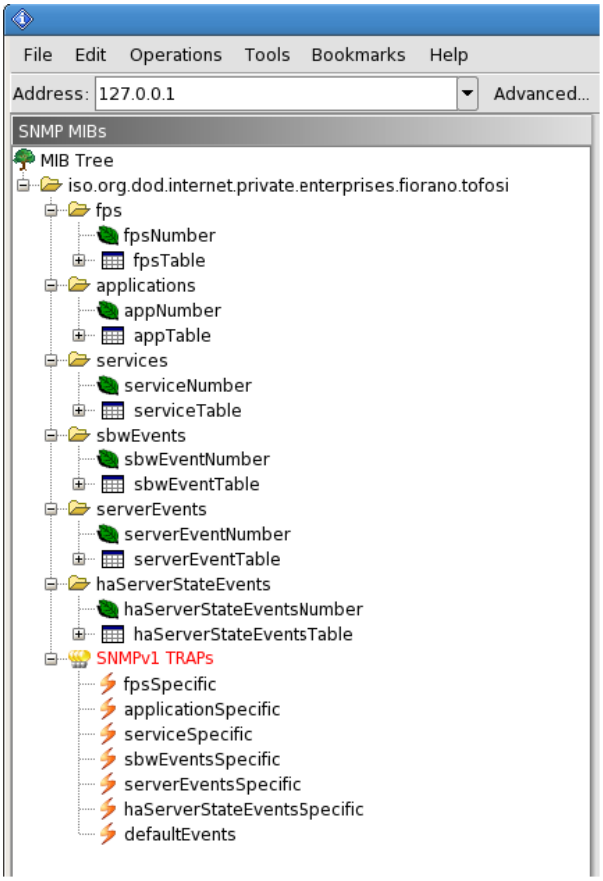
Running

- **Windows**: Right-click on the executable and Run-as-administrator.
- **Linux**: Open a terminal in the folder where you have extracted the zip file, or navigate to the folder via terminal, and enter the following command (after the dollar prompt) to run the tool. Enter the password when asked.

$ sudo ./browser.sh

Loading fiorano_mib.txt

From the menu bar, click on "File" and the "Load MIBs", A File chooser will open. Navigate to $FIORANO_HOME\esb\server\profiles\<profile_name>\FES\conf and load the file fiorano_mib.txt. You can see the following structure loaded

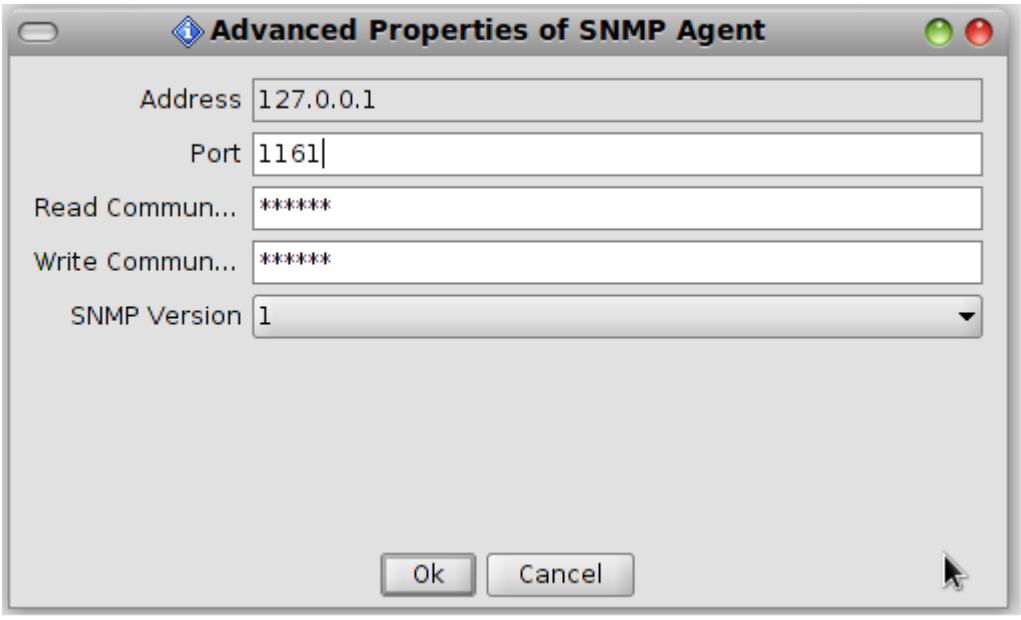


Configuring

Just below the menu bar is the address bar, where you can put in your custom configuration. In the address bar, click on **Advanced** button, the **Advanced Properties of SNMP Agent** windows will open. Here you can enter your configuration. Enter the configuration corresponding to the one configured in FES profile in Studio/eStudio.

Enter the following:

- **Port**: 1161 [whatever port entered in SNMP Agent Port in Studio]
- **Read Community**: public [whatever entered in Setting the SNMP community in Studio]
- **Write Community**: public [same as above]
- **SNMP Version**: 1



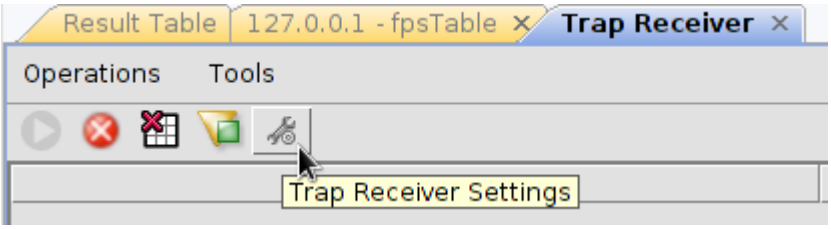
After configuring click **OK** to save the configuration.

Checking the events

Right click on any one of the tables listed in the tree view on the left. For example, navigate to fps-> fps Table and right-click on it and click on Table View to view the events related to fps.

Configuring Traps

From the menu bar, click on "Tools" ->"Trap Receiver" to open the trap receiver windows. In linux users may not be able to open the trap receiver window if the MIB browser is not started in super-user mode. Users can view the traps received in the trap receiver window.

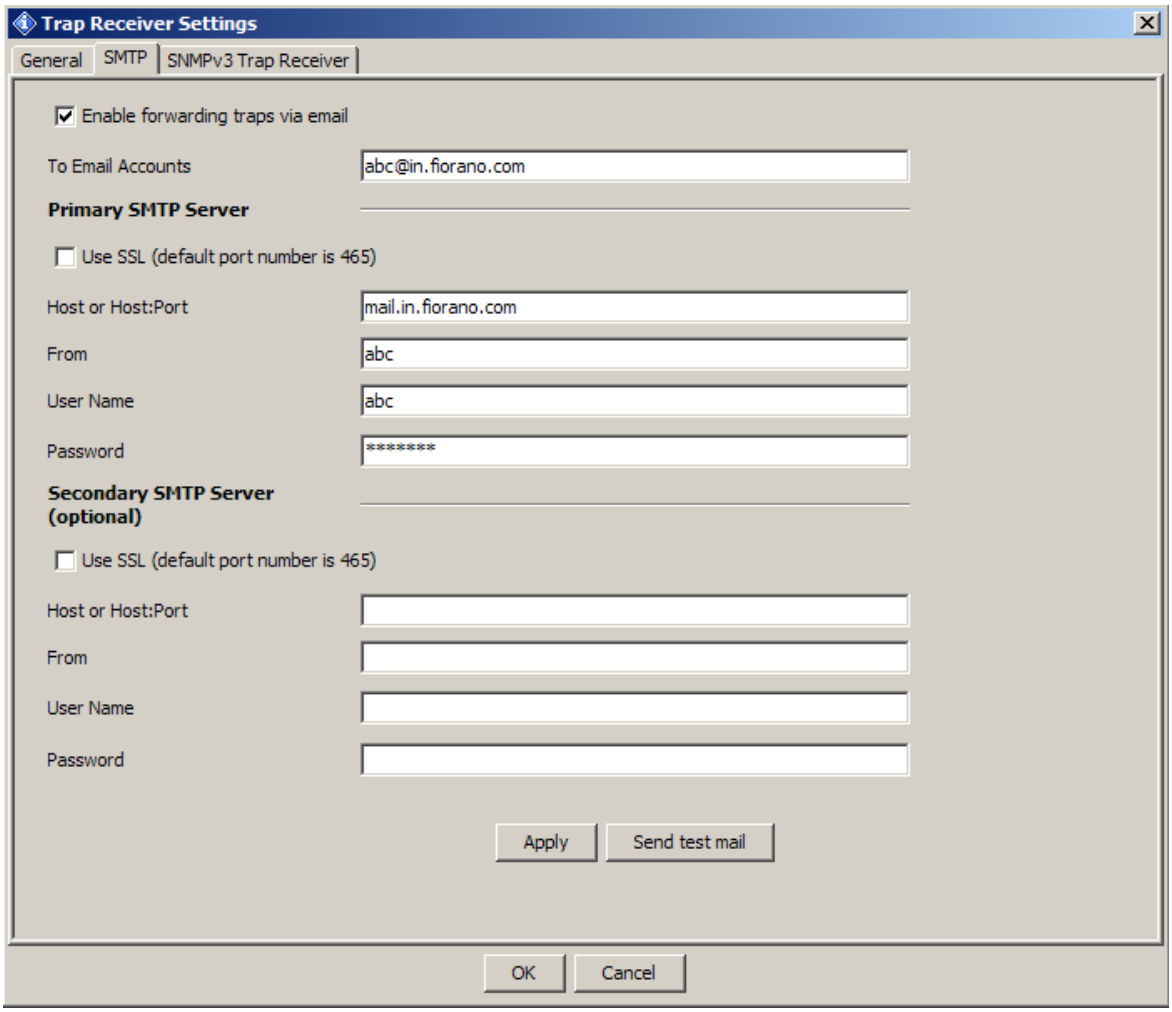


Configuring Mail for Traps

In the Trap Receiver windows, go to "Trap Receiver Settings" to configure Mail settings if you want to be notified via mail when you receive Traps.

1. Select the checkbox **Enable forwarding traps via email**.
2. In the **To Email Accounts** field, enter your email address.
3. Select the checkbox **Use SSL** to enable SSL.
4. Enter your primary SMTP Server configuration parameters
5. Enter your SMTP Hostname, for example mail.in.fiorano.com

6. Enter **From**, **Username**, and **Password** fields.
7. You can also configure a secondary SMTP server.
8. After configuring, you can test your configuration by clicking on **Send test mail** button.
9. Click on **Apply** and then **OK** button when done. You are now ready to receive mail when a trap is received.



2.6.8 Setting the Offline Server Log Levels

The path of the log files and the log levels of the server are set in the profile of the server. These properties can be located in the File named `Configs.xml` located under the `conf` directory of each profile, that is: `$Fiorano_Home/esb/server/profiles/<profileName>/<Server mode>/conf` where server mode can be either FES or FPS.

Examples:

- $Fiorano_Home/esb/server/profiles/profile1/FES/conf/Configs.xml
- $Fiorano_Home/esb/server/profiles/haprofile1/primary/FES/conf/Configs.xml

- $Fiorano_Home/esb/server/profiles/profile2/FPS/conf/Configs.xml

The various log levels of the Server are:

- Fatal – 1
- Error – 2
- Warning -3
- Info -4
- Debug -5
- Trace -6

The following figure illustrates a section in `Configs.xml` containing the `Logger` node.

```
<LOGGER LogLevel="4" ObjectName="Fiorano.Loggers:ServiceType=Logger,Name=Fiorano.Esb,type=config">
    <Appender AppenderName="OutAppender" AppenderType="file"
              FileName="esbout.log" IsAppend="true"
              LogPattern="[%d{dd/MMM/yyyy HH:mm:ss}] %-5p [%c{1}] %m%n"
              MaxBackupIndex="4" MaxFileSize="1000000" MaxFilterLevel="10"
              MinFilterLevel="4" PrintTarget="System.out" ThresholdLevel="10"/>
    <Appender AppenderName="Syslog4jEsbOutAppender" AppenderType="syslog" LoggingEnabled="false"
              Hostname="localhost" Protocol="udp" Port="514" Facility="local0"
              LogPattern="[%d{dd/MMM/yyyy HH:mm:ss}] %-5p [%c{1}] %m%n"
              MaxFilterLevel="10" MinFilterLevel="4" ThresholdLevel="10"/>
    <Appender AppenderName="ErrAppender" AppenderType="file"
              FileName="esberr.log" IsAppend="true"
              LogPattern="[%d{dd/MMM/yyyy HH:mm:ss}] %-5p [%c{1}] %m%n"
              MaxBackupIndex="4" MaxFileSize="1000000" MaxFilterLevel="3"
              MinFilterLevel="1" PrintTarget="System.err" ThresholdLevel="10"/>
    <Appender AppenderName="Syslog4jEsbErrAppender" AppenderType="syslog" LoggingEnabled="false"
              Hostname="localhost" Protocol="udp" Port="514" Facility="local1"
              LogPattern="[%d{dd/MMM/yyyy HH:mm:ss}] %-5p [%c{1}] %m%n"
              MaxFilterLevel="3" MinFilterLevel="1" ThresholdLevel="10"/>
</LOGGER>

<LOGGER LogLevel="3" ObjectName="Fiorano.Loggers:ServiceType=Logger,Name=Fiorano,type=config">
    <Appender AppenderName="OutAppender" AppenderType="file"
              FileName="mqout.log" IsAppend="true"
              LogPattern="[%d{dd/MMM/yyyy HH:mm:ss}] %-5p [%c{1}] %m%n" MaxBackupIndex="4"
              MaxFileSize="1000000" MaxFilterLevel="10" MinFilterLevel="4"
              PrintTarget="System.out" ThresholdLevel="10"/>
    <Appender AppenderName="Syslog4jMqOutAppender" AppenderType="syslog" LoggingEnabled="false"
              Hostname="localhost" Protocol="udp" Port="514" Facility="local2"
              LogPattern="[%d{dd/MMM/yyyy HH:mm:ss}] %-5p [%c{1}] %m%n"
              MaxFilterLevel="10" MinFilterLevel="4" ThresholdLevel="10"/>
    <Appender AppenderName="ErrAppender" AppenderType="file"
              FileName="mqerr.log" IsAppend="true"
              LogPattern="[%d{dd/MMM/yyyy HH:mm:ss}] %-5p [%c{1}] %m%n" MaxBackupIndex="4"
              MaxFileSize="1000000" MaxFilterLevel="3" MinFilterLevel="1"
              PrintTarget="System.err" ThresholdLevel="10"/>
    <Appender AppenderName="Syslog4jMqErrAppender" AppenderType="syslog" LoggingEnabled="false"
              Hostname="localhost" Protocol="udp" Port="514" Facility="local3"
              LogPattern="[%d{dd/MMM/yyyy HH:mm:ss}] %-5p [%c{1}] %m%n"
              MaxFilterLevel="3" MinFilterLevel="1" ThresholdLevel="10"/>
</LOGGER>
```

There are two Logger Nodes present in the configs.xml. Each Logger has two appenders one Out Appender and one Error Appender(along with the corresponding appenders for syslog). These appenders are file based Appenders.

The log level can be changed by changing the LogLevel attribute. The default log level of the logger having the objectName - Fiorano.Esb is 4 and the default log level of the logger having the objectName - Fiorano is 3. The default names of the log files to which appenders logs data to are: esbout.log, esberr.log, mqout.log, and mqerr.log.

The default location of these files is $Fiorano_Home/runtimedata/<SoaServerType>/<profileName>/run/logs where the SOA Server Type can either be Enterprise Servers or Peer Servers.

The syslog4j appenders can be enabled by setting the LoggingEnabled property to true and configuring the attributes Hostname, Protocol, Port, Facility etc. Note that the syslog daemon on the receiving box needs to be appropriately configured to receive logs over the network using the syslog4j appenders.

Note: If the Server is running in a shared HA profile, the `logs` directory gets created in the path of the directory specified as the `-dbPath` parameter while running the script `server` present in `$Fiorano_home/esb/server/bin`

Example:

Server.bat/sh –profile haprofile_shared/primary –dbPath /home/Fiorano/db

Here the logs directory gets created in the location `/home/Fiorano/db`

If the absolute path is not specified, the log files are created in the location `/home/Fiorano/db`. Providing an absolute path to the `FileName` attribute such as `/home/Fiorano/esbout.log`, results in log being created within the path specified.

The logger having the name `Fiorano.Esb` is used by the Fiorano SOA server and the logger having the name '`Fiorano`' is used by the embedded FioranoMQ present in a Fiorano SOA server, that is, if you are running the Fiorano Enterprise Server all logs of the Enterprise server are diverted to the files `esbout.log` and `esberr.log` and MQ related data is logged to `mqout.log` and `mqerr.log`.

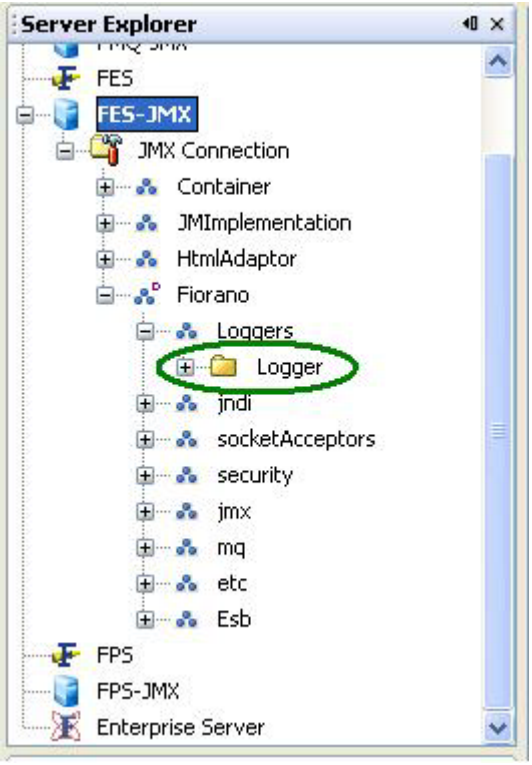
Note: Fiorano SOA server refers to the Fiorano Enterprise Server or the Fiorano Peer Server.

Note: These changes have to be made offline.

2.6.9 Changing the Running Server Log Levels

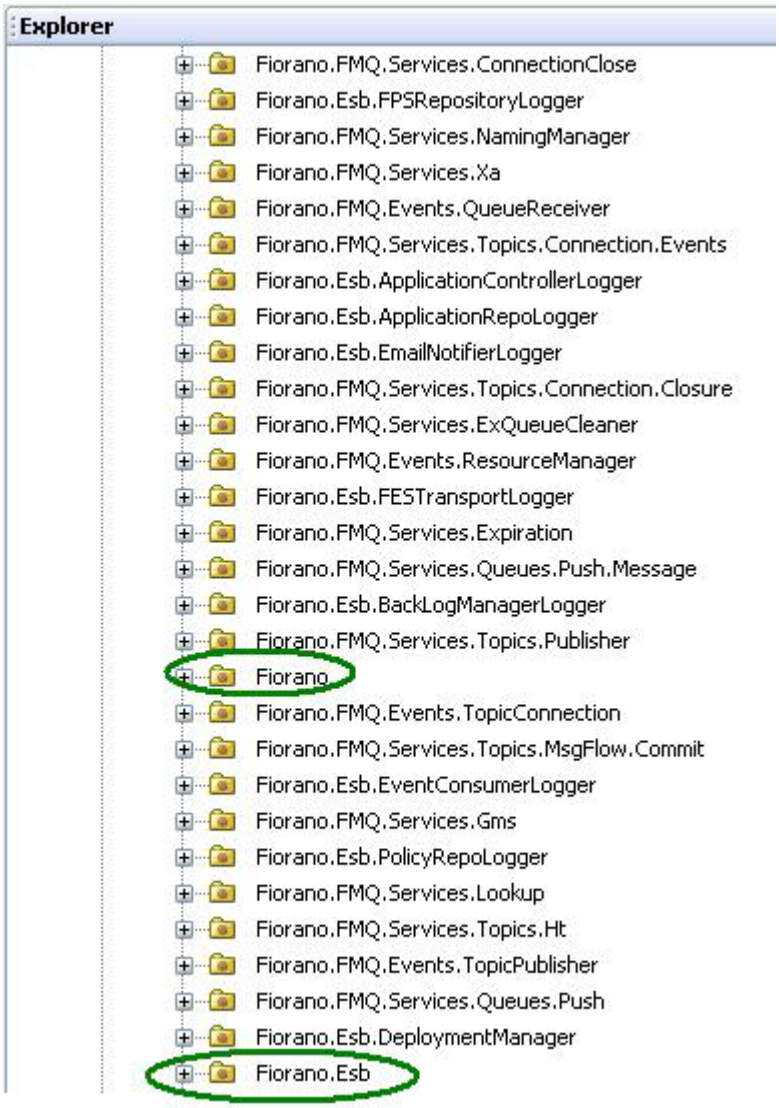
Changing the running server log levels can be done by logging into the server via JMX. Log into JMX from Fiorano Studio and Navigate to JMX Connection -> Fiorano->Loggers->Logger.

The following figure illustrates the JMX Tree of the Fiorano Enterprise Server once a user has logged into JMX.



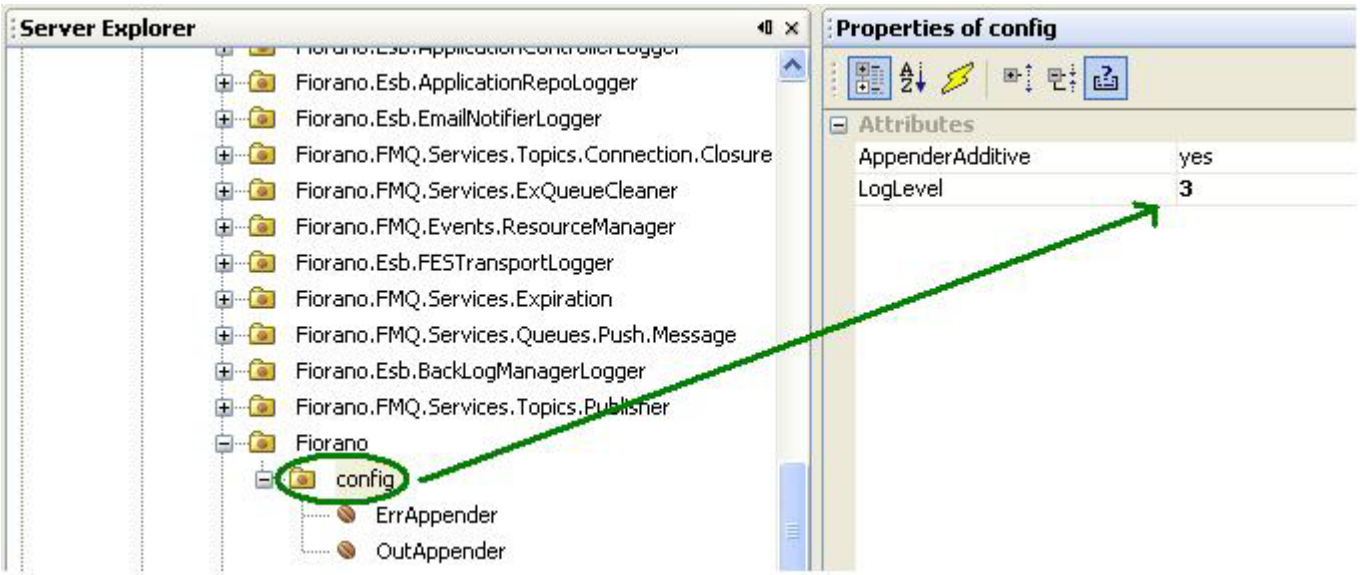
Expand the Logger node which shows multiple logger nodes.

The following figure illustrates the expanded view of the Loggers under the Logger Node shown above.



Select the `Fiorano` Logger, expand it and click on the `config` Node. Change the Logger log level in the properties window.

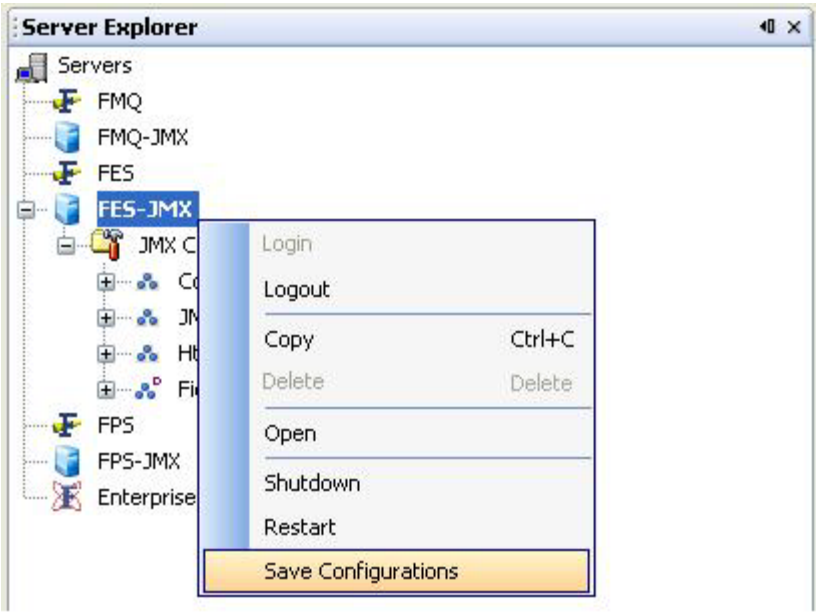
The following figure illustrates the `Fiorano` Logger `config` node and its properties in the Properties of config pane.



This changes the log level for the Fiorano Logger. Perform the same steps for the `Fiorano.Esb` Logger too.

Once this is done, right-click on the Server-JMX node and click on **Save Configurations** option from the pop-up menu. This saves the changed configuration for the logger.

The following figure illustrates the **Save Configurations** option for FES-JMX.



2.7 Fiorano Enterprise Repository

The Fiorano Enterprise Repository acts as an Enterprise-wide registry and repository for a Fiorano SOA Deployment. The repository is maintained by default in a File based data store. The FER (Fiorano Enterprise Repository) resides within the Fiorano Enterprise Server and manages the following:

- Event Process
- Pre-built Components and user defined components.
- Peer Configurations
- Principals and Access Control Lists
- Runtime State
- State Based Workflow Documents
- Events
- Alerts
- Policies

2.7.1 Event Process Repository

The Event Process Repository manages all the pre-built Event Processes and composed Event Processes. This is a layered repository over a System Store and User Store. The System Store stores the default set of Event Processes which are shipped along with the product and the User Store stores all the Event Processes which are subsequently composed by users. The location of the System and User Event Process repository are:

- **System Repository**: %FIORANO_INSTALL_DIR%\esb\server\repository\applications
- **User Repository**: %FIORANO_HOME%\runtimedata\repository\applications

The system repository is read-only and the entire user defined Event Processes are stored in the user repository. If you wish to move user-created Event Processes from one installation to another compatible installation, the application folder can be copied from one location to another. If format-compatibility does not exist between the versions, then the User has to export all the User Defined Applications in order to import them into the new installation. The Application Repository can be copied and backed up at anytime.

The Structure of the File Based Store for an Event Process is explained briefly below. An Event Process is arranged into Files and Directories.

- **EventProcess.xml** – Main index file which stores the details of the components. This file is linked to other files that contain configurations and schemas used for communication as well as for the layout of the workflow diagram.
- **Config** – Stores all the component configurations used in the Event Process.
- **Schemas** – Contains the schemas referred by the components used to specify the format of the expected input / output messages.
- **Env** – Deployment and environment properties.

An Event Process must be treated as a single unit/entity and has to be copied as such. Copying partial data from folder will corrupt the Event Process data.

2.7.2 Component Repository

Components or Services in Fiorano SOA are business functions, each with a set of configurations, which execute a single unit of work. Components can be developed and registered with the Fiorano Component Repository. A Registered component can be re-used any number of times in an Orchestrated event process. Supports for versioning of components are built into the repository.

The Component repository manages all pre-built components and the custom-created components. This repository is a layered over the system repository and the user repository. The System repository is a read-only repository that stores all the prebuilt components, while the user repository stores all user-defined custom components. The locations of the two repositories are:

- **System Repository**: %FIORANO_INSTALL_DIR%\esb\server\repository\components
- **User Repository**: %FIORANO_HOME%\runtimedata\repository\components

A Component repository can be backed up at runtime and can be version controlled. This repository can be retrieved and replaced at any time, but the changes will not be affected until:

1. The application which uses any of the changed/replaced components is stopped.
2. The *componentcache* property found in the properties panel of the event process is disabled and
3. A Connectivity and Resource Check (CRC) is done for the Event Process.

Each peer server has its own local cached copy of the component and the above steps are necessary for the Peer Server to update the component from the Enterprise Repository.

Migrating the component repository between compatible Fiorano versions involves copying the **components** directory to the corresponding location in the new installation. The System repository should not be copied as it is read-only and there can be a newer version of prebuilt components in the new installation. Please refer to the SOA installer Release Notes for the component compatibility matrix between different versions. For migrating custom components between non-compatible versions, each custom component has to be explicitly exported and imported into the new installation.

The Component repository is organized in files and libraries. Every component is represented by a directory with its GUID as the directory name and each directory has a sub-directory within it which stores the version of the component. Every component has the libraries required for that particular component, together with a Service Descriptor. A Service Descriptor describes the structure of the component and its input ports and output ports.

2.7.3 Peer Repository

The Fiorano Enterprise repository also manages the configurations of all Peer Servers on the Fiorano network. Each Enterprise Server has its own Peer repository where it tracks the configurations of all Peers connected to it. When a Peer Server connects with the Fiorano Enterprise Server, the configurations are referred to in the Peer repository of the Enterprise Server to which the Peer is connected and the Peer fetches the configuration from the corresponding repository. If the Peer registers with the Enterprise server for the first time, the Peer configuration is pushed into the Peer Repository of that particular Enterprise Server.

Peer HA Profiles have a special property which inhibits this behavior of fetching the configuration from the Enterprise repository. Since two different HA Profiles connect with the same name, the configurations cannot be fetched and loaded in this case. This special property is internal to Fiorano and is not exposed to the user. When one of the default HA Profiles is copied to create a new profile, this special property is carry forwarded. This property cannot and should not be set in a standalone profile.

The Peer repository is a transient repository built dynamically as the peers get registered, and hence any migration effort should not involve copying this repository. Additionally, this repository should not be backed up and restored. It is recommended that whenever a Peer profile is changed, the profile should be cleared form this repository for the change to take effect otherwise the configuration present in the Peer repository gets pushed to the Peer Server and the Peer Server loses the changes made to its local profile.

The peer repository can be located at:
%FIORANO_HOME%\runtimedata\repository\EnterpriseServers\{FES_profile_name}\FES\Peers\.

Each peer profile is a directory with the name of the Peer and with a set of definition and configuration files inside its folder.

2.7.4 Security Store

The Fiorano Enterprise repository manages the Users, Groups and Access Control lists for each principal. This is a File Based Store and the format is internal to Fiorano and it is not visible to the end user. The Security Store can be found at:
%FIORANO_HOME%\runtimedata\EnterpriseServers\{FES_profile_name}\FES\run\SDB.
Fiorano has a centralized user and permissions group store, any changes to this user and permission group store are propagated to all connected Peer Servers, whose local security stores are then updated. This store contains information about the users, groups and the access control list for each permission.

The security store can be backed up at runtime; however, restoring the store requires the Enterprise to be stopped. The security store can be migrated between compatible versions of Fiorano. For non-compatible versions of Fiorano; all of the users, groups and access control lists need to be recreated.

2.7.5 Runtime Store

The Fiorano Enterprise Server stores the current state of an Event Process and other Metadata such as the system level destinations created within the admin runtime store. When an Event Process is started, the state of the Event Process is stored in this admin store and is restored whenever the Fiorano Enterprise Server is started. State of the Event Process can either be RUNNING or NOT_RUNNING and this State needs to be persisted in order to restore the state of Event Processes on Server restarts. When servers are restarted, the Enterprise Server refers this repository and automatically starts those Event Processes that were running the last time the servers were able to persist in the state of the Event Process. There is no guarantee that the state will be the very same state as when the servers were brought down or crashed since it may have persisted in the state in which it may have crashed. The Fiorano Enterprise Server automatically caches the state of the Event Process when there is a change to it.

The term **state** here refers to key information relating to the deployment of an Event Process and not the actual runtime data flowing between components participating the Event Process. The local naming and directory module binds the objects to a name in this runtime store. This is a single file located at:

%FIORANO_HOME%\runtimedata\EnterpriseServers\{FES_profile_name}\FES\run\admin.dat

If the state of all Event Processes needs to be cleared, then the `admin.dat` can be deleted. This is a transient data structure which should not be backed up and restored. This data is stored in an internal format and is not visible to the end user.

If the Enterprise Server is running in High Availability mode, its state together with that of it's backup/secondary server, is maintained in:

%FIORANO_HOME%\runtimedata\ EnterpriseServers\{HA_PROFILE_NAME}\{PROFILE_TYPE}\FES\run\haStatus.dat

This is a transient data structure used to maintain the current state of an HA server as well as it's previous state. The file format is internal and is not readable; as such, this data file should not be backed up or migrated to a different installation. This data store is common to all servers running in High Availability mode. All Peer Servers running in HA mode have a similar data store with identical semantics.

The runtime data store that stores the pending messages and the destinations which are responsible for the actual Peer-to-Peer message traffic between different components are present in each Peer Server in the following location:

%FIORANO_HOME%\runtimedata\PeerServers\{PROFILE_NAME}\FPS\run

This transient message store can be backed up and migrated if there are no changes in the Event Processes or the components. The pending messages will be processed the next time the servers are up after migration.

2.7.6 State Based Workflow Repository

The State Based Workflow (SWB) or Document tracking creates Document events that are raised in the Peer Servers and forwarded to the Enterprise Server. The Enterprise Server is responsible for storing the Documents for auditing and tracking purposes. For more details, refer to section 5.3.8.1 Configuring Document Tracking.

SBW documents are pushed into a temporary persistent cache from the Peer Servers, from where they are picked up by the Enterprise Server and inserted into the configured SBW Database. By default, this database is configured as the internal H2 database and stores the record in its proprietary format in the file system.

Persistent cache:

Persistent cache is a transient data store into which tracked documents are written temporarily before they are moved to the configured external database for storage. This cache can be found at:

%FIORANO_HOME%\runtimedata\EnterpriseServers\{FES_Profile_Name}\FES\run\db\PersistantDB\PQ.Tifosi_Enterprise_Server_SBW_Events

This cache should not be cleared manually; the size of this cache can build up when the configured Document tracking store is not available. As such, the SBW configured database cannot be down for an extended period of time. If it is down for an extended period of time, the document tracking configurations have to be changed to point to a different external database or else the cleardb script will be used to clear the persistent cache of the SBW Documents. The Persistent cache is a transient data store and the data is stored in an internal record format which is not visible to the end user and should not be migrated.

Document Tracking H2 Database
%FIORANO_HOME%\runtimedata\EnterpriseServers\{FES_Profile_Name}\FES\doctracking_db

This is the default SBW data store shipped with the Fiorano SOA Platform. The database is an H2 data store, which should not typically be used in a production setting. In Production, an external database should be configured to store and retrieve the tracked documents and periodical cleaning of the tracked documents should be scheduled and executed with the help of Fiorano scripts. The database can be archived for auditing purposes and can be backed up regularly.

For more information on the structure of SBW database and for instructions on how to configure a different external database to be used for SBW tracking, please refer to section 5.3.8 Document Tracking.

2.7.6.1 Disabling SBW Tracking

To disable SBW Tracking globally, perform the following steps:

1. Login to FES-JMX and locate Fiorano->Esb->Sbw->SBWManager->SBWManager->config.
2. Set the property named 'EnableDocTracking' to 'no'.

2.7.7 Event Repository

The Fiorano SOA platform raises Events for all meaningful operations that occur in the system. These System Events are collected on the Enterprise Server and the state of each artifact (Event processes, components, server status etc) is determined by the sequence of such events. All captured System Events are inserted into a configured database. If the component is configured to raise the monitoring events on the various performance metrics, then these events are also stored in the Event repository. The default database for the Event repository is the H2 database shipped by default along with the Fiorano SOA Platform.

2.7.8 Custom Event Listener

Custom applications can be written to listen to the System Events and SBW Events. For more information, see section 2.8 Event Tracking.

Refer to RTL Java documentation of *com.fiorano.esb.fes.rtl.events.FioranoEventsManager* for more details on registering with the Enterprise Server for listening to System and User events.

2.7.9 Alert Repository

The Fiorano Enterprise Repository manages all Alert configurations. The Alert configurations are looked up when a notification is being executed. This is a File Based Repository located at:

%FIORANO_HOME%\runtimedata\repository\Notifications

Each alert is represented by a single XML file listed in the above directory. The name of the file is the unique identifier of the Alert. The structure of the document varies depending upon the transport over which the notification is configured. Fiorano does not recommend changing values directly in the XML or any interaction with the OS in a way that affects the file system. In particular, an "Antivirus" or any other program that locks files or hinders the creation and deletion of files should not be run over this repository (or any of the Fiorano repositories).

The Alert repository can be backed up and versioned in any source control and restored at anytime during the Fiorano Server Execution. The Updated notification from the Repository will be picked up automatically at the next execution of the notification. This repository can also be migrated from one version to another through copying the file system provided the repositories are claimed interoperable in the release notes of newer versions of Fiorano. Please check the release notes for the interoperability matrix on this repository.

The Alerts are queued up as jobs which are then picked up by the Job Manager and executed serially. There is no guarantee that the alert will be executed within a specific period of time. If the notification job crashes in the middle of the execution the data will be lost and will not be re-executed when the server starts up. However, the pending list of notifications will continue to be executed.

2.7.10 Policy Repository

Fiorano governance solutions use a Policy-Based approach. Policies can be created for a condition and on achieving the condition a notification over a transport is executed. The Fiorano Enterprise Server manages the policies in a default File-Based repository. This Policy Repository is internally dependent on the Alert Repository for fetching the notification information which it then executes.

%FIORANO_HOME%\runtimedata\repository\Policies

Backlog policies are one type of policy that register for a particular count of the pending requests at an execution port of a service and execute notifications based on the count. The backlog policies are stored in the following location:

%FIORANO_HOME%\runtimedata\repository\Policies\Backlog

For each type of policy, a directory is created inside the Policy repository. Each backlog policy is stored in an XML file whose name matches the name of the policy ID. Each type of policy has a specific XML structure that includes the condition details and the alert lists to be executed when the condition is met.

The Policy repository can be backed up and versioned in any source control and can be restored at any time when the Fiorano server is not running. The Updated policies from the Repository will be picked up automatically from the web console upon a refresh. This repository can also be migrated from one version to another through copying the file system provided the policy structures are claimed interoperable in the release notes of the newer version of Fiorano. Please check the release notes for the interoperability matrix on this repository.

A Policy can be either in an ACTIVE or an INACTIVE state. The current state of the policy persists along with the policy. Once a policy is applied, a restart of the servers automatically re-applies the policy and the Peer Servers get notified about these policies. Backing up and restoring the policy repository will also restore the state of the policy at the time it was backed up.

2.7.11 Changes in Repository Location

A Layered repository approach is implemented after version SOA 2007 SP1. This has brought the following changes in the location of the repository:

Data	Location (Till SOA 2007 SP1)	Old Script (From SOA 2007 SP2 to SP4)	Old Script (From SOA 2007 SP5 and further release)
ReadOnly Repository	fiorano.home/esb/fes/repository	fiorano.home/esb/fes/repository	fiorano.home/esb/server/repository
Run Repository	NA	user.dir/repository	user.dir/repository
Default Repository (HA Secondary)	fiorano.home/esb/fes/repository_backup	fiorano.home/esb/fes/repository	fiorano.home/esb/server/repository
Runtime Repository	NA	user.dir/repository_backup	user.dir/repository_backup

(HA Secondary)			
Peer Repository	fiorano.home/esb/fes/repository/peers	user.dir/EnterpriseServers/<profilename>/peers	user.dir/EnterpriseServers/<profilename>/FES/peers
run folder(FES)	fiorano.home/esb/fes/repository/run	user.dir/EnterpriseServers/<profilename>/run	user.dir/EnterpriseServers/<profilename>/FES/run
run folder(FPS)	fiorano.home/esb/fps/repository/run	user.dir/PeerServers/<profilename>/run	user.dir/PeerServers/<profilename>/FPS/run
runtimecomponents.dat (FES)	fiorano.home/esb/fes/<profilename>/conf/runtimecomponents.dat	user.dir/EnterpriseServers/<profilename>/run/ runtimecomponents.dat	user.dir/EnterpriseServers/<profilename>/FES/run/ runtimecomponents.dat
runtimecomponents.dat (FPS)	fiorano.home/esb/fps/<profilename>/conf/runtimecomponents.dat	user.dir/PeerServers/<profilename>/run/ runtimecomponents.dat	user.dir/EnterpriseServers/<profilename>/FPS/run/ runtimecomponents.dat
Events db	fiorano.home/esb/fes/bin/derbyDB	user.dir/EnterpriseServers/<profilename>/events_db	user.dir/EnterpriseServers/<profilename>/FES/events_db
Doctracking (SBW) db	fiorano.home/esb/fes/bin/derbyDB	user.dir/EnterpriseServers/<profilename>/doctracking_db	user.dir/EnterpriseServers/<profilename>/FES/ doctracking_db
Derby.log	fiorano.home/esb/fes/bin/derbyDB	user.dir/EnterpriseServers/<profilename>/run/logs/derby.log	user.dir/EnterpriseServers/<profilename>/FES//run/logs/derby.log
Container.log	Container.log	user.dir/EnterpriseServers/<profilename>/run/logs/Container.log	user.dir/EnterpriseServers/<profilename>/FES//run/logs/Container.log

2.8 Events Tracking

Fiorano SOA platform raises an event for any meaningful event that occurs in the system. These System events are collected in the Enterprise Server and are processed and the state of the artifacts (Event processes, components, server status, and so on) is determined by the sequence of such Events. If the component is configured to raise the monitoring events on the various performance metrics, then these events are also stored in the event repository. The default database is configured as H2 and the database is File-Based.

2.8.1 Configuring Event Tracking

The Enterprise Server is by default configured to insert all System events into the default H2 Events database (A different events database can be configured for Event Tracking, see next section). If at any point the user decides to switch off event tracking, they can do so by opening the profile and navigating to Fiorano->Esb->Events->FESEventsManager (as shown in figure 2.8.1) and changing the property named EnableSystemEventTracking to false. This value can be changed at runtime by logging into FES-JMX.

Another property named ListenForUserEvents in the same property panel decides whether the Enterprise Server listens to the monitoring events published by component instances. This property is by default set to false. If monitoring is enabled for component instances (see section 9.5 Creating Mappings), this property needs to be changed to true so that these user events are recorded.

Note: This property cannot be changed at runtime. Changing this property via the FES-JMX connection would require a server restart for the change to take effect. If this property is set to true, the property EnableSystemEventTracking will decide whether these events will be inserted into the Events database or not.

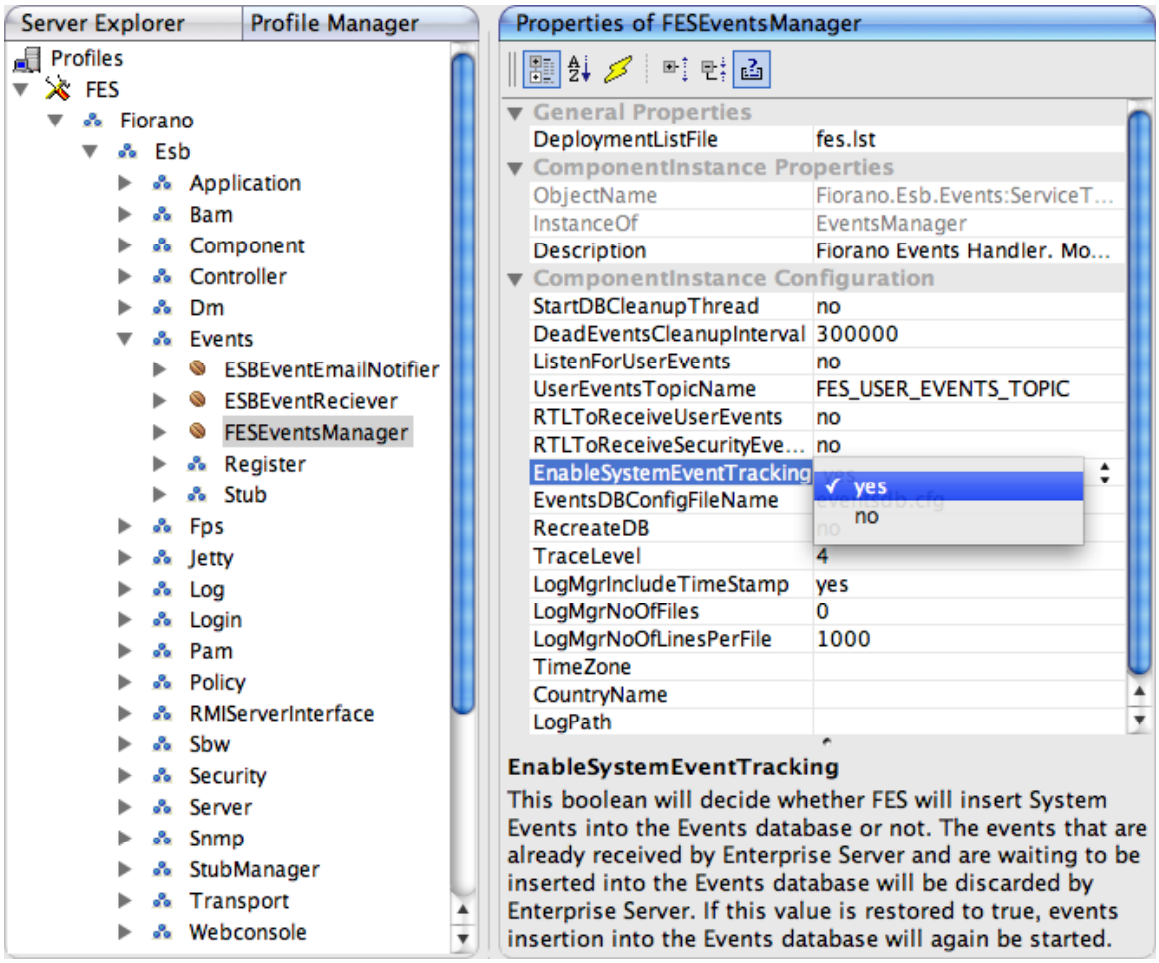


Figure 2.8.1: Configuring Event Tracking

2.8.1.1 Disabling Event Tracking

To disable Event Tracking globally, perform the following steps:

1. Login to FES-JMX and locate
2. Fiorano->Esb->Events->EventsManager->FESEventsManager->config. Set the property named 'EnableSystemEventTracking' to 'no'.

2.8.2 Configuring Specific Database

The event tracking feature is configured as part of FES to track System and User events into the H2 database running within the Enterprise Server. This can be changed by customizing the *eventsdb.cfg* file present in: FIORANO_INSTALL_DIR\esb\server\profiles\<profilename>\FES\conf. This file contains all the DB specific configurations used for event tracking. Depending on the type of database used, you might have to modify the *<dbtype>_jdbc.cfg* file, and this is set in the **JDBC_PROPERTIES property** of *eventsdb.cfg* file. The file *<dbtype>_jdbc.cfg* contains more database specific configurations. For example, an error code used by the database to indicate whether or not a table is present in the database or not. Additionally, this file also contains names of the data types that will be used by the database. These datatype names are mentioned against unique numbers. These numbers are the constants that are used to identify generic SQL types, called JDBC types (Please refer to javadoc for java.sql.Types).

The default configuration shipped with the installer uses the H2 database.

Note:

1. After configuring a profile to use a database other than the default database, jdbc driver for that database needs to be added under <java.classpath> tag in server startup configuration file (either $FIORANO_HOME/esb/server/bin/server.conf or $FIORANO_HOME/esb/fes/bin/fes.conf, whichever is applicable) prior to starting the Enterprise Server.
2. You would have to use the same settings to connect to the DB when using a third-party tool. All the database queries used for retrieving workflow related data is kept in *events.sql* file.
3. When using MS SQL for events tracking, mssql_jdbc.cfg may need to be configured according to the database driver being used. MSSql 2000 driver follows SQL 99 conventions which quote the SQLState string for table not found exception as **42S02**. On the other hand, MSSQL 2005 driver follows XOPEN SQLState conventions which quote the same SQLState string as **S0002**. By default, all fes profiles are configured according to the standards followed by the MSSql 2000 driver. If someone uses MSSql 2005 database, or uses the MSSql 2005 driver for MSSql 2000 (2005 driver is backward compatible with 2000 driver and can be used), then the file has to be re-configured accordingly.

Important: It is strongly recommended that the User employ a commercial grade DB in a production system.

For File-Based databases like apache and HSQL, the default location is in the ESB_USER_DIR (which is set in fiorano_vars script). The user has to give the complete path with these variables resolved when using the JDBC URL in a third-party tool.

Example

The default H2 db JDBC URL is configured as **ESB_DEFAULT_DB_DIR/events_db;create=true** which resolves to **ESB_USER_DIR/EnterpriseServers/<profilename>/FES/events_db** and further into something like C:\Program Files\Fiorano\<FioranoSOAProductVersion>\runtimedata\EnterpriseServers\<profileName>\FES\events_db depending on the actual settings.

2.8.3 Database Table Structure

The exact schema of the tables varies from database to database according to the configurations provided in *<dbtype>_jdbc.cfg file*. An explanation of the tables and the various fields is given below:

Table name: TPS_EVENTS & TES_EVENTS

Column Name	Type	Description
EVENT_ID	INTEGER	Auto Generated
EVENT_CATEGORY	VARCHAR(255)	Category of event i.e. Information, Warning or Error
GENERATION_DATE	TIMESTAMP	Time at which the event was generated
EVENT_SOURCE	VARCHAR(255)	Server name which generated the event
EVENT_SCOPE	VARCHAR(255)	Scope of the event
EVENT_MODULE	VARCHAR(255)	Module to which the event belongs
DESCRIPTION	VARCHAR(255)	Short description of the event
EVENT_STATUS	VARCHAR(255)	Event constant representing the type of event
EXPIRY_TIME	TIMESTAMP	Event expiry time

Table name: SERVICE_EVENTS

Column Name	Type	Description
EVENT_ID	INTEGER	Auto Generated
EVENT_CATEGORY	VARCHAR(255)	Category of event i.e. Information, Warning or Error
GENERATION_DATE	TIMESTAMP	Time at which the event was generated
EVENT_SOURCE	VARCHAR(255)	Server name which generated the event
EVENT_SCOPE	VARCHAR(255)	Scope of the event
EVENT_MODULE	VARCHAR(255)	Module to which the event belongs

DESCRIPTION	VARCHAR(255)	Short description of the event
EVENT_STATUS	VARCHAR(255)	Event constant representing the type of event
EXPIRY_TIME	TIMESTAMP	Event expiry time
SERVICE_GUID	VARCHAR(255)	The GUID of the service to which the event belongs
SERVICE_VERSION	VARCHAR(255)	The service version to which the event belongs
SERVICE_INST_NAME	VARCHAR(255)	Service Instance name
APP_GUID	VARCHAR(255)	Application Name

Table name: APPLICATION_EVENTS

Column Name	Type	Description
EVENT_ID	INTEGER	Auto Generated
EVENT_CATEGORY	VARCHAR(255)	Category of event i.e. Information, Warning or Error
GENERATION_DATE	TIMESTAMP	Time at which the event was generated
EVENT_SOURCE	VARCHAR(255)	Server name which generated the event
EVENT_SCOPE	VARCHAR(255)	Scope of the event
EVENT_MODULE	VARCHAR(255)	Module to which the event belongs
DESCRIPTION	VARCHAR(255)	Short description of the event
EVENT_STATUS	VARCHAR(255)	Event constant representing the type of event
EXPIRY_TIME	TIMESTAMP	Event expiry time
APP_GUID	VARCHAR(255)	The Event Process GUID
APP_NAME	VARCHAR(255)	The Event Process Name

Table name: SECURITY_EVENTS

Column Name	Type	Description

EVENT_ID	INTEGER	Auto Generated
EVENT_CATEGORY	VARCHAR(255)	Category of Event, that is, Information, Warning or Error
GENERATION_DATE	TIMESTAMP	Time at which the Event was generated
EVENT_SOURCE	VARCHAR(255)	Server name which generated the Event
EVENT_SCOPE	VARCHAR(255)	Scope of the Event
EVENT_MODULE	VARCHAR(255)	Module to which the Event belongs
DESCRIPTION	VARCHAR(255)	Short description of the Event
EVENT_STATUS	VARCHAR(255)	Event constant representing the type of event
EXPIRY_TIME	TIMESTAMP	Event expiry time
USER_NAME	VARCHAR(255)	Name of the user for which the Event was generated

Table name: USER_EVENTS

Column Name	Type	Description
EVENT_ID	INTEGER	Auto Generated
EVENT_CATEGORY	VARCHAR(255)	Category of event, that is, Information, Warning or Error
GENERATION_DATE	TIMESTAMP	Time at which the Event was generated
EVENT_SOURCE	VARCHAR(255)	Server name which generated the Event
EVENT_SCOPE	VARCHAR(255)	Scope of the Event
EVENT_MODULE	VARCHAR(255)	Module to which the Event belongs
DESCRIPTION	VARCHAR(255)	Short description of the Event
EVENT_STATUS	VARCHAR(255)	Event constant representing the type of Event
EXPIRY_TIME	TIMESTAMP	Event expiry time

SERVICE_GUID	VARCHAR(255)	The service instance GUID
TPS_NAME	VARCHAR(255)	The Peer Server name
TEXT_DATA	VARCHAR(255)	Event Text
BYTES_DATA	IMAGE	BLOB field representing user event data

2.9 Connecting to Server Using JConsole

To connect Fiorano server using JConsole, perform the following steps:

1. Open server.conf, fes.conf or fps.conf (whichever is applicable according to the script being used to run the server) and add the JVM argument "com.sun.management.jmxremote" under the java.system.props section.
2. Start the server, Start JConsole using JDK_HOME/bin/JConsole
3. To connect JConsole to server process, in the advanced tab, specify the url service:jmx:rmi:///jndi/rmi://localhost:2047/fmq and username/password using which a connection is desired, for example, admin/passwd.

2.9.1 How to connect to the FES/FPS using jconsole [running JDK6] SOA 9

To connect to the FES/FPS using jconsole SOA 9, perform the following steps:

1. Add the following values under <java.system.props> in the file $FIORANO_HOME/esb/server/bin/server.conf

   com.sun.management.jmxremote

   com.sun.management.jmxremote.port=<somePortNumber>

   com.sun.management.jmxremote.authenticate=false

   com.sun.management.jmxremote.ssl=false
2. If JDK version 6 is to be used to launch the server, remove/comment the line '../../../extlib/mx4j' in server.conf file. To comment a line, it should start with #. That is, the commented line will be '#../../../extlib/mx4j'.
3. Start the server, copy $FIORANO_HOME/framework/lib/fiorano-rmi-client.jar to $JDK_HOME/bin
4. Create a file called 'policy.pol' under $JDK_HOME/bin & paste the following content in it:

   ```
   grant {
       permission java.security.AllPermission "", "";
   };
   ```

5. Go to $JDK_HOME/bin type the following command:

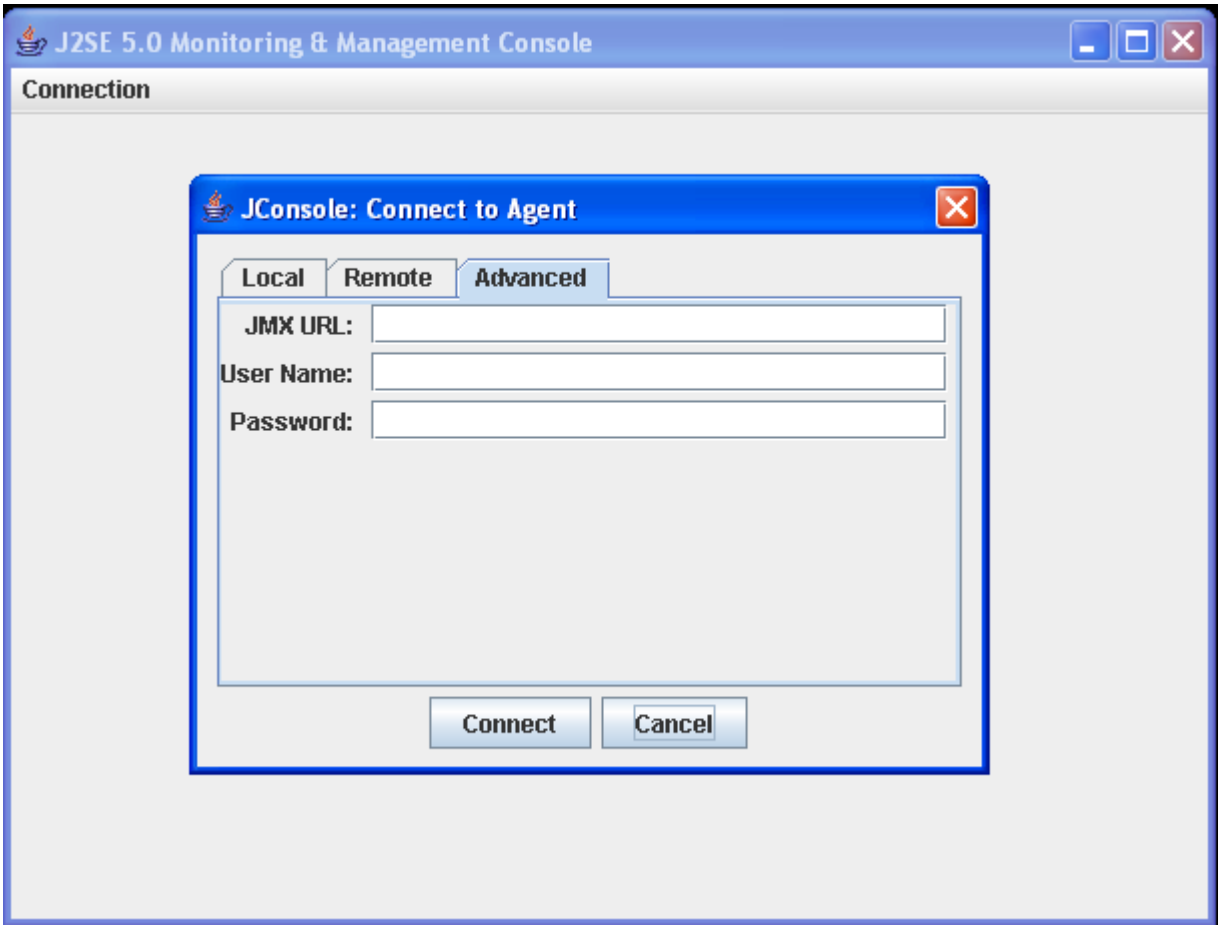
For Windows:

jconsole -J-Djava.security.policy=policy.pol -J-Djava.security.manager -J-Djava.rmi.server.codebase=file:///fiorano-rmi-client.jar -J-Djava.class.path=..\lib\tools.jar;..\lib\jconsole.jar;.\fiorano-rmi-client.jar

For Unix:

j jconsole -J-Djava.security.policy=policy.pol -J-Djava.security.manager-J-Djava.rmi.server.codebase=file:///fiorano-rmi-client.jar-J-Djava.class.path=../lib/tools.jar:../lib/jconsole.jar:./fiorano-rmi-client.jar

6. Once Jconsole starts, click on the advanced tab



Give the following values for the 'properties mentioned' in the above figure.

JMX URL

service:jmx:rmi://<IpAddressOfServer>/jndi/rmi://<IpAddressOfServer>:<rmiConnectorPortOfServer>/fmq

Note: RmiConnector port is printed on server console.

UserName

Some valid user name, say 'admin'

Password

Some valid password for a valid user, say 'passwd' for admin user

7. Press Connect and you should be able to profile the server.

2.9.2 Starting multiple servers in same machine

When the steps mentioned in previous section are followed, the user will not be able to launch more than one server on same machine using server.bat/sh files.

In order to be able to launch more than one server on same machine, create copies of server.conf and server.bat/sh files and rename them as server1.conf and server1.bat/sh (Note that these .conf and .bat/sh files can be renamed to any name, but both these files should have same name). Now, provide some other port number under server1.conf file for the property 'com.sun.management.jmxremote.port=<somePortNumber>'. After doing this, server1.bat/sh can be used to launch the second server. For example, the command 'server1.sh -profile profile2 -mode fps' can be used to launch the second server.

2.10 Running Fiorano SOA Profiles as NT Services

The Fiorano SOA has a concept of profile based server, for example, profile1, profile2, haprofile, and so on. All these profiles can be run either as a Fiorano Enterprise Server or a Fiorano Peer Server by specifying their mode as 'fes' or 'fps' respectively.

These profiles can be run as NT services on Windows NT, Windows 2000 and Windows XP, Windows Server 2003/2008 operating systems. Any profile present under %FIORANO_HOME%/esb/server/profiles can be run as an NT service.

2.10.1 Configuring Server JVM Settings for running as NT-Service

The 'server.conf' file located under %FIORANO_HOME%/esb/server/bin contains various settings related to the JVM like bootclasspath, classpath, server memory, system properties, and so on. This file is used while creating configuration files required to run the server as a nt-service.

The configuration files are generated under the following directories:

- For Enterprise Servers: %FIORANO_HOME%/runtimedata/EnterpriseServers/%PROFILE_NAME%/service/
- For Peer Servers: %FIORANO_HOME%/runtimedata/PeerServers/%PROFILE_NAME%/service/

Once the server is installed as a service, future changes to the file 'server.conf' will have no effect on the server as the configuration files will not be regenerated. The server profile needs to be uninstalled and then installed again for the changes to be effective. Make sure you have made the entire server JVM settings before installing the profile as a service.

On how to change the memory settings for the server, refer to **Question 1** in section 2.10.6 FAQs.

Important: Before installation of server profiles as services on Windows Server 2003/2008, the following line needs to be added to file %FIORANO_HOME%/esb/server/bin/server.conf 'java.io.tmpdir=C:/Windows/temp' under the '<java.system.props>' tag.

2.10.2 Configuring Email Alerts for Server JVM Restart/Sudden Exit/JVM Hung Events

The Wrapper binary which is used to run the server as a service can generate email alerts when the server either restarts, exits unexpectedly or becomes hung.

Please refer to the file 'server.conf' located under '%FIORANO_HOME%/launcher' for the configurations.

Once the server is installed as a service, future changes to the file 'server.conf' will have no effect on the server. The server profile needs to be uninstalled and then installed again for the changes to be effective. Make sure you have made all the settings before installing the profile as a service.

2.10.3 How to install/uninstall a StandAlone (Non-High Availabilty)Profile as NT Service

To install a StandAlone (Non-High Availabilty)Profile as NT Service, perform the following steps:

1. Open a command prompt. Go to %FIORANO_HOME%/esb/server/bin/service directory
2. To install FES profile as an NT service, run the command install-server.service.bat -mode fes -profile %PROFILE_NAME%

   Example: install-server.service.bat -mode fes -profile profile1
3. To install FPS profile as an NT service, run the command install-server.service.bat -mode fps -profile %PROFILE_NAME%

   Example: install-server.service.bat -mode fps -profile profile2

   Note: If mode is not specified, it is taken as 'fps'.
4. After installation you should be able to see Fiorano ESB Server <%PROFILE_NAME%> NT service or Fiorano Peer Server <%PROFILE_NAME%> NT service installed.

To uninstall a StandAlone (Non-High Availability) Profile as NT Service, perform the following steps:

1. To remove FES NT service, run the command uninstall-server-service.bat -mode fes -profile %PROFILE_NAME%

Example: uninstall-server.service.bat -mode fps -profile profile1

2. To remove FPS NT service, run the command uninstall-server-service.bat -mode fps -profile %PROFILE_NAME%

   Example: uninstall-server.service.bat -mode fps -profile profile2

   Note: If mode is not specified, it is taken as 'fps'.

3. Un-Install should stop and remove the NT service for the given profile from the services list.

2.10.4 How to install/uninstall a Profile as NT Service with High Availability Profile(HA)

While running the servers in HA mode, the servers need to access network shares.

To access the network share, the server needs to configure with the user credentials of a User Account of the system.

1. Specify the credentials in the file %FIORANO_HOME%/launcher/server.conf

   Uncomment the following lines in the file and specify the user account name and password.

   #wrapper.ntservice.account=.\userName

   #wrapper.ntservice.password=passwd

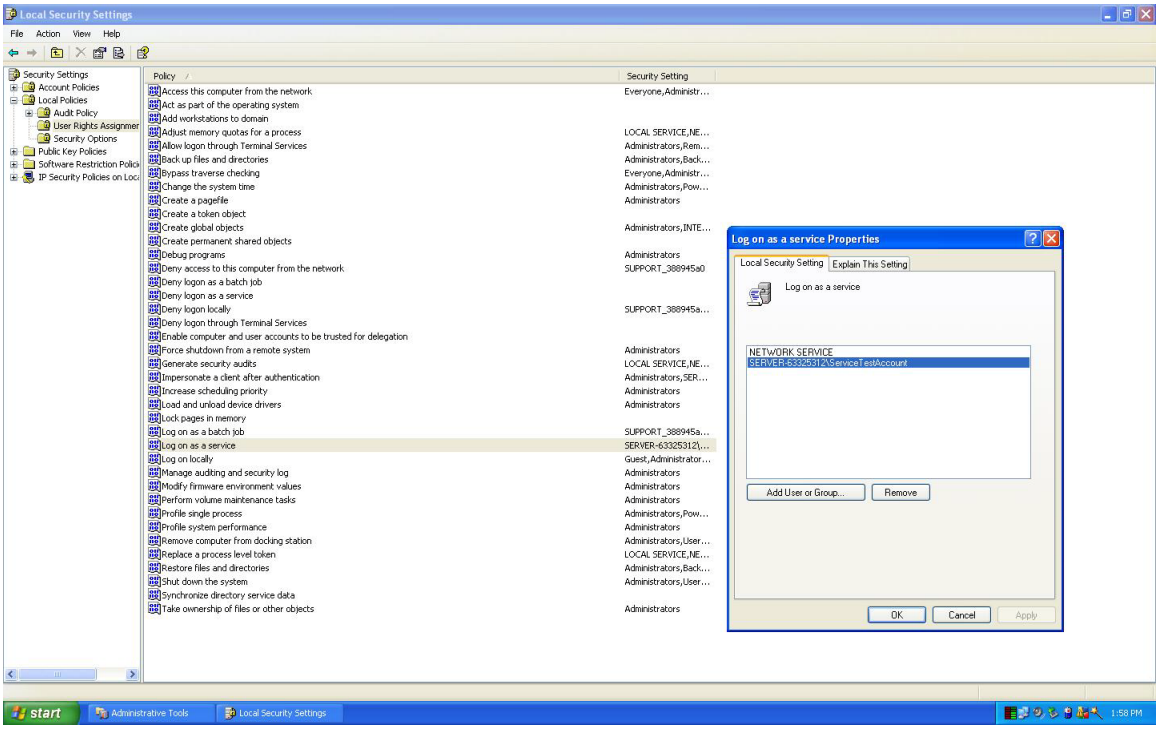
   Use an account name in the form: {DomainName}\{UserName} i.e wrapper.ntservice.account={DomainName}\{UserName}

   If the account belongs to the built-in domain, then you may specify the name in the form: .\{UserName}.

   If no value is specified, the LocalSystem account is used. This account does not have privileges to access network shares.

2. Grant 'Logon as Service' permission to the user account whose credentials have been provided.

   To set the "Logon as Service" right, goto the "Administrative Tools" folder in your control panel. Open the "Local Security Policy" applet. Expand "Local Policy" and then click on "User Rights Assignment". On the right side, you will find a "logon as service policy". Right-click or double-click to access its properties dialog, and then add the user that you wish to allow to run the service.



3. Open a command prompt. Go to %FIORANO_HOME%/esb/server/bin/service directory
4. To install FES/FPS(Replicated HA Profile) as an NT service, run the command install-server.service.bat -mode <mode> -profile %PROFILE_NAME% where <mode> is either 'fes' or 'fps'

   Example: install-server.service.bat -mode fes -profile haprofile1/primary

   Where, haprofile1/primary is the <PROFILE_NAME>

   To install FES/FPS(Shared HA Profile) as an NT service, run the command

   install-server.service.bat -mode <mode> -profile %PROFILE_NAME%

   Example: install-server.service.bat -mode fes -profile haprofile_shared/primary -dbPath \\192.168.1.213\db

   where <mode> is either 'fes' or 'fps'& haprofile_shared/primary is the <PROFILE_NAME%>

Important: -dbPath refers to the path of the network share hosting the shared database. In the above example,\\192.168.1.213\db is the network share. UNC Path has been used.

   Note: If mode is not specified, it is taken as 'fps'.

5. After installation you should be able to see "Fiorano ESB Server <%PROFILE_NAME%>" NT service or Fiorano Peer Server <%PROFILE_NAME%> NT service installed.
6. To remove FES/FPS NT service, run the command

   uninstall-server-service.bat -mode <mode> -profile %PROFILE_NAME%

where <mode> is either 'fes' or 'fps'

Example: uninstall-server.service.bat -mode fes -profile haprofile1/primary

7. Uninstall should stop and remove the NT service for the given profile from the services list.

2.10.5 Viewing Server Logs

Once started, console Logs for FES & FPS service can be viewed in the following log files:

%FIORANO_HOME%/runtimedata/EnterpriseServers/%PROFILE_NAME%/service/service.log.

%FIORANO_HOME%/runtimedata/PeerServers/%PROFILE_NAME%/service/service.log.

Other logs pertaining to the server can be found under:

- For Enterprise Server: %FIORANO_HOME%/runtimedata/EnterpriseServers/%PROFILE_NAME%/FES/run/logs
- For Peer Server: %FIORANO_HOME%/runtimedata/PeerServers/%PROFILE_NAME%/FPS/run/logs

2.10.6 FAQs

Question 1: How to change Memory Settings (Xms & Xmx) of Server while running as a service?

Answer: Uninstall the service (refer section How to install/uninstall a StandAlone (Non-High Availabilty)Profile as NT Service and How to install/uninstall a Profile as NT Service with High Availabilty Profile(HA))

Open 'server.conf' located under %FIORANO_INSTALLATION_HOME%/esb/server/bin and change the Xms & Xmx values. (Refer section Configuring Server JVM Settings for Running as NT-Service.)

Save the file and proceed to install the service again using the script.

Question 2: Certain Components like FTP-Put / FileReader / File Writer that are launched in-Memory are not able to access network shares?

Answer: Since the components are launched in-memory they are part of the peer server. The server needs to have user credentials of the System to access network shares when run as a nt-service. Refer Section How to install/uninstall a Profile as NT Service with High Availability Profile(HA) (Steps 1 and 2).

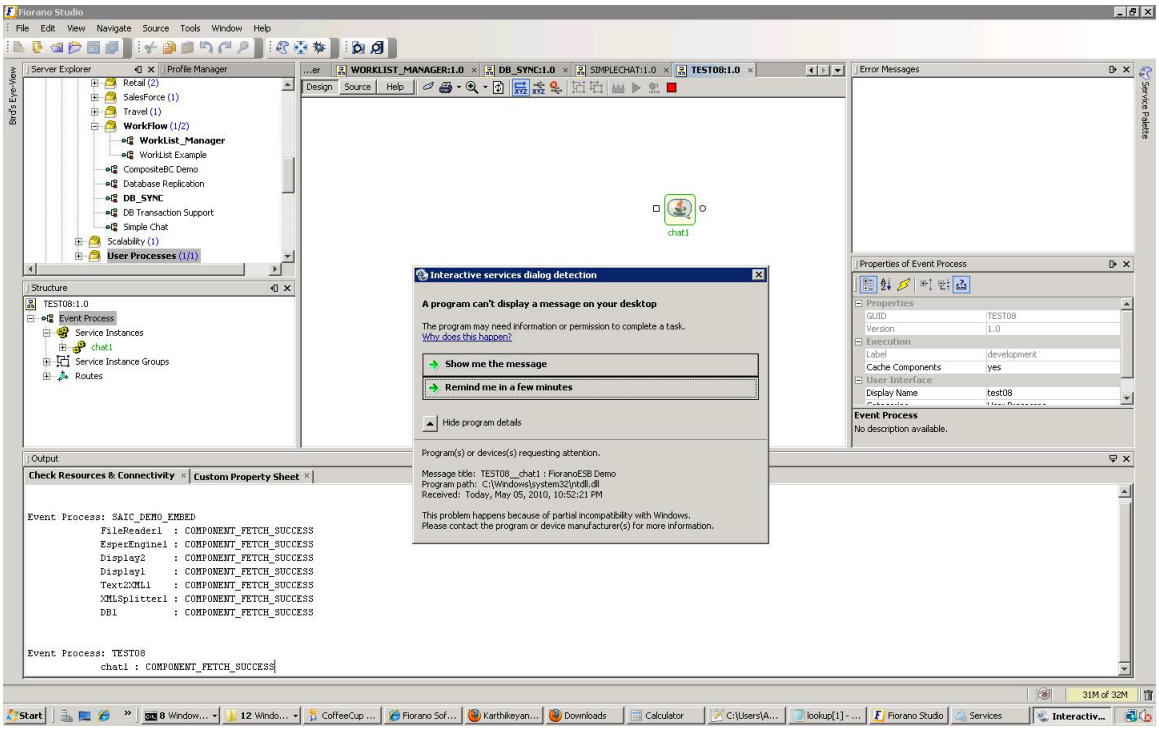
Note: You need to uninstall the service, make the required changes and then install the server as a service.

Question 3: Feeder / Display Components (Components with Graphical User Interface) do not show up on Windows server 2003, 2008 & Vista but they appear green from Fiorano Studio?

Answer: This is due to security restrictions put in place by the OS. The components will be visible on Desktop 0, which is a background desktop that you have to switch to see the service interface.

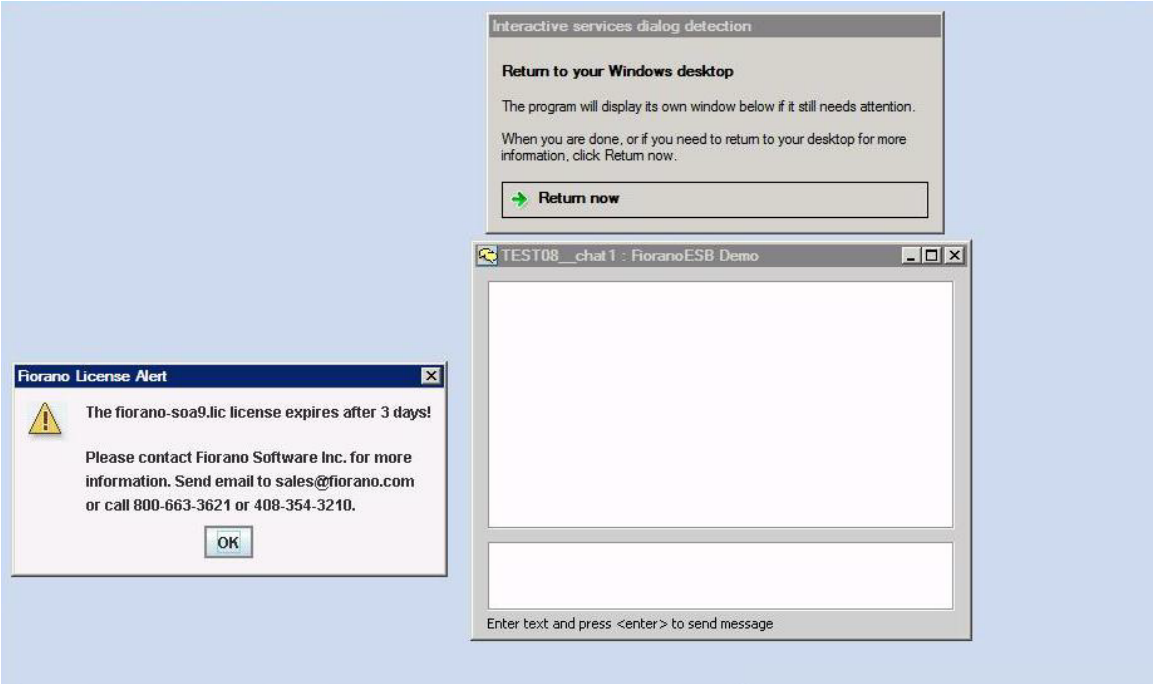
To see the components display, perform the following steps:

1. Start the service **Interactive service detection**. When we launch a flow containing the components like feeder/display, a pop-up should appear.



2. Click on **Show me the message**.

This should take you to the Desktop 0 of system.



3. To get back to the original desktop, click on the 'return now' option. you can move back and forth the desktops using the interactive services desktop popup dialog.

2.11 Event Monitoring

Fiorano ESB Server hosts various JMS topics for the ESB Peer Servers to publish information related to application, component and internal state changes. These events are used to keep the ESB Server and the Peer Server synchronized with each other. Events generated within the Enterprise Server are not published to any JMS topic and can be subscribed to using various other transport protocols. All these events are referred to as control messages within the Fiorano Network.

Custom monitoring applications can be written to subscribe to these control messages so as to monitor the Fiorano Network. In addition the Fiorano Network can be monitored using the Web Dashboard. This document gives insight of all the necessary details to build one such custom monitoring application in addition to describing the in-built monitoring and reporting capabilities.

2.11.1 Event Types And Subscription

This section describes various kinds of events that are raised in the Fiorano Network. These events are used to perform various tasks such as updating necessary data structures, synchronizing application states, notify connected clients and so on.

2.11.2 Audit Events

All audit events generated by the Peer Servers are published onto the Enterprise Server topic 'FES_AUDIT_EVENT_TOPIC'. These audit events provide a means of accountability for changes done in the system and also help in detecting a security violation in certain cases.

ApplicationAuditEvent

- **ApplicationLifeCycleAuditEvent**: This event is raised when there is a change in the Event Process life cycle such as from a "stopped" to a "launched" state or vice-versa.
- **ApplicationRepositoryAuditEvent**: This event is raised when there is a modification in the Event Process Repository such as when changes are made to an Event Process, Deletion of an Event Process, Addition of an Event Process to a repository and so on. This audit event is raised only within the Enterprise Server as the Peer Server does not have an Event Process repository of its own.

ServiceAuditEvent

- **ComponentLifeCycleAuditEvent**: This event is raised when there is a change in the Component life cycle such as from a "stopped" to a "running" state or vice-versa.
- **ServiceRepositoryAuditEvent**: This event is raised when there is a modification in the component repository of either the Enterprise Server or the Peer Server.

PrincipalStoreSyncAuditEvent

This event is raised when the Peer Server synchronizes its principal store with that of the Enterprise Server.

AuthenticationAuditEvent

This event is raised when the User tries to login/create a connection with the Server.

AuthorizationAuditEvent

This event is raised when the User tries to perform an operation that is protected by permissions granted to the User such as looking up a connection factory, creating a JMS connection, launching an event process, changing audit policies, deleting audit events and so on.

SecurityAuditEvent

This event is raised when there is a modification in the security database of the Server such as when there is a new User creation, a User deletion, a group creation, an addition of a member to a group, the granting of specific permissions to a User and so on.

Refer to **Chapter 12** in for detailed description of Audit Events and Audit Functionality as a whole.

2.11.2.1 Subscribing to Audit Events

To subscribe to audit events generated by any Server, add an audit event listener by opening profile in nStudio and navigating to Fiorano -> etc -> AuditManager. Right-click on AuditManager node and select Add -> AuditEventListener (see screenshot below). Provide a fully qualified class name for the audit listener class of this listener and add this class to the Server classpath.

Alternatively, Users may create a subscriber to topic **FES_AUDIT_EVENT_TOPIC** present in the Enterprise Server to listen to audit events from all connected Peer Servers. Note: Audit Events from the Enterprise Server need to be subscribed to by using the method described above.

2.11.3 System Events

System Events have an important place within the Fiorano System since they determine how the system behaves in a particular/specific situation. All Events are defined in the **fiorano.tifosi.dmi.events** package present in **$FIORANO_HOME/esb/shared/lib/esb-shared-tif-dmi.jar**. This section provides a brief explanation of these System Events and also describes various ways to subscribe to Events.

2.11.3.1Event Description

A brief explanation of various System Events is as given below:

1. **HAStatusEvent**: This Event is launched within the Enterprise Server when there is a change of state of an HA Server such as from the ACTIVE to the STANDALONE state.
2. **HARepositorySyncEvent**: This Event is launched within Enterprise Server when HA Servers synchronize their databases with each other.
3. **UserEvent**: This Event is launched within the ESB network by business components whose monitoring configuration has been enabled.
4. **ApplicationEvent**: This Event is launched by the Enterprise Server when an application is launched, killed, created, saved, or deleted. Application launch and kill events are also handled by the Peer Servers.
5. **RouteEvent**: The Enterprise Server launches this Event when a debugger is set or removed from a running Event Process.
6. **BacklogMonitorEvent**: This Event is launched by the Peer Servers when the configured backlog level is reached as defined by the backlog policies active on that Peer Server.
7. **LowMemoryEvent**: This Event is launched by all Fiorano Servers when they detect that memory usage of a Server has crossed the specified threshold.
8. **SBWEvent**: This Event is launched by the Peer Servers when a message reaches a port on which document tracking is enabled. On receiving these Events from the Peer Servers, the Enterprise Server stores them into a preconfigured document tracking database.
9. **SecurityEvent**: The Enterprise Server launches this Event when a User action results in security violations (as determined by the ACL of that particular User in the Enterprise Server such as launching or killing an Event Process or a Service Instance.
10. **ServiceEvent**: This Event is launched by the Peer Server when a Service Instance is launched or killed.
11. **ServiceRepositoryUpdationEvent**: The Enterprise Server launches this event when some modification is made to the Service Repository of the Enterprise Server such as registering a new service, addition of a resource to an existing service and so on.
12. **SPEvent**: The Enterprise Server raises this Event at the time of startup and shutdown.
13. **TPSEvent**: This Event is launched by both the Enterprise Server and Peer Servers when connecting to/disconnecting from the Enterprise Server. Note: On receiving this Event from a Peer Server, the Enterprise Server synchronizes the states of all Event Processes with the Peer Server.

Refer to section 2.8 for detailed description of Audit Events and Audit Functionality as a whole.

2.11.3.2 Subscribing to System Events

System Events generated within the Peer Servers are published onto various monitoring topics. Users can create subscribers to these topics to listen to the Events generated by the Peer Servers. (See next section for the list of events published onto various such topics.)

Note that such a subscriber would not receive events generated within the Enterprise Server, as these Events are not published onto any topic. For listening to Events generated within both the Enterprise Server and the Peer Server using a single application Users can register an Event listener within the Enterprise Server using the Run Time Library (RTL) APIs. These RTL APIs provide support for subscribing to Events from the Enterprise Server and the Peer Servers such as:

- HAStatusEvent
- HARepositorySyncEvent
- UserEvent
- ApplicationEvent
- RouteEvent
- SBWEvent
- SecurityEvent
- ServiceEvent
- ServiceRepositoryUpdationEvent
- SPEvent
- TPSEvent

A RTL Event Subscriber can listen to all System Events except Backlog and Low Memory Events. To subscribe to these two of Events from the Peer Server, create a subscriber for the topic 'FES_BACKLOG_MONITORING_TOPIC' and for the topic 'FES_SYSTEM_EVENTS_TOPIC' along with appropriate message selectors as shown in the next section.

To subscribe to Low Memory Events from within the Enterprise Server, Users need to register a JMX notification listener with the JMX Object identified using the following name: *Fiorano.jmx.notifications:ServiceType=EventManager,Name=ResourceEventManager.*

Refer to $FIORANO_HOME/fmq/samples/JMX/JMXNotifications.java for a sample code to register the JMX Notification listener.

Note: Please refer to the section named Sample Subscriber Applications for JAVA Samples to create a Topic Event Subscriber and a RTL Event Subscriber.

2.11.4 Event Topics

System Events from Peer Servers are published onto various monitoring topics present within the Enterprise Server. Each type of Event has a JMS message header property named 'DATA_TYPE' or 'EVENT_TYPE' set to a unique value to distinguish between various Events. This section provides a list of the Event Topics along with the list of Events that are published by the Peer Servers. Please note the different values given for 'DATA_TYPE' / 'EVENT_TYPE' properties of these Events. These values may be used to create message selectors Users wish to subscribe to only a selected set of Events from an Event Topic.

1. **FES_SYSTEM_EVENT_TOPIC**
   - Application Event (DATA_TYPE = 302)
   - Service Event (DATA_TYPE = 303)

- TPS Event (DATA_TYPE = 305)
- Low Memory Event (DATA_TYPE = 404)

2. **FES_SBW_EVENTS_TOPIC**
   - SBW Event (DATA_TYPE = 307)
3. **FES_USER_EVENTS_TOPIC**
   - User Event (DATA_TYPE = 217)
4. **FES_BACKLOG_MONITORING_TOPIC**
   - Backlog Monitor Event (DATA_TYPE = 403)
5. **FES_AUDIT_EVENT_TOPIC**
   - Audit Event (EVENT_TYPE = 600)

     Note: The value of DATA_TYPE will be different for different types of audit events as shown below.

     - Application Life-Cycle Audit Event (DATA_TYPE = 105)
     - Application Repository Audit Event (DATA_TYPE = 103)
     - Component Life-Cycle Audit Event (DATA_TYPE = 106)
     - Service Repository Audit Event (DATA_TYPE = 104)
     - Principal Store Sync Audit Event (DATA_TYPE = 108)
     - Authentication Audit Event (DATA_TYPE = 1)
     - Authorization Audit Event (DATA_TYPE = 2)
     - Security Audit Event (DATA_TYPE = 3)

2.11.5 Sample Subscriber Applications

Please refer to the samples named TopicEventSubscriber.java and RTLEventSubscriber.java located under $FIORANO_HOME/esb/samples/SamplePrograms/EventSubscriber which illustrate the use of APIs to subscribe to various Events. Custom applications can be built around these samples to perform various actions on receiving the Event.

2.11.6 Built-In Event Tracking / Reporting

Using the Fiorano dashboard or the Event Manager tool, Users can view various types of System Events raised within the Fiorano Network. As these events are stored into configurable a SQL database, Users can search for historical events using these tools. The Event Viewer can be used to view the System Events listed below.

- Application Events
- Service Events
- Security Events
- TPS Events
- SP / Enterprise Server Events

- SBW Events

In addition, FioranoMQ provides built-in support for registering SMTP alerts for various types of System Events. For Backlog and Memory Events, Users have a choice to sign up for JMS alerts. Alert registration can be done via the dashboard. Refer to sections 2.5.1.1 and 2.5.1.4 for further instructions. Alerts can be registered for the System Events listed below.

- Application Events
- Service Events
- Security Events
- TPS Events
- SP / Enterprise Server Events
- Backlog Notification Events
- Low Memory Events

2.11.7 Event Parameters

Various Enterprise Server Level parameters determine whether the Enterprise Server will listen to a particular type of Event and whether these events will be forwarded to RTL clients. These parameters can be configured via Profile Manager in nStudio. The parameters that may be configured are as described below:

1. **FES ->Fiorano ->Esb ->Events ->FESEventsManager**

   - EnableSystemEventTracking

     This parameter determines whether System Events will be inserted into the Events databasefor future reference. By default, this property is **Enabled**.

   - ListenForUserEvents

     This parameter determines whether the Enterprise Server will create a subscriber to listen for User Events. If RTL clients are to receive User Events, this parameter should be set to **Yes**. By default, this property is set to **No**.

   - RTLToReceiveUserEvents

     This parameter determines whether User Events captured by the Enterprise Server (given the ListenForUserEvents flag is **Enabled**) will be forwarded to RTL clients. By default, this property is set to **No**.

   - RTLToReceiveSecurityEvents

     This parameter determines whether Security Events raised within the Enterprise Server will be forwarded to RTL clients. By default this property is set to **No**.

2. **FES ->Fiorano ->Esb ->Sbw ->SBWManager**

   - ListenForSBWEvents

     This parameter determines whether the Enterprise Server will create a subscriber to listen for SBW Events published by the Peer Servers. If RTL clients are to receive SBW Events, this parameter should be set to **Yes**. By default, this property is **Enabled**.

- EnableDocTracking

  This parameter determines whether the SBW Events received by the Enterprise Server (given the ListenForSBWEvents flag is **Enabled**) will be inserted into the SBW database for future reference.

- RTLToReceiveSBWEvents

  This parameter determines whether the SBW Events captured by the Enterprise Server (given the ListenForSBWEvents flag is **Enabled**) will be forwarded to RTL clients. This is an expert property specified under the profile node listed above.

2.11.7.1Examples

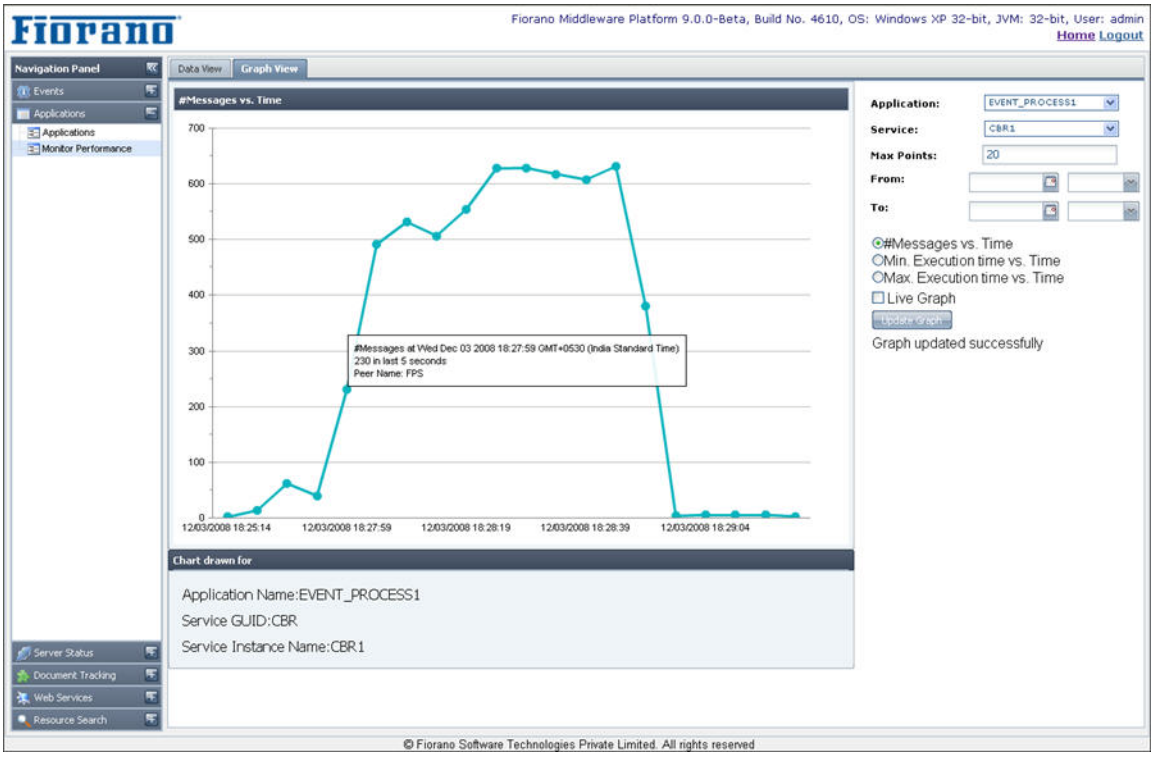
This section provides some examples of extracting useful information such as response time, throughput and so on from the Events described above.

2.11.7.2 Measuring Response Time

Each SBW Event captures information of the time the message was tracked. This information can be used to measure the response time of a service by enabling document tracking at both input and output ports of the service. This feature can also be used to measure the time taken for the message to flow from one service to another. In the screenshot below, document tracking has been enabled on the input port of chat1 service and output port of chat2 service. It is possible to view that it took 69 milliseconds for the message to flow from chat1 service to the chat2 service.

2.11.7.3 Measuring Throughput

Fiorano pre-built components provide functions to enable monitoring via their Custom Property Sheet (CPS) panel. When monitoring is enabled, these components publish information about the messages processed within configured interval in the form of User Events. These events provide information about the number of messages processed within this time. The dashboard can be used to view data both in tabular as well as graphical forms (see screenshot below). Please, refer to **section 2.5.1.3** for more details.



2.11.7.4 Service Status Monitoring

Service Events raised by the Peer Server contain information about the current status of the service such as whether or not the service is running. Using the dashboard, Users can register to receive SMTP alerts when a particular service within an Event Process is launched or killed. These alerts can be useful in detecting abnormal service termination due to unexpected errors such as a JVM crash or an unintentional shutdown. Please refer to **section 2.5.1.1** in for more details.

2.11.7.5 Messaging Threshold

Using the dashboard, Users can create backlog policies and define SMTP/JMS alerts to be sent when a service has more than the specified number of messages in its pending messages queue. Users can also create more than one policy for a single service to receive alerts at multiple levels of a pending message queue. For example, Users can register to receive SMTP/JMS alerts when the size of the pending message queue reaches 5, 10 or 15 either in an upward or a downward direction. Please refer to **Chapter 11** in for more details on Backlog Monitoring feature.

2.11.7.6 Server Memory Threshold

Using the dashboard, Users can create memory usage policies to receive SMTP/JMS alerts when a Server memory reaches a specified threshold. These alerts can be used as a preventive measure to avoid further problems caused by Out Of Memory problems. Please refer to **section 2.5.1.4** in for more details.

Chapter 3: Component and Component Instances

This chapter discusses Service components - the building blocks of Fiorano composite applications and Event Processes. A service component is an application that performs a specific task (for example, sending e-mails or reading a file).

This section below discusses Service Component characteristics, together with the details of configuring, deploying and managing service components in the Fiorano Environment.

3.1 Service Components Characteristics

Service Components may be synchronous or asynchronous. Service components are loosely coupled to allow them to interact by exchanging messages in a flexible manner within a Fiorano Event Process. Service components are the building blocks of Fiorano applications (Event Processes). A service component is an application that performs a specific task (for example, sending e-mails or reading a file). Service components are loosely coupled to allow them to interact by exchanging messages in a flexible manner within a Fiorano Event Process. Each Service Component has an interface that defines the acceptable formats of each input and output. Most service components typically require runtime parameters and other variables to be configured before they are run. Configurations allow customization of Service Components for the specific business scenario/situation being solved by the Event Process. Service components can be categorized as synchronous and asynchronous, based on the manner in which a component is invoked for processing.

3.1.1 Synchronous Components

A synchronous component, as the name suggests, is invoked in request – reply format. Thus the invoking client of this component waits for the component to process the request and send back the response. Fiorano synchronous components implement the J2EE Connector Architecture (JCA) interface, which mandates that each component has a single input and single output. JCA is a standard API that is part of the J2EE platform that implements synchronous “function calls” semantics.

The Fiorano platform includes a Business Component Development Kit (BCDK) for the development of synchronous components. This framework abstracts the implementation details of JCA.

All synchronous components can also be invoked asynchronously. This is achieved through the usage of a JMSGateway component that provides a layer (wrapper) on top of the BCDK to make the components event-driven, allowing them to be invoked using asynchronous JMS semantics.

3.1.2 Asynchronous Components

Asynchronous components or event-components are based on JMS semantics. Each asynchronous component can have multiple inputs and outputs. Inputs of asynchronous Fiorano components listen for events (JMS messages) via listeners on specific JMS destinations.

Asynchronous components have the concept for ports which are abstracted form of JMS destinations. A port can be a queue or a topic and components listen to input ports for JMS messages and send JMS messages on to the output ports. An asynchronous component can define how many input and output ports it needs. These can be added through static definition or dynamically when a component is configured.

3.1.3 Design Choices

When developing a component in the Fiorano SOA Platform, a design decision of when to use a synchronous component versus an asynchronous component requires a careful understanding of the strengths of each of the types of components. The following details for each of the component types are useful in making a decision:

Synchronous Components

- Scheduling, error handling and connection pooling are abstracted
- Can be deployed in external J2EE JCA containers
- Cannot have multiple ports in output/input
- Cannot handle server sockets
- Cannot have runtime UI with manual intervention
- Supports only Java programming language for component creation

Asynchronous Components

- Allows multiple input and output ports
- Supports multiple programming languages for component creation like Java, C, C++, C# etc.
- Handles server sockets
- Allows components to have manual intervention
- Does not have abstracted layers like synchronous components

Based on the business context, users can determine the type of component to use and/or create.

3.2 Service Component Characteristics, Configuration, and Deployment

As discussed above, a service component is typically configured for purposes of customization on a per instance basis when used within a Fiorano application.

3.2.1 Component Launch Semantics

Fiorano components can be launched in three ways.

1. **Separate Process:** When using the components in a Fiorano Event Process, the component instance can be configured to be launched as a separate process. This is the default launch mode, and in this mode the Fiorano Peer Server launches the component as a separate process; for components written in Java, a separate JVM is created for each component instance. The Fiorano Peer Server controls the launch and stop of components launched in separate processes.
2. **In-Memory:** Components written in the Java programming language can be launched in the same process as the peer server. This is the in-memory launch mode, in which the component is launched in a separate thread of the JVM of the Fiorano Peer Server. As in the separate process launch mode, the Fiorano Peer Server controls the launch and stop of components launched in-memory. Please refer to section 3.4.8 InMemory Launch for further details on In Memory launch.
3. **Manual:** When an end-user wishes to control the launch and stop of the components in a Fiorano application, specific components can be set to launch in ***manual mode***. When a component is set to launch in the manual mode, the Fiorano Peer Server does not try to launch the component. The end-user can use the Fiorano scriptgen tool present in <fiorano_install_dir>/esb/tools/scriptgen as illustrated in Figure 3.2.1 to generate a script that launches the component. Alternatively, the end-user can use the Fiorano Enterprise Server API to launch this component from another application. Please refer to section 3.4.6 Manual Deployment for further details on using scriptgen.

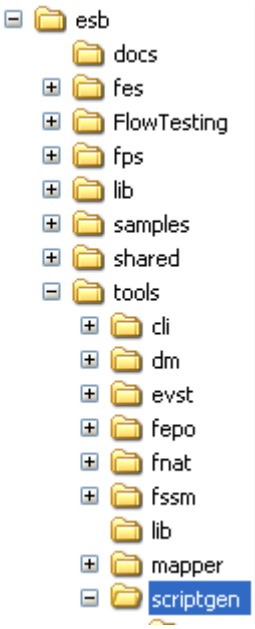


Figure 3.2.1: Directory structure showing scriptgen tool location for manual launch

3.2.2 Setting Component Launch Type in the Fiorano Studio

The launch mode for each component can be set using the Fiorano Studio which is part of the Fiorano Studio Tool. On selecting a component, the launch mode in the menu can be changed to one of the three modes mentioned and as illustrated in the Figure 3.2.2.

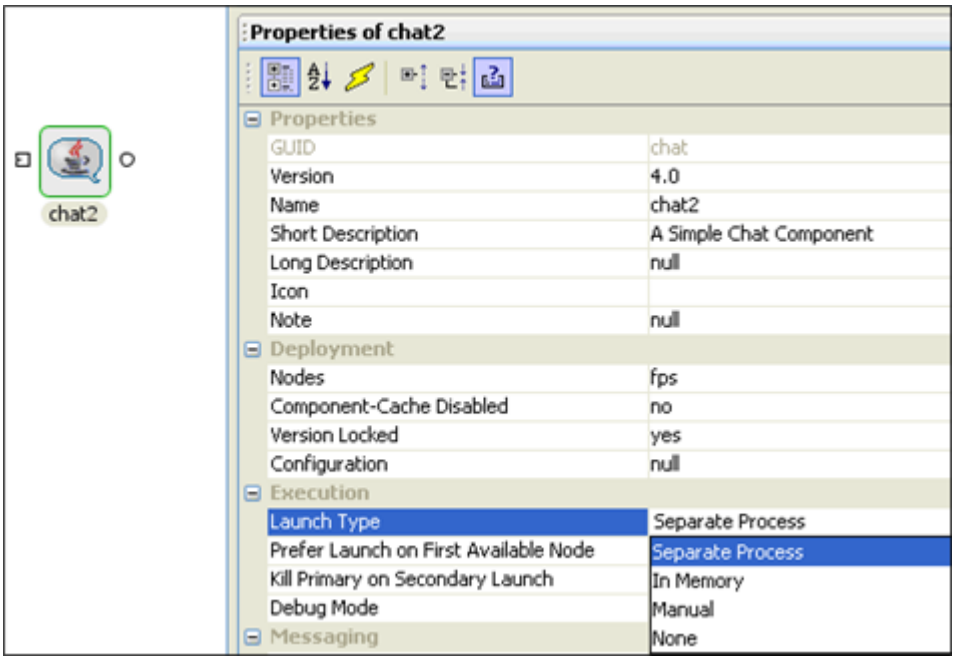


Figure 3.2.2: Changing launch mode of a component in Fiorano Studio

Every component expects a set of arguments at launch time. The runtime arguments are typically part of the configuration property sheet that is associated with the component, which is customized on a per-component-instance basis. Runtime arguments are processed at the startup of the component.

3.2.3 Launching Components Using the Fiorano Studio

When using the Separate Process and the In-memory launch types, the launch and stop of the components is controlled using the Fiorano Studio tool (present within the Fiorano Studio). There are specific toolbar buttons available to launch and stop components as illustrated in the Figure 3.2.3. The tool internally uses the Fiorano Enterprise Server API to launch and stop any components.

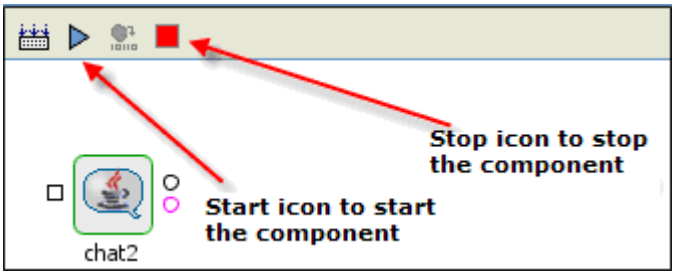


Figure 3.2.3: Component Launch and Stop buttons in Fiorano Studio

You can also launch and/or kill a component in a running Fiorano application using menus.

3.2.3.1 Launching a Component in a Running Application

Right-click on the component and select **Execution**. Select **Run** from the menu list as illustrated in Figure 3.2.4.

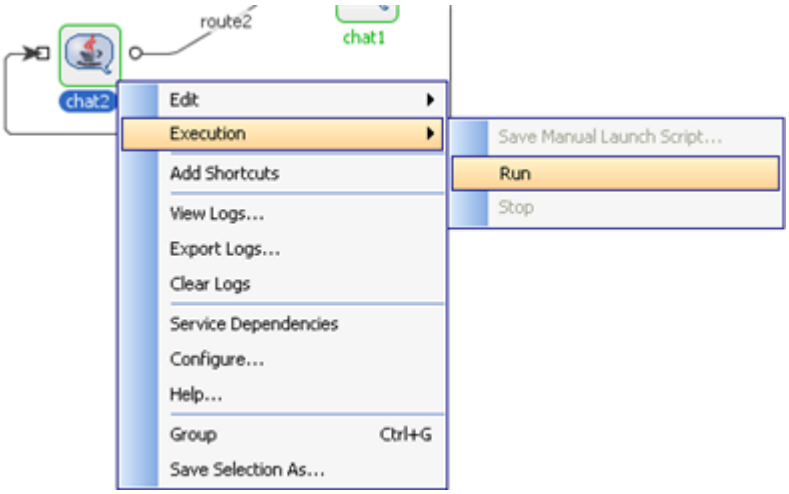


Figure 3.2.4: Launching a Component

3.2.3.2 Stopping a Running Component Instance

1. Right-click on the component and select **Execution**.
2. Select **Stop** from the menu list as illustrated in Figure 3.2.5.

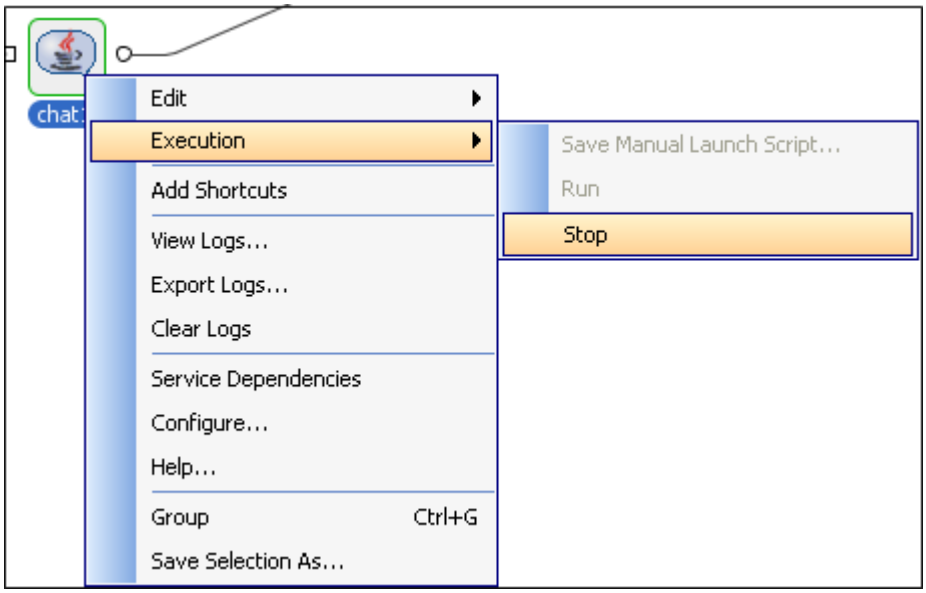


Figure 3.2.5: Stopping a Component

3.3 Service Component Configuration

Most components have a set of configurable parameters which can be set to appropriate values to customize instances of the component for a particular Fiorano application. Configuration is captured within a configuration property sheet as described in the following sections.

3.3.1 CPS for Component instance configuration

Most components have a GUI-based Configuration Property Sheet (CPS). This is usually a dialog or wizard which has a list of configurable parameters for the component. These customizable parameters have their own editors to facilitate the end user in adding appropriate values.

3.3.1.1 Launching the CPS

1. Right-click on the component.
2. Select the **Configure** option from the menu list.

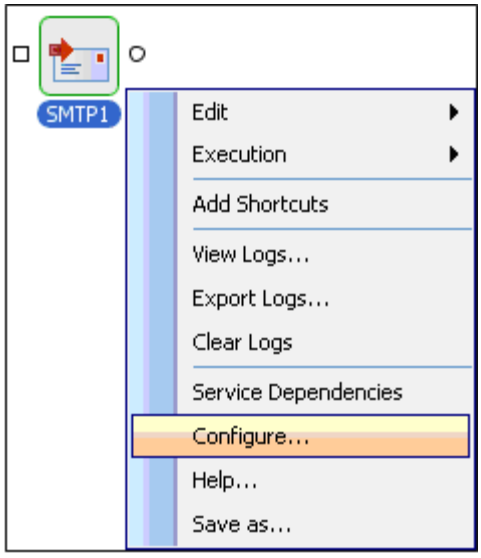


Figure 3.3.1: Launching the CPS

3. Alternatively, you can double-click on the component to open the **CPS**.

The resources of the component are loaded to show the CPS and a dialog box illustrates the progress of the process as shown in the Figure 3.3.2.

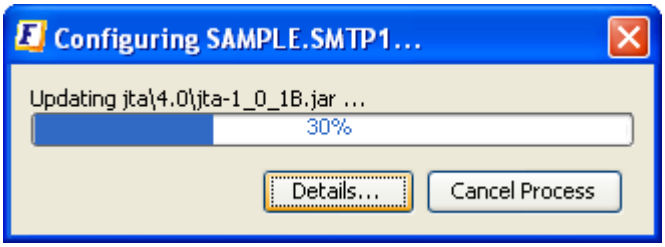


Figure 3.3.2: Fiorano Studio Dialog Loading the Resources

Figure 3.3.3 illustrates the loaded CPS of the SMTP component as an example.

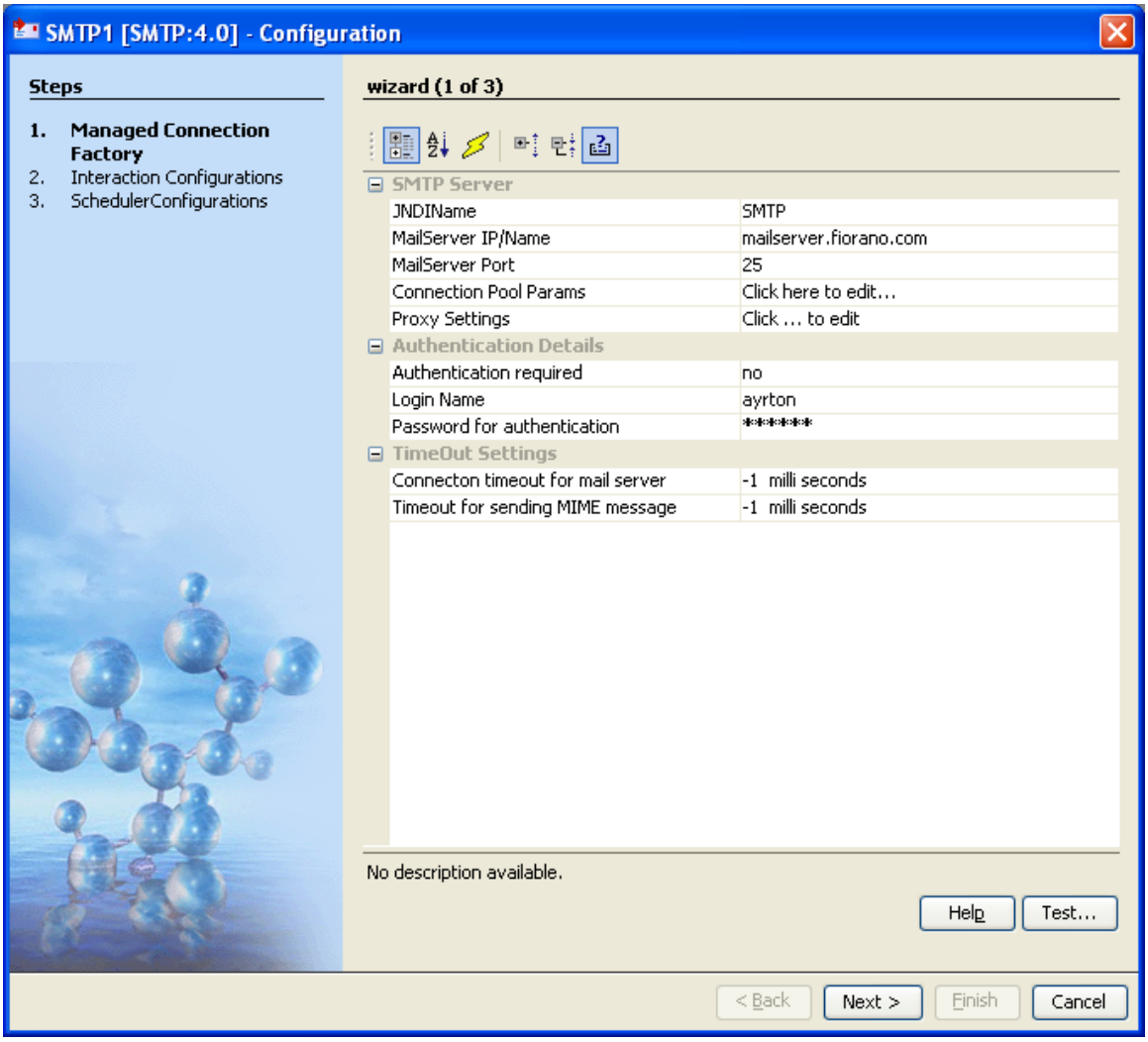


Figure 3.3.3: A typical CPS (SMTP component)

3.3.1.2 Customizable and Expert Properties

A component's customizable properties are typically different from that of other components. Some properties are hidden by default and are shown only when the "Show Expert Properties" icon is clicked as illustrated in the Figure 3.3.4. These are a set of advanced properties that are specific to each particular component instance.

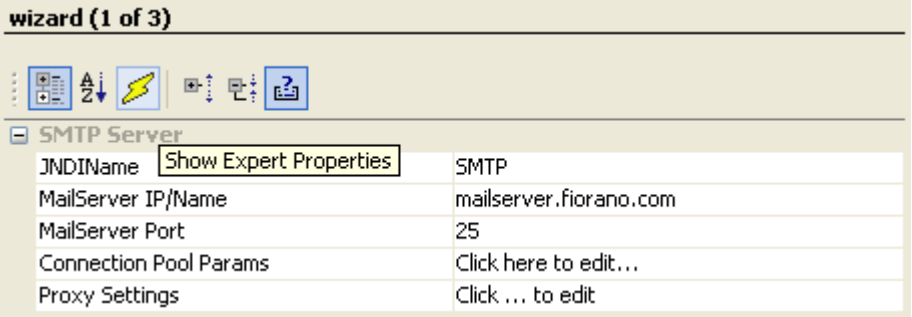


Figure 3.3.4: A yellow lightening symbol enables/disables the expert properties

3.3.1.3 Online Help for components

Every CPS has a help button (as shown in the Figure 3.3.5), which explains the properties being captured in the CPS. Online help provides describes each property, together with the kind of value(s) expected by the property.

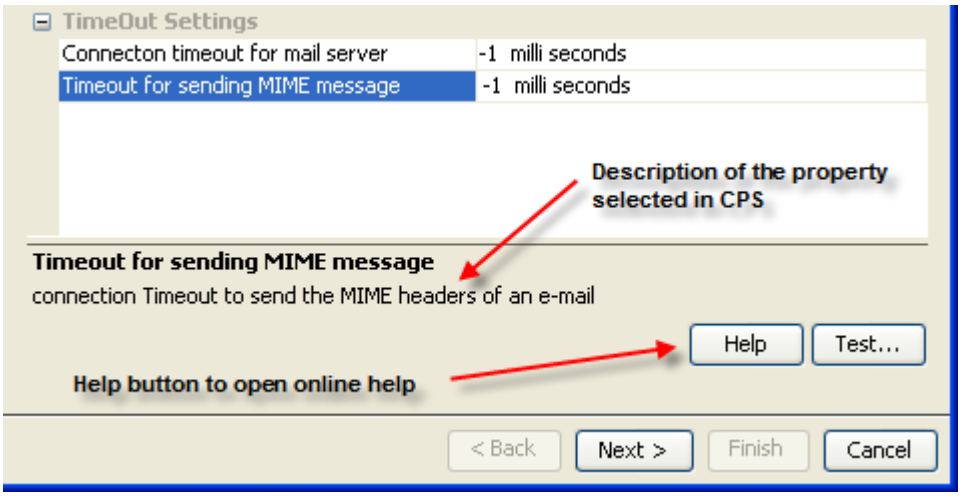


Figure 3.3.5: Help button in CPS for context sensitive help

The description for the selected configurable property can be seen above the help button. Figure 3.3.6 illustrates a typical help screen that shows when a help button is clicked.

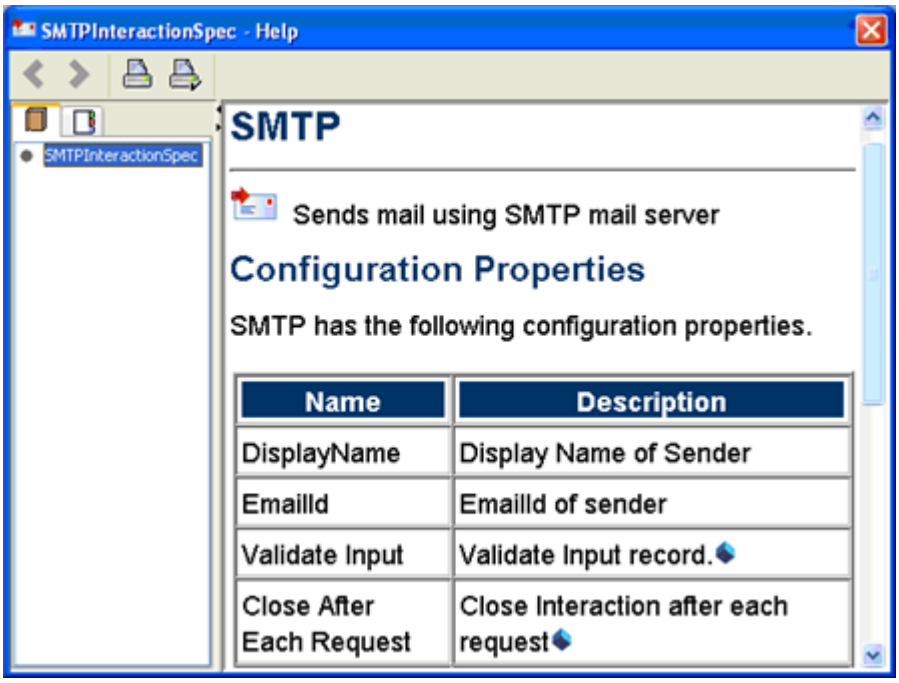


Figure 3.3.6: Typical help screen on clicking the help button

When a CPS is closed by clicking the Finish button, the configuration is persisted as part of the Fiorano application. This is later made available to the component at runtime via JNDI lookup.

3.3.1.4 Runtime Arguments

On selecting a component in the Fiorano Studio, the runtime arguments for the component are displayed in the properties window (as shown in the Figure 3.3.7). These runtime arguments can be accessed within the component at runtime through the parameters passed to the component when launching it using a launcher. A standard runtime argument added for all Java components is "JVM_PARAMS". This captures the JVM parameters which the end user might want to pass when the component is launched a separate process. Runtime arguments for a component can be added using the Fiorano Studio.

If the component's launch mode is changed to InMemory, the JVM_PARAMS property will not be displayed in properties view. To use the properties passed as JVM_PARAMS for an InMemory launched component, these properties need to be added to peer server's configuration file under <java.system.props> tag before starting the peer server. Peer server's configuration file can be located as either $FIORANO_HOME/esb/server/bin/server.conf or $FIORANO_HOME/esb/fps/bin/fps.conf depending on the script being used to run the peer server.

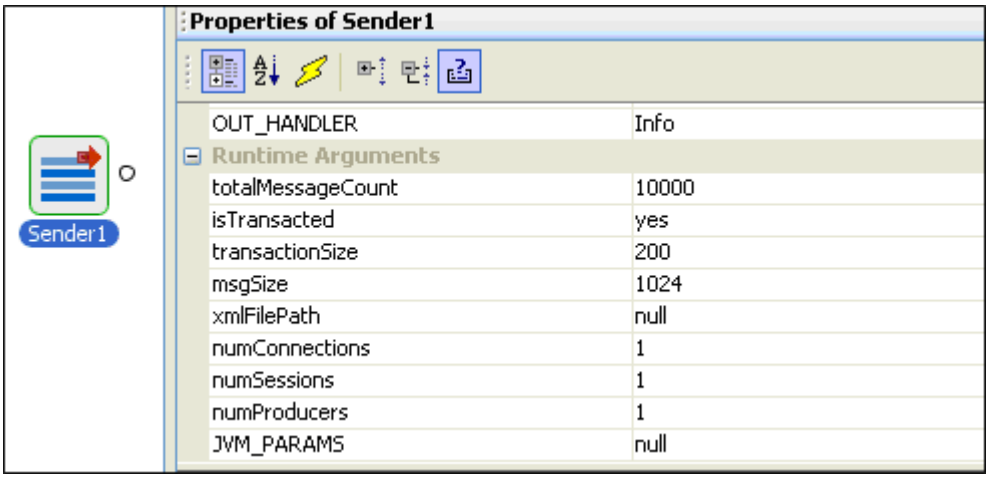


Figure 3.3.7: Fiorano Studio showing runtime arguments in properties window

Note: Changing the Node Name at runtime for Worklist and Aggregator Components is not supported. Unlike other components, Worklist and Aggregator components have state information written to the local disk.Moving the Worklist (or Aggregator) from one peer server to another one results in state data loss. In case of Worklist, not only the data loss, the external application, that is, Worklist web application will not show work items saved in the Worklist after the node change.

To achieve high availability for Stateful components, configure the back-end data store in clustered/HA relational database, like Oracle, DB2, and so on. Or deploy the components on a Peer Server that is running in Shared HA mode.

3.3.2 Component Dependencies and System Libraries

Every component has a set of dependencies on various libraries. Libraries can be the JAR files, DLLs, property files, configuration XML or any file which the component would require either at configuration time or at runtime. Fiorano already provides some standard libraries (some of them being 3rd party), which can be directly added to the list of dependencies for a component. If there is a library which is needed only for a particular component, it can be added exclusively for that component.

The Fiorano Studio is used to add dependencies to an existing component, as illustrated in Figure 3.3.8 and explained in more detail later in this section. You can select the component for which you need to add a dependency and select Edit (as shown in Figure 3.3.9 and 3.3.10). The dialog shows two sections

- The first section is where you add dependencies which are applicable only for this component. A copy of the dependency library is kept with the component.
- The second section is where you add dependencies which are already registered with Fiorano. These are called system libraries. In this case a copy of library is not added the component. There is reference created in the component to locate where to find the dependency library files.

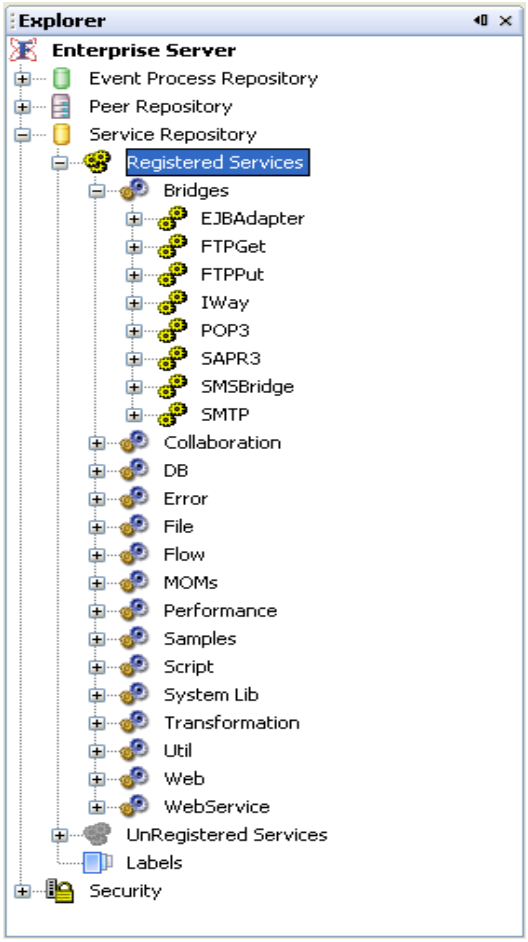


Figure 3.3.8: Explorer view of Fiorano Studio showing the component repository

3.3.2.1 Viewing the Resources of a Component

1. Right-click on the component and choose **Edit** from the drop-down menu as shown in Figure 3.3.9.

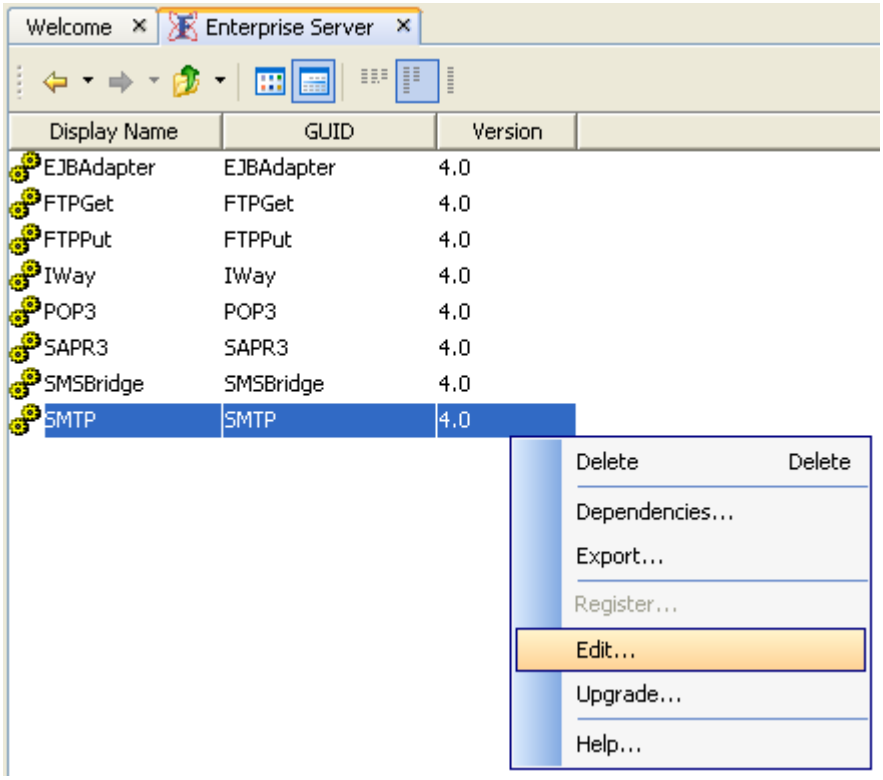


Figure 3.3.9: Using the Edit **option to add resources using the Fiorano Studio**

Figure 3.3.10 illustrates the resources of the SMTP component as an example.

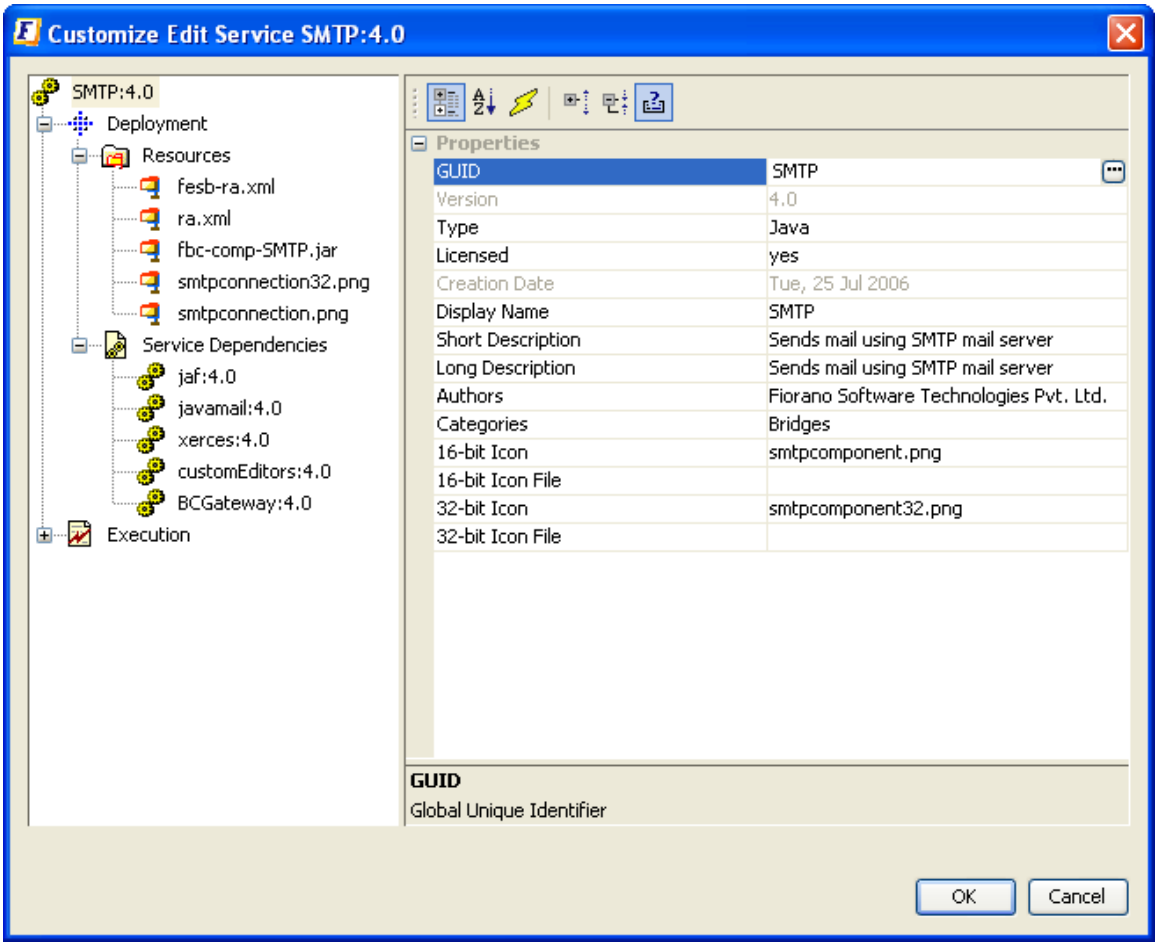


Figure 3.3.10: List of dependencies for SMTP component

3.3.3 Add New Library Dependencies

New dependencies can be added using the Add Resources option. Figure 3.3.11 illustrates how to add new dependency libraries which are applicable only to this component. Figure 3.3.12 and 3.3.13 illustrate how to add new system libraries, which are already registered with the Fiorano Enterpriser Server as dependencies to this component.

3.3.3.1 Adding New Resource/Dependency

1. View the Resources of the Component as described in the section 3.3.2.1 Viewing the Resources of a Component.
2. Right-click on the Resources tree menu on the left hand side as shown in Figure 3.3.11.
3. Choose the **Add Resources** option from the menu list. Using the file chooser, select the files to add as resources.

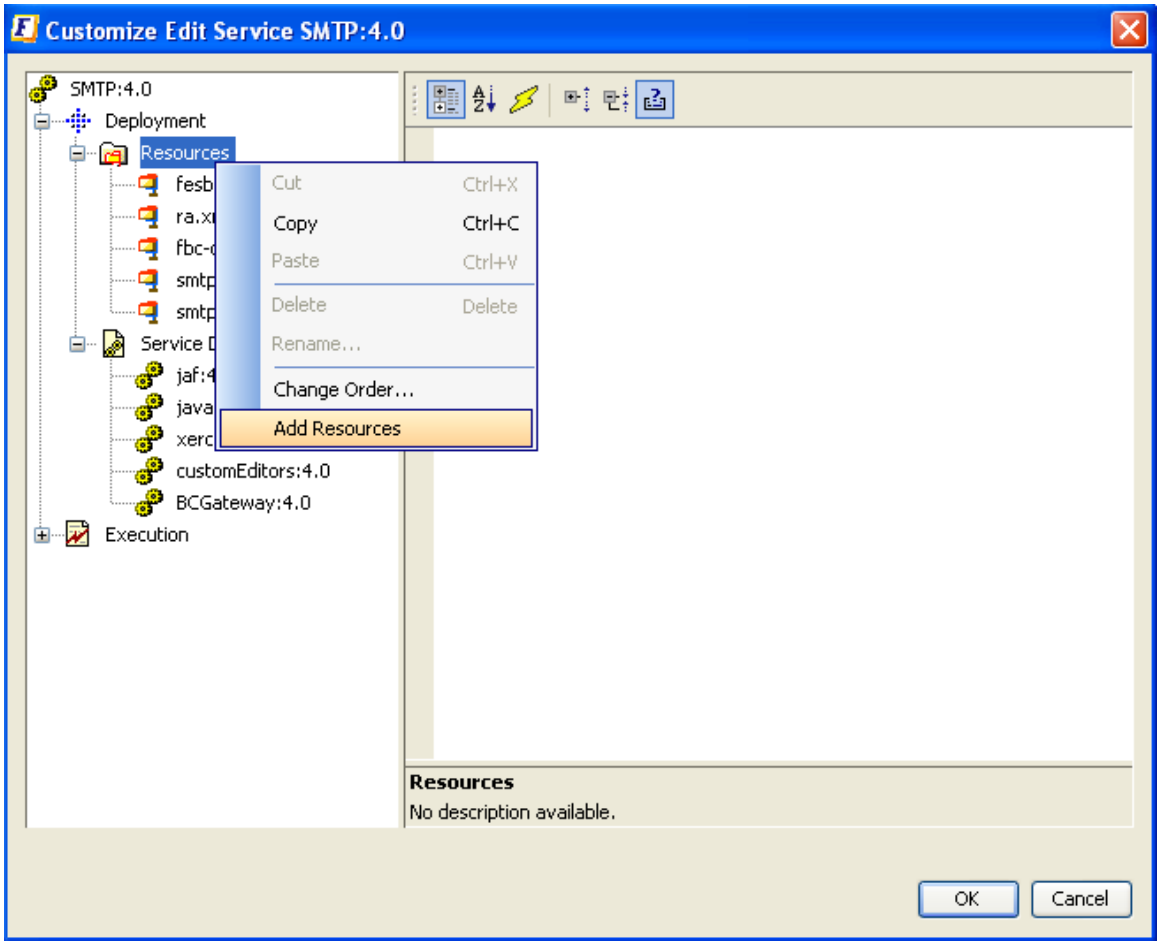


Figure 3.3.11: Right-click on Resources to add new resources

3.3.3.2 Adding Service Dependencies

1. Right-click on the Service Dependencies tree menu on the left hand side (as shown in Figure 3.3.12).
2. Select the option **Add Service Reference** from the menu. A **Customize Service Reference** dialog box appears, as shown in Figure 3.3.13. Select one or more of the system libraries in this dialog and click **OK** to apply changes.

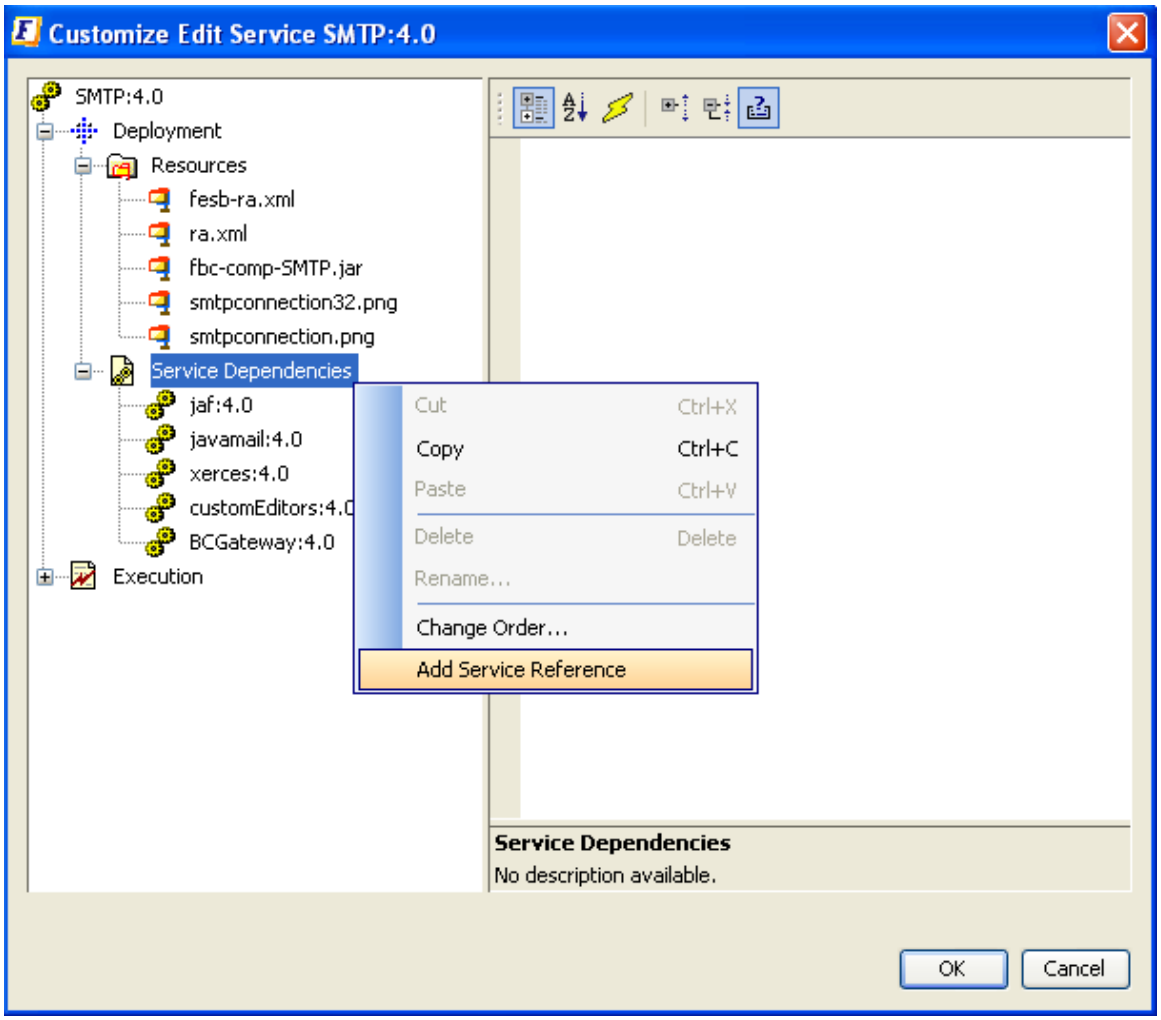


Figure 3.3.12: Right-click on Service Dependencies to add new system library reference

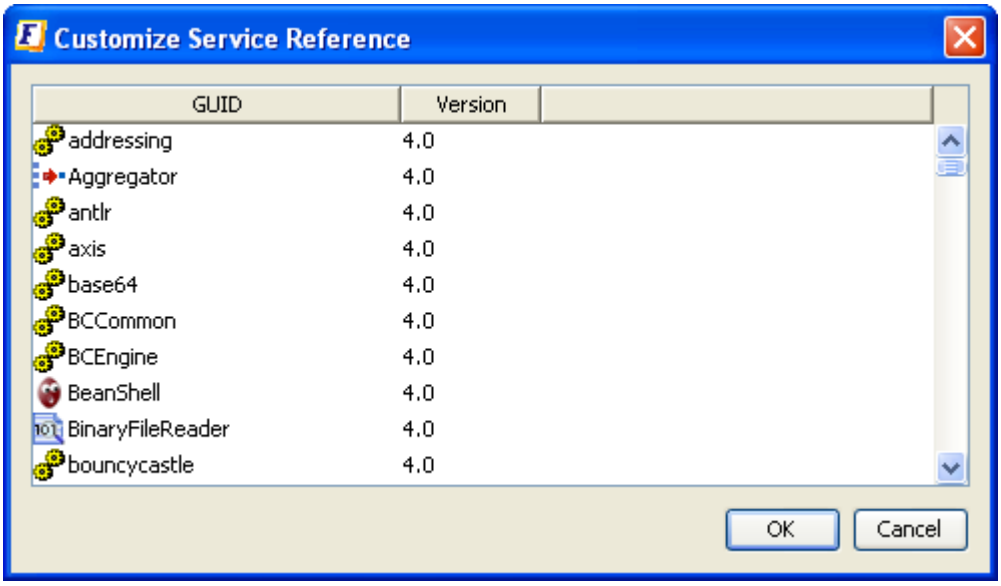


Figure 3.3.13: UI to add system libraries as dependencies.

The following table lists the system libraries developed by the Fiorano for use within component implementations. Not that this does not include any 3rd party libraries.

Service Dependency	Jar File Name	Description
CompositeComponentEngine	cce.jar	Implementation for the BPEL process.
CompositeComponent	fbc-comp-CompositeComponent.jar ,fesb-comp-CompositeComponent.jar	Component for execution of BPEL process.
BCCommon	bcc.jar	Common classes required by components.
BCEngine	bce.jar	TrGateway(Transport Gateway for BCs) and BCDK(Abstract Implementation for all the BCs)
BCGateway	fesb-comp-bcgateway.jar	EDBC wrapper for BC components over JMSGateway
customEditors	fbc-comp-customEditors.jar	Editors to be used in CPS to capture properties like SchemaEditor(XSD,DTD),ErrorPanels(ErrorConfiguration),SSL Panels(SSLConfiguration) and so on.

Service Dependency	Jar File Name	Description
Jdbc	fbc-comp-jdbc.jar	Classes which handle Database specific jdbc operations
Framework	Framework.jar	LicenseManager,Swing,xml(dom,sax,saxon,xsd related) classes
TifosiJavaRTL	TifosiJavaRTL.jar	Filechooser,wizard,xsdanddtd parsersupport,tifosihelpbroker classes.
esbCustomEditors	fesb-comp-esbCustomEditors.jar	Editors (which are specific to Fiorano ESB) to be used in CPS.
FioranoJavaRTL	fmq-client.jar,fmq-rtl.jar	JMS implementation classes of FioranoMQ.
Dmlparser	fbc-comp-dmlparser.jar	Parser for DML(Data Manipulation Language: insert,delete,update and select) statements.
FileMatcher	fbc-comp-filematcher-api.jar,fbc-comp-filematcher-local.jar	Classes required for searching file/directory names based on filename pattern.
Transformer	fbc-comp-Transformer.jar	Transformer engine classes for converting

3.3.3.3 Adding Resources to Class Path

This section demonstrates adding JDBC drivers to classpath of all adapters in DB component. All the components under DB category share a common System Library which handles the JDBC connection; hence JDBC drivers should be added as resources to System Library JDBC.

Any additional resources required by any component can be added in similar fashion after identifying the correct component or system library.

1. Login to **Enterprise Server** and navigate to Enterprise Server → Service Repository → Registered Services → System Lib → JDBC.

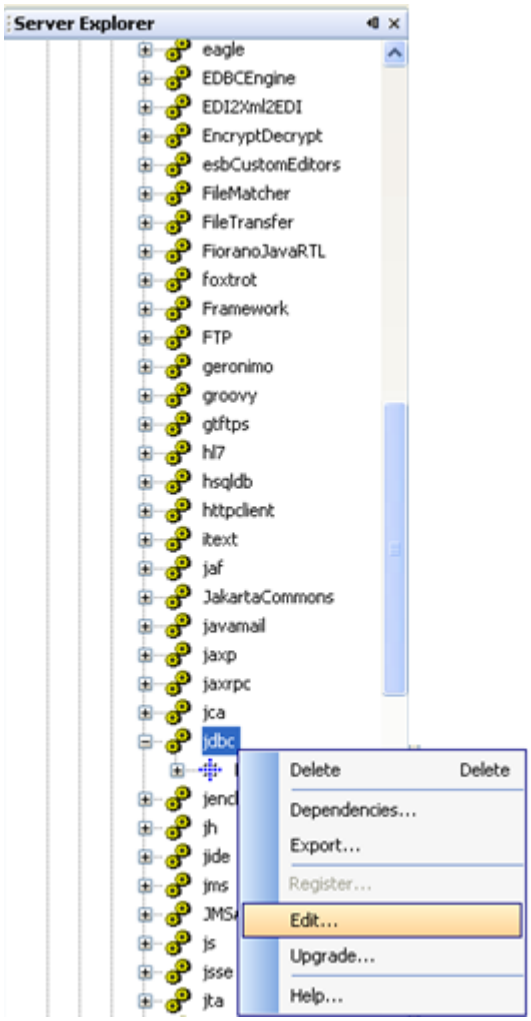


Figure 3.3.14: Right-click jdbc to customize service

2. Right-click jdbc and click on **Edit** button from the pop-up menu. The Customize **Edit Service jdbc:4.0** dialog box appears.

3. In the **Customize Edit Service jdbc:4.0** dialog box, navigate to Deployment -> Resources, now right-click on **Resources** and click on **Add Resources**. The **Add Resources** dialog box appears.

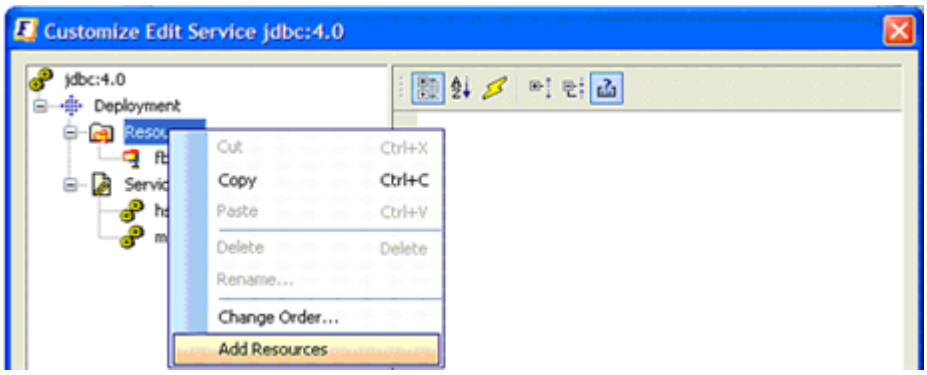


Figure 3.3.15: Customize Edit Service jdbc

4. In the **Add Resources** dialog box, browse to location containing jar files and select required files, now click the **Open** button.

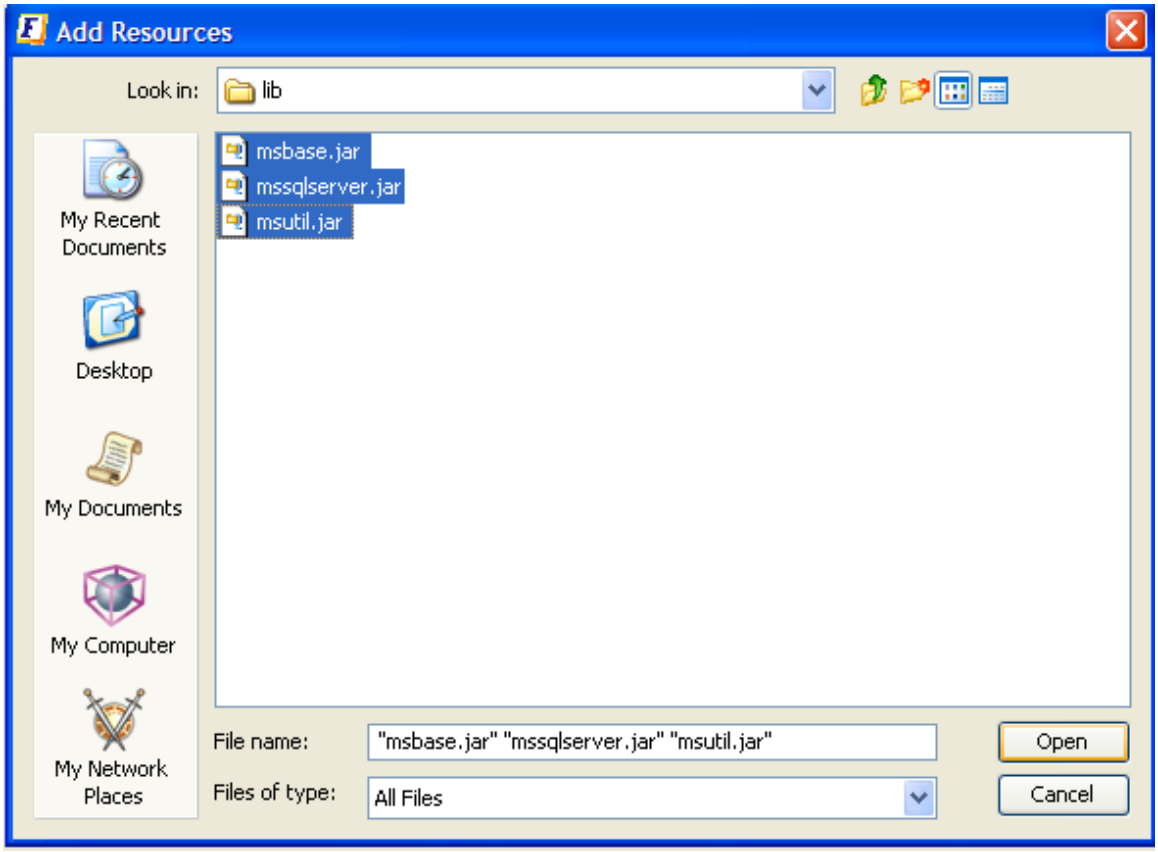


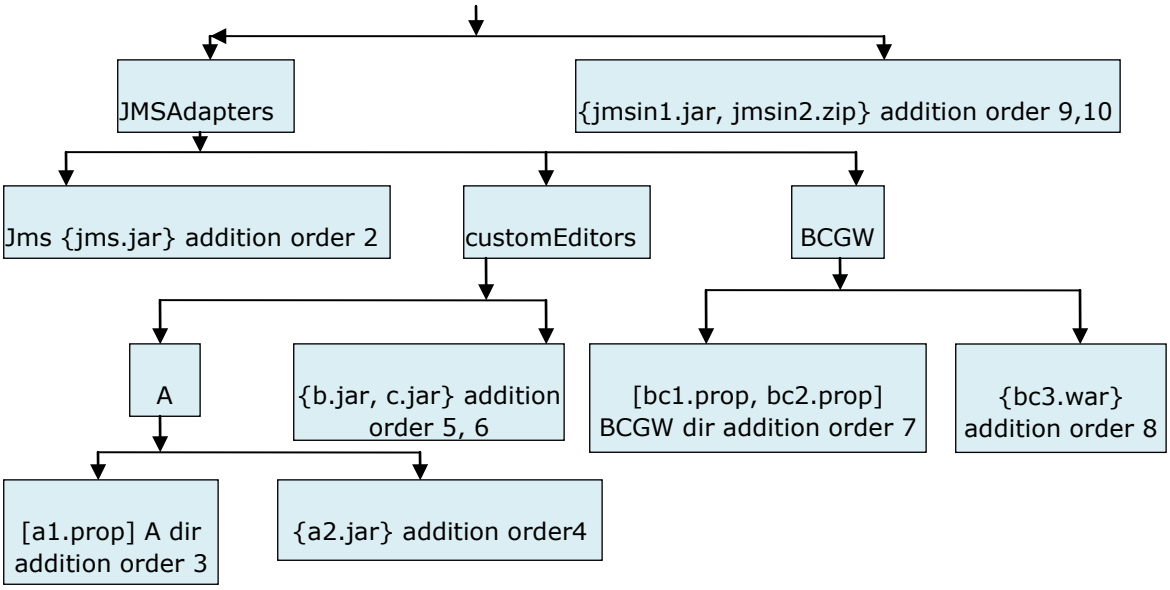
Figure 3.3.16: Add Resources

5. Now, close all the open dialog boxes. The new resources are added to the class path.

3.3.3.4 Component Classpath Generation

1. Add Component_Home directory, a Depth First Traversal of the dependency tree of component is carried out.
2. All resources of the leaf node of dependency tree added in the following fashion:
    a. If the dependency directory contains any required resource other than java archive such as jar/zip, the directory is added; then, all available java archive files are added
    b. At the end, the fmq client jar, the licenses dir, the xml-catalog dir and the server profiles dir is added.

Example:



.../fmq-rtl.jar (addition order 11)

.../licenses (addition order 12)

.../xml-catalog (addition order 13)

.../esb/server/profiles (addition order 14)

Legend:

1. {} – enclosed elements are all java archives
2. [] – enclosed elements are all non-java archives like .properties file etc.

Note:

1. The order of dependencies is as shown in Service Editor in Studio.
2. If the component needs some library and the peer server has some version of said library already present in its classpath, the Component should be run in separate Process to prevent class loader issues with peer.

3. If multiple version of some library is being added in the component classpath, as per java classpath rules, the version appearing earlier would be used for finding any required class.

3.3.4 Creating New System Libraries

A new library can be added using the Fiorano Studio tool, using the **Add Service** option as illustrated in Figure 3.3.17. Provide the details as mentioned in the wizard and click **Finish** to create a new system library.

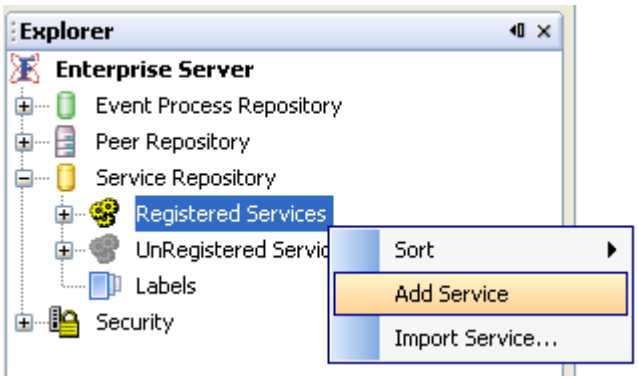


Figure 3.3.17: FSSM tool menu button showing the "Add Service" option in Explorer view

3.3.4.1 Adding a New System Library

1. Add a New Service as discussed in section 3.3.3.3 Adding Resources to Class Path.
2. Provide a name to the library in the **Service GUID** field of the **Customize New Service** dialog, and select the **Type** as **Library** as shown Figure 3.3.18.

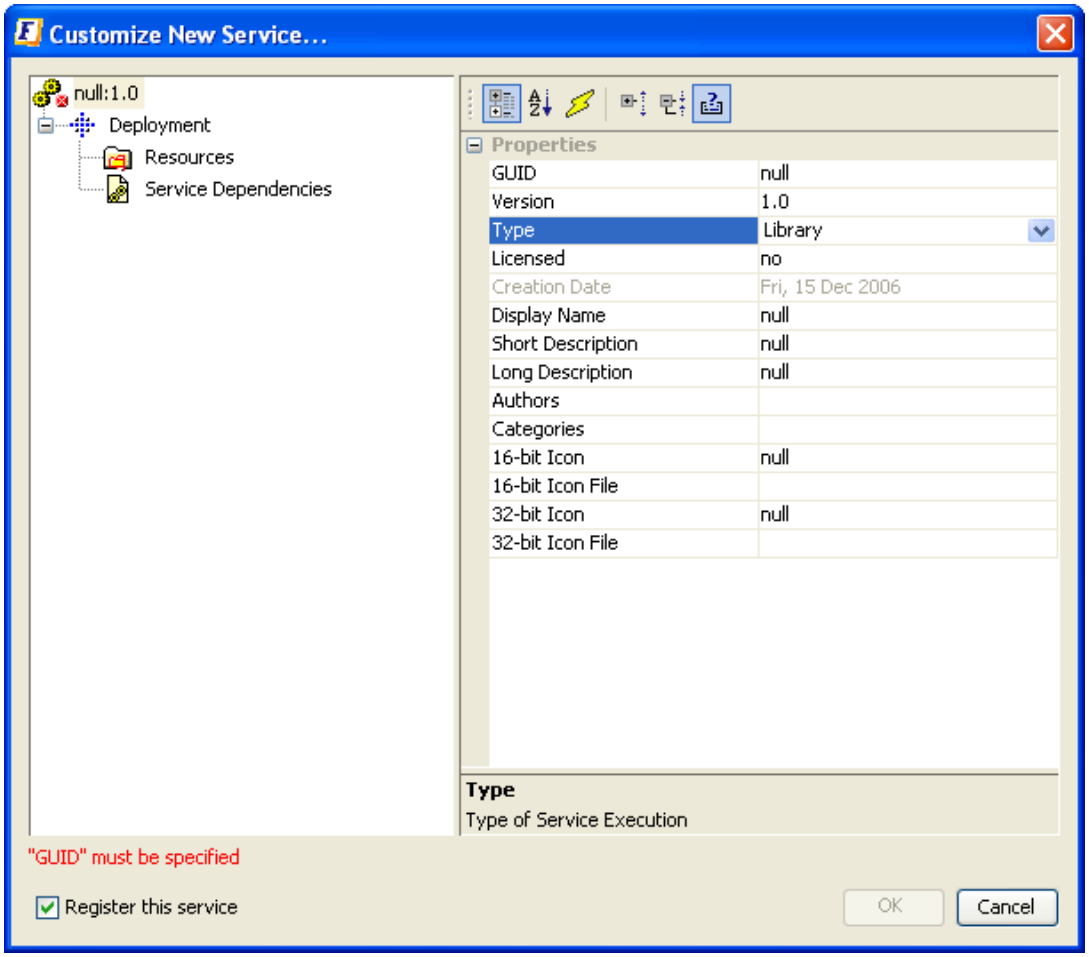


Figure 3.3.18: Creating a new system library

3. Right-click on **Resources** and select **Add Resources** from the menu list. Add library files which comprise this system library as shown in Figure 3.3.19.

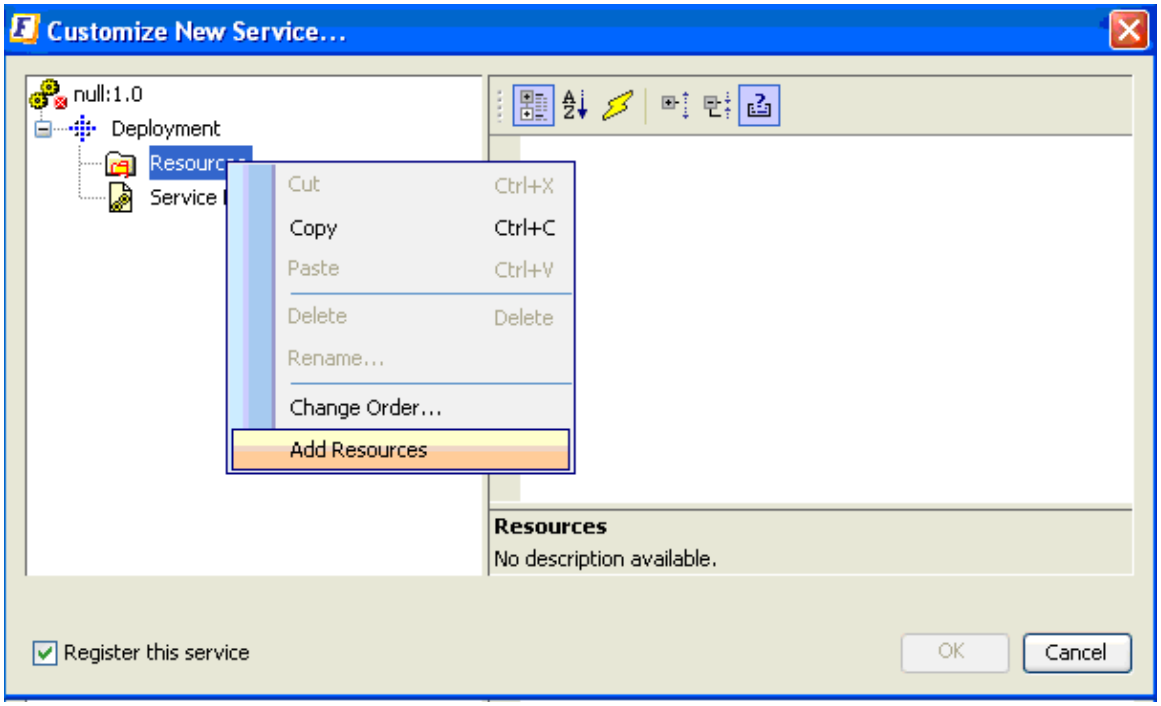


Figure 3.3.19: Menu option to add files as resources

4. After reviewing the details, click the **OK** button in the panel illustrated in Figure 3.3.20.

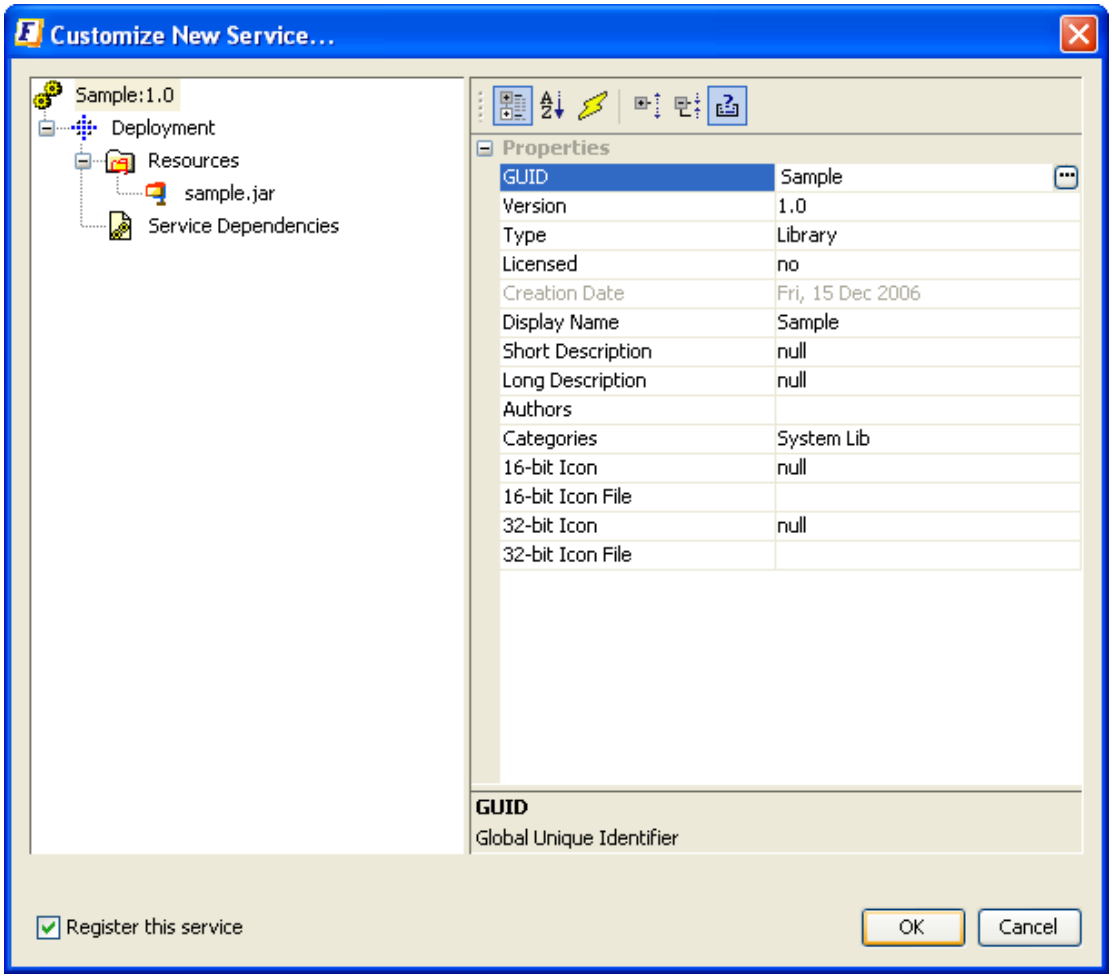


Figure 3.3.20:.Properties of the new library

The current service repository contains a category called System Lib which has all the system libraries registered with the Fiorano Enterprise Server.

3.3.5 Configuring Logging Parameters

For every service component, the log settings can be configured in the Properties window. For a chosen service instance the log settings are available under the categories **Log Manager** and **Log Module Instances** as shown in Figure 3.3.26.

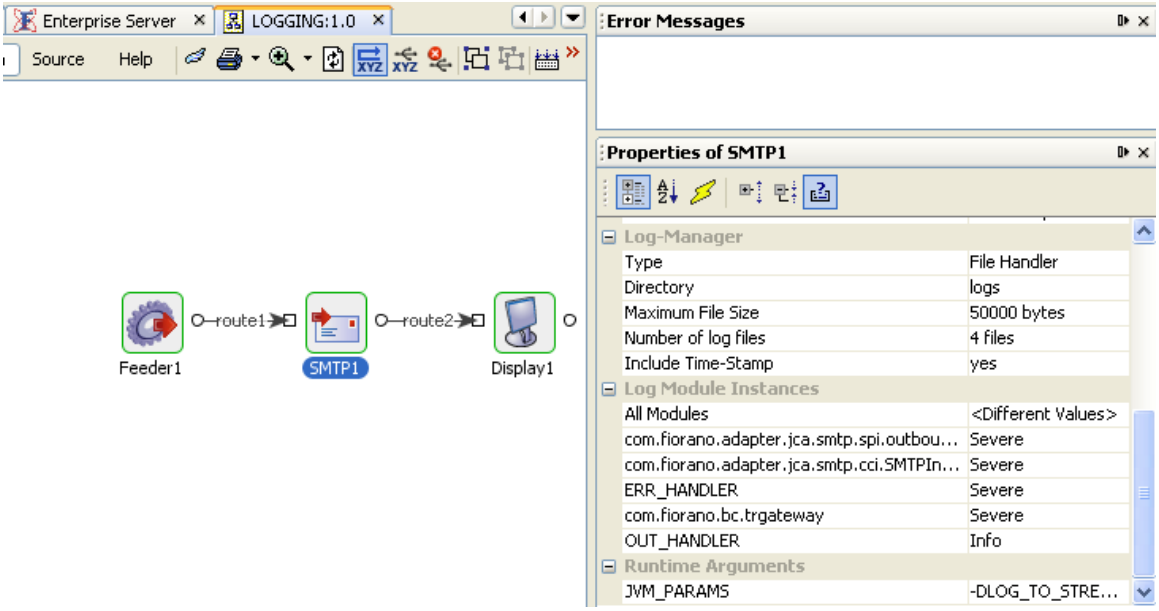


Figure 3.3.26: Log Module Instances

For every log module, the log level can be configured from the drop down list as shown in Figure 3.3.27. The different log levels at which the details can be logged are **Severe, Warning, Info, Config, Fine, Finer, Finest** and **All.**

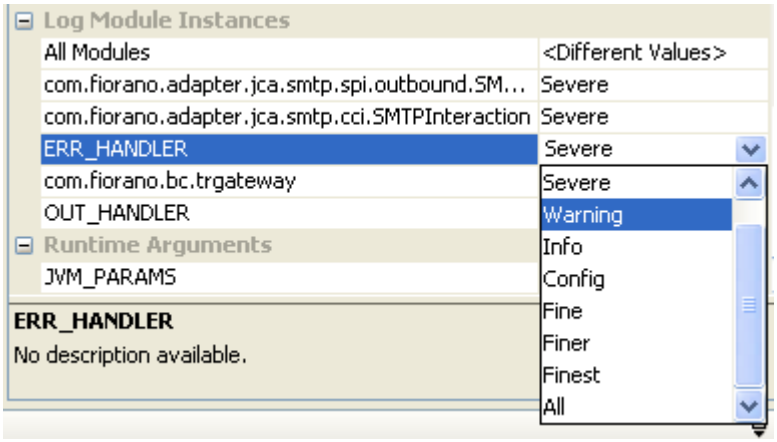


Figure 3.3.27: Log levels

Severe is the highest logging level and **All** the lowest. A log module accepts all messages that are logged at the levels greater than or equal to the configured level. That is, if the configured log level for a log module is **Severe**, only the messages at **Severe** level is logged. If the specified level is **Info**, the messages at **Info**, **Warning** and **Severe** is logged. Specifying **All** logs the messages at all levels.

Note: Messages of all log modules logged at **Severe/Warning** level appear in Error Logs of the component. Messages at the remaining levels appear in Out Logs.

Log handlers can be configured in Log Manager as shown in Figure 3.3.28.

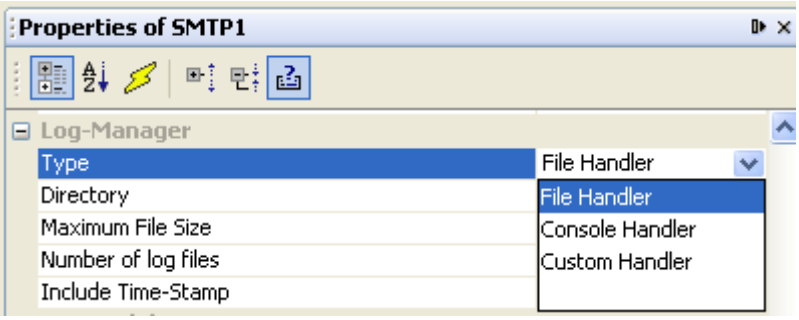


Figure 3.3.28: Configuring Log Handler

File Handler: When the log handler is a **File Handler**, the logs is written to files.

Example: When an event process **Logging** is created as shown in Figure 3.3.26, a directory **LOGGING** is created at <FIORANO_HOME>/runtimedata/PeerServers/<profile>/FPS/run/logs which in turn contains a folder for every service instance. These folders contain the files for out and error logs as shown in Figure 3.3.29.

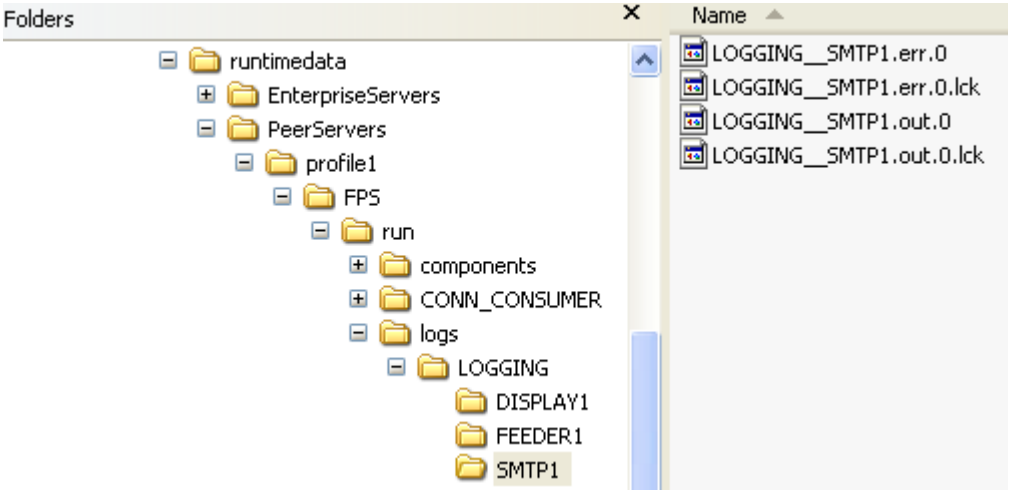


Figure 3.3.29: Log files

Console Handler: When the log handler is a **Console Handler**, the messages is logged on the console of the peer server on which the component is running. Figure 3.3.31 shows the logs on the peer console when the event process **Logging** (shown in Figure 3.3.26) is launched with the SMTP component configured with the log settings shown in Figure 3.3.30.

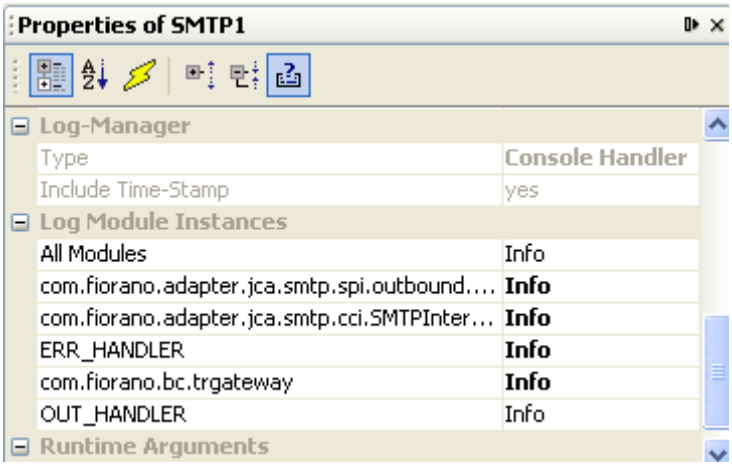


Figure 3.3.30: Log settings

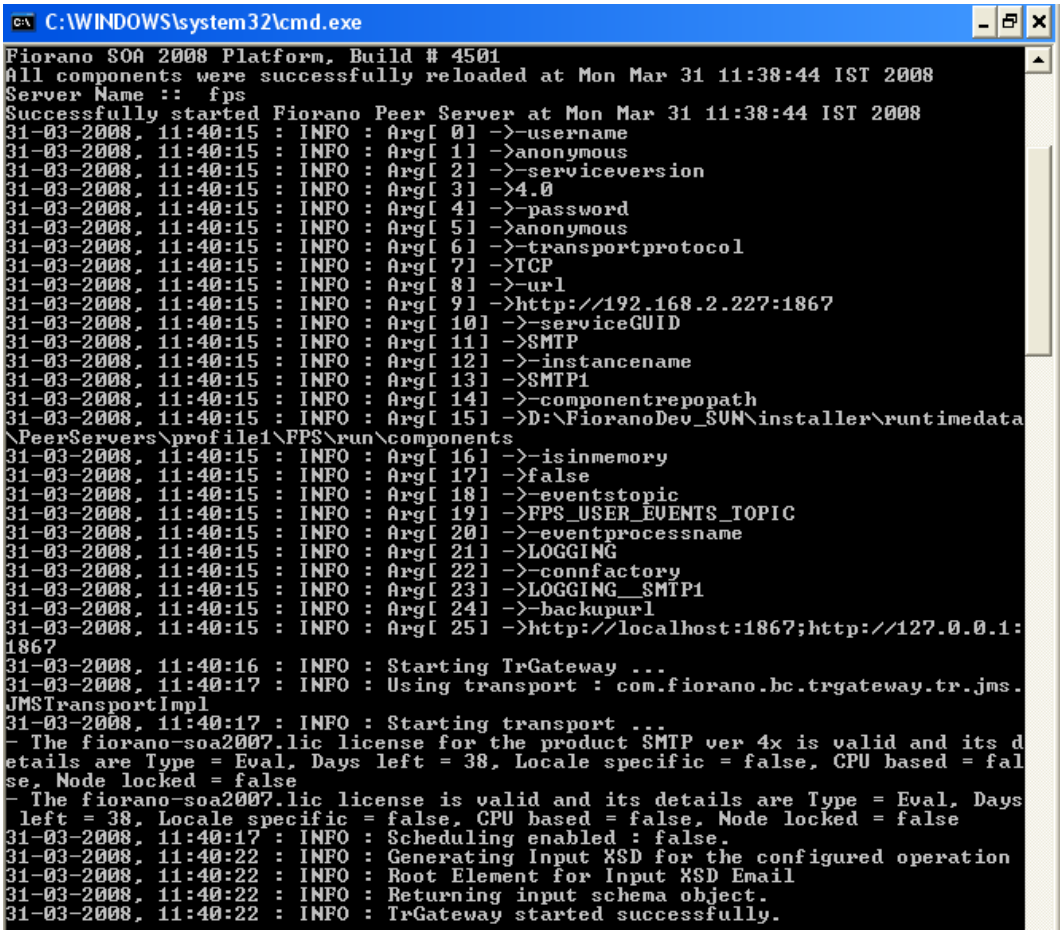


Figure 3.3.31: Logs on the peer server console

Custom Handler

In Fiorano SOA, Service instances can be configured for custom handlers also. Custom Log handler allows to redirect the log messages to user defined location like file, console, JMS destination, and so on.

To add the handler for the service instances that should use the custom handler,

1. In the properties pane, select **Log Manager** → **Type** as **Custom Handler**.
2. Specify the class for **Log Manager** → **Class Name** property. This should be a fully qualified class name.
3. Save the application.

Custom Handler implementation

The custom handler should be an instance of java.util.logging.Handler and should implement the abstract methods:

- public void publish(LogRecord record)
- public void flush()
- public void close()

This class should be made available to the peer server on which the component runs by adding appropriate classpath entry under <java.classpath> section of %FIORANO_HOME%\esb\fps\bin\fps.conf.

3.4 Component Deployment

All service components need to be registered with the Fiorano Enterprise Server to be used in a Fiorano application. The process of registration can be done using the Fiorano Studio too as discussed below. The components created using the wizard (in the Fiorano Studio tool), can be deployed through scripts available when the component is created. New component deployment (and registration) is covered in the Component Creation section.

A component needs to be compiled before being registered with the Fiorano Enterprise Server. For components written in the Java programming language, a JAR file needs to be created from the class files. If the component is created in an alternate language like C, C++, C# etc, an executable must be created instead.

Use the **Customize New Service** UI to register a new component. This UI is same the UI which starts when we create / register a new system library. Select the Java or Non-Java option in the Type parameter for the appropriate component type. This is illustrated in Figure 3.4.1.

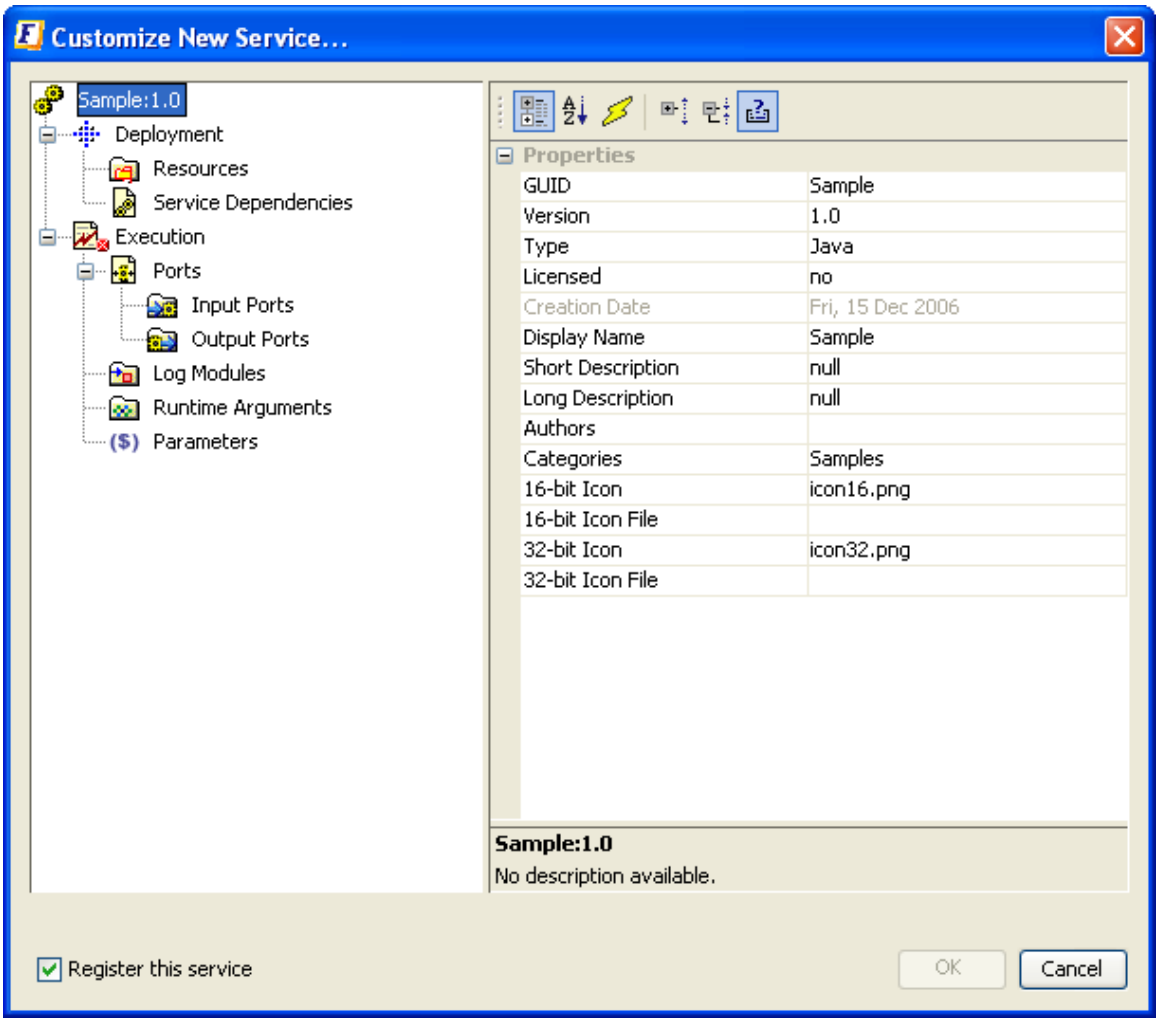


Figure 3.4.1: Customize New Service wizard for a service component

The deployment information for the new component is displayed, as illustrated in Figure 3.4.2.

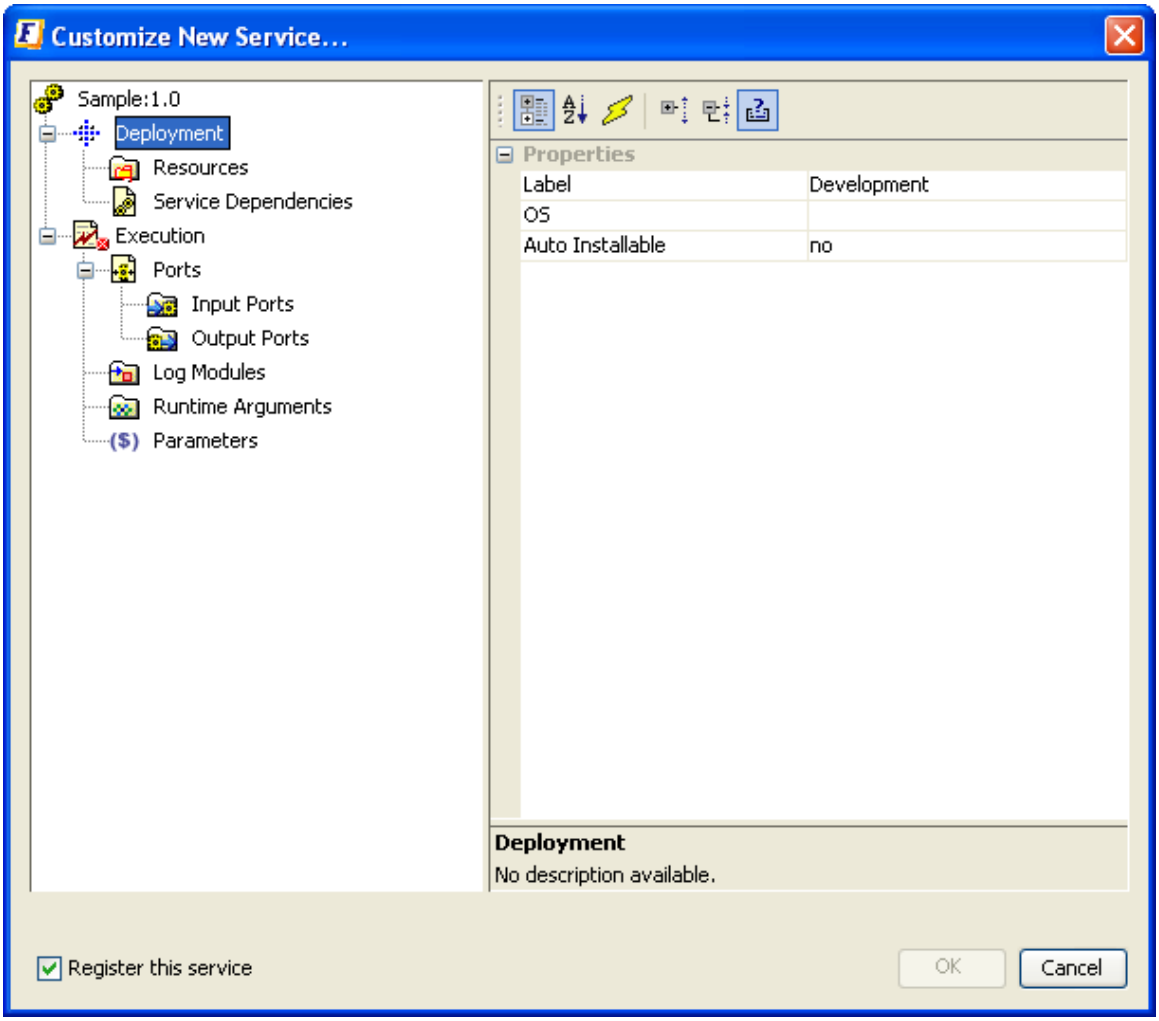


Figure 3.4.2: Shows the deployment information

The UI has subsections **Resources** and **Service Dependencies**, which are discussed in the section 3.3.3 Adding New Library Dependencies.

Clicking on the **Execution** icon on the Left-hand-side pane in the **Customize New Service** dialog displays (in the right hand pane) the list of execution parameters that can be set for a component. This includes the Launch-mode types (In-memory, Manual, Auto), as well as the name of the Executable in the Executable property. For components built in Java programming language, the executable name is the fully qualified class name. For Non-Java components, it is the name of the executable file.

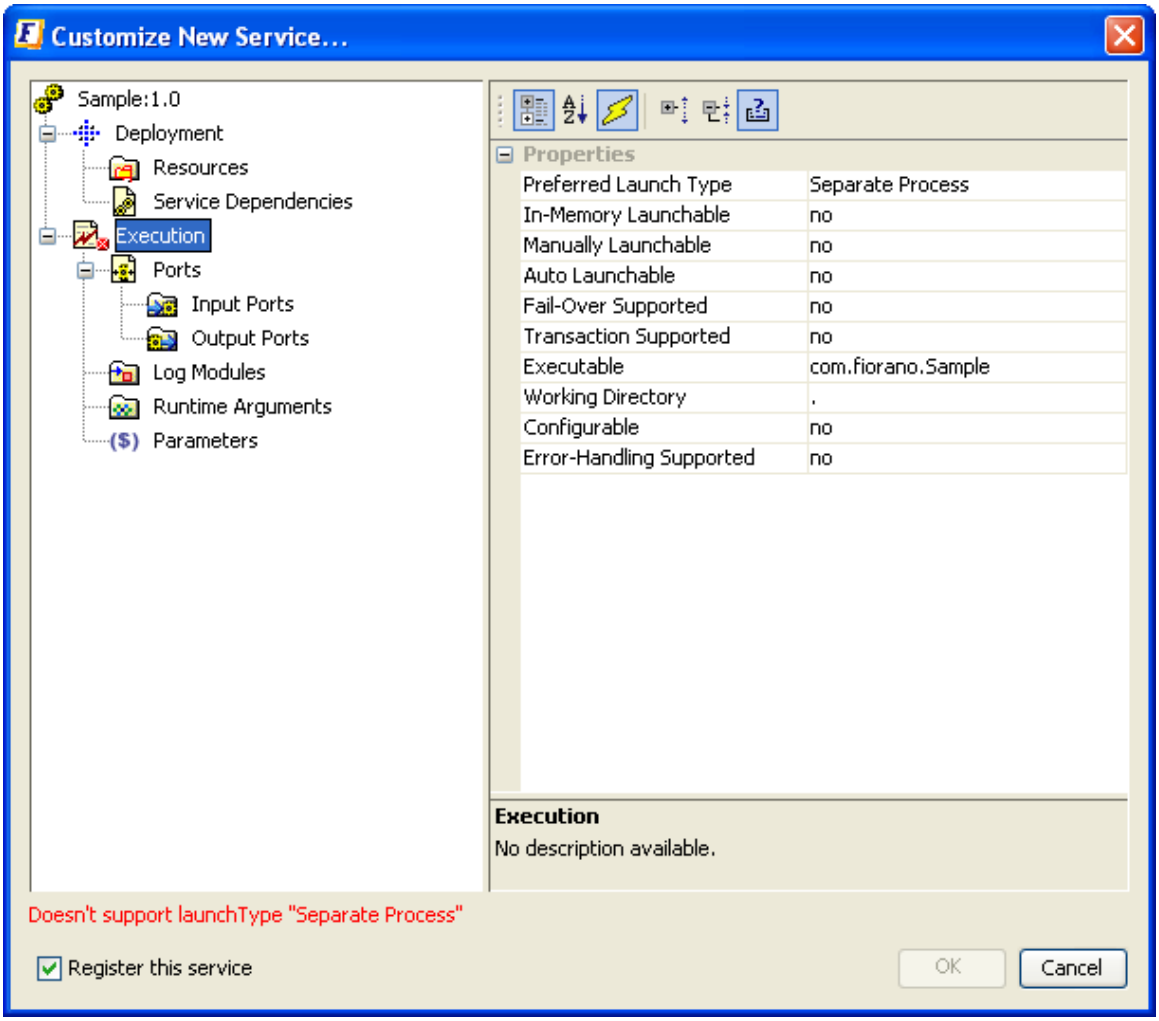


Figure 3.4.3: Execution information for new component registration

Each Asynchronous Service Component (also referred to as an EDBC, for Event Driven Business Component) can have a number of inputs and outputs as determined by the developer of the component. Input and output ports can be added in the Customize New Service dialog as applicable to the new component.

3.4.1 Adding Ports for the Component

1. Right-click on the **Execution**->**Ports**->**Input Ports** option in the left tree menu, and select the **Add Ports** option from the menu list as illustrated in Figure 3.4.4.

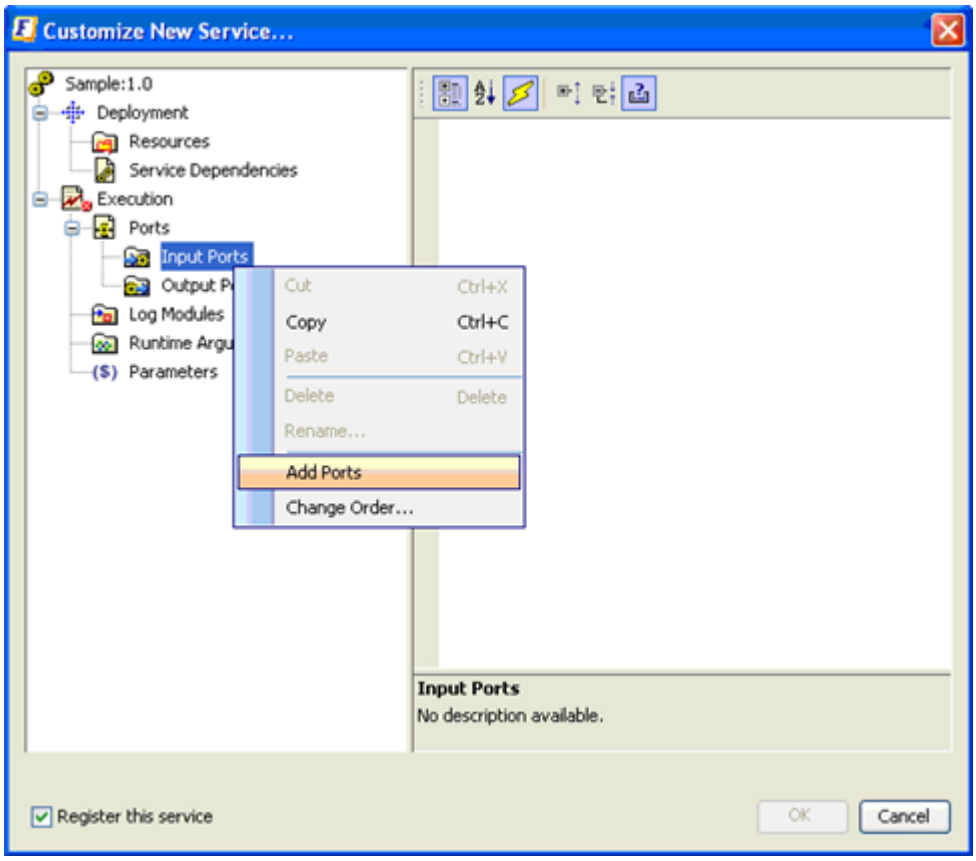


Figure 3.4.4: Adding ports

2. Specify the name of the port being added as shown in Figure 3.4.5.

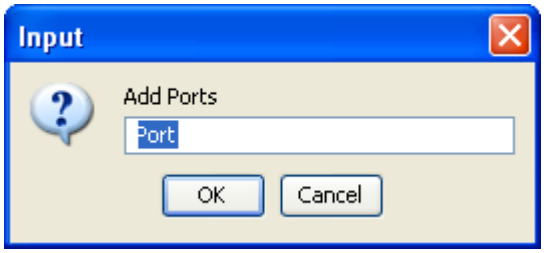


Figure 3.4.5: UI to provide port name

3. This is shown in the Figure 3.4.6. Specify the port properties as required by selecting the Port name in the UI.

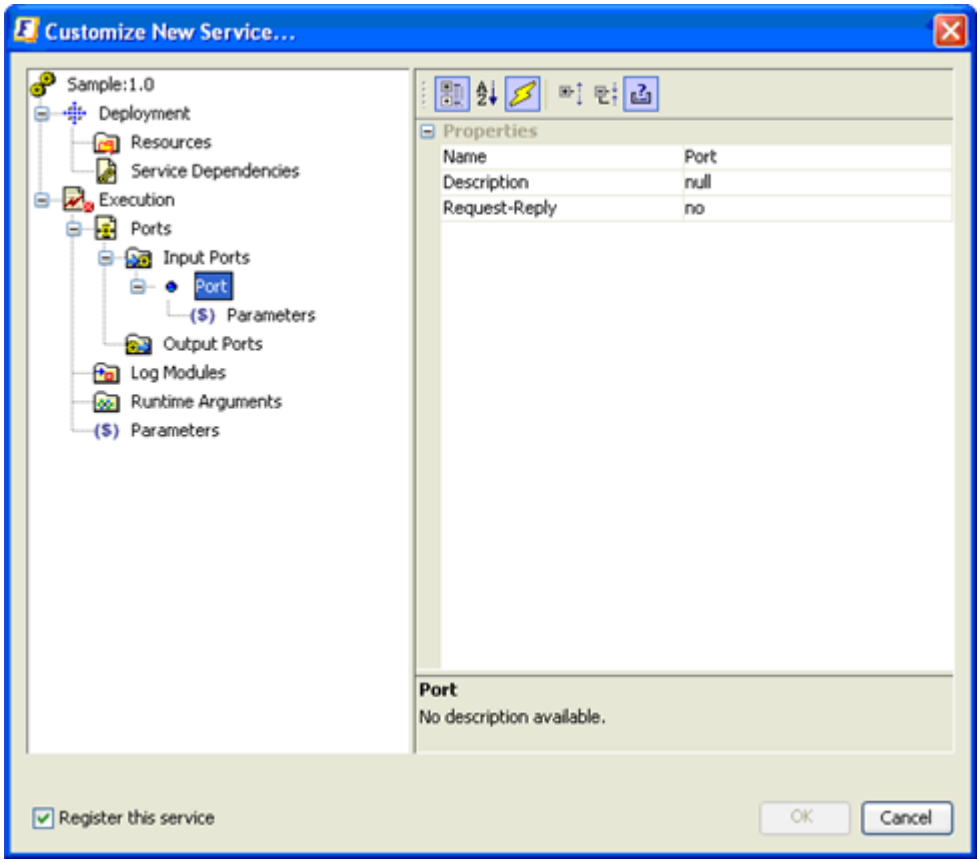


Figure 3.4.6: Modify the port properties required

3.4.2 Adding Log Modules for the Component

1. Right-click on the **Log Modules** in the left tree menu and select the **Add Log Modules** option from the menu list, as shown in Figure 3.4.7.

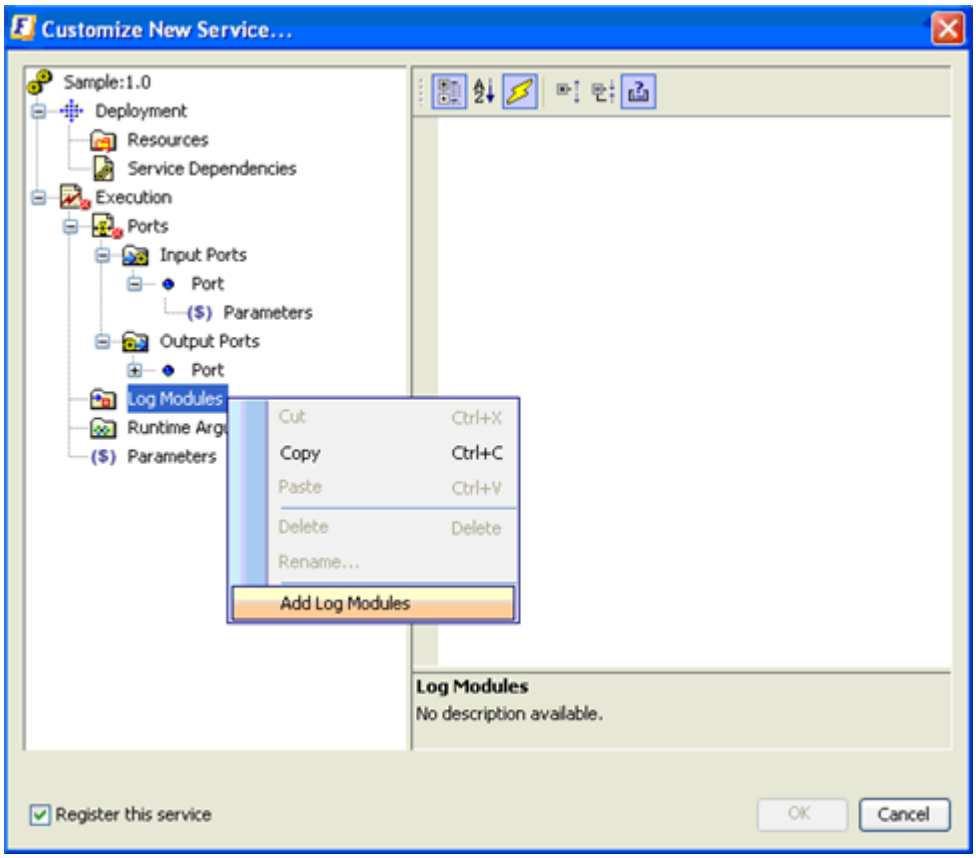


Figure 3.4.7: Menu to add log modules

2. Provide the appropriate log module name as used in the code as shown in Figure 3.4.8. The logger names provided here must be the same as the ones being used in the code.

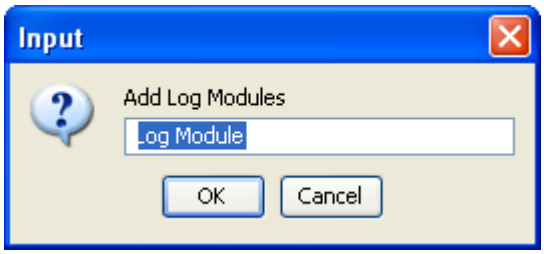


Figure 3.4.8: UI to provide log module name

3.4.3 Adding Runtime Arguments for the Component

1. Right-click on Runtime Arguments in the left tree menu and select **Add Runtime Arguments** from the menu list as shown in Figure 3.4.9.

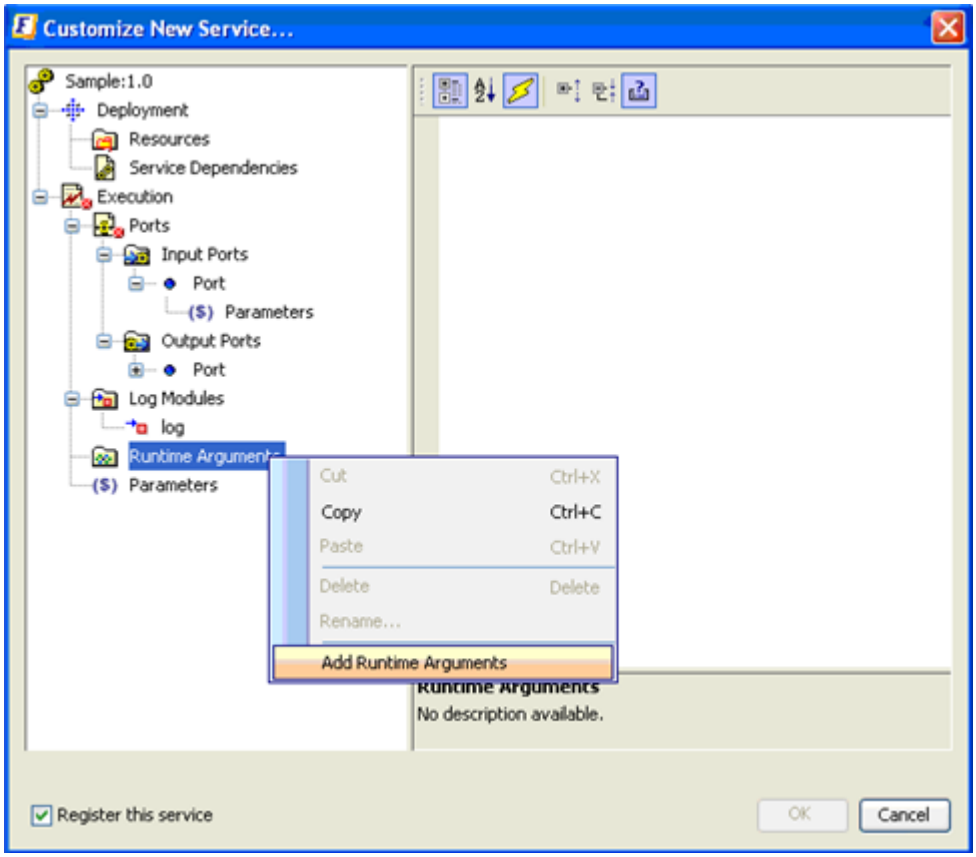


Figure 3.4.9: Menu to add new runtime arguments

2. Provide the name of the runtime argument in the **Add Runtime Arguments** text box. This should be the same name that is expected in the code.

Figure 3.4.10: UI to provide the name of the argument

3. Modify the details of the runtime argument by selecting it in the left-tree menu of the Customized New Service dialog as illustrated in Figure 3.4.11.

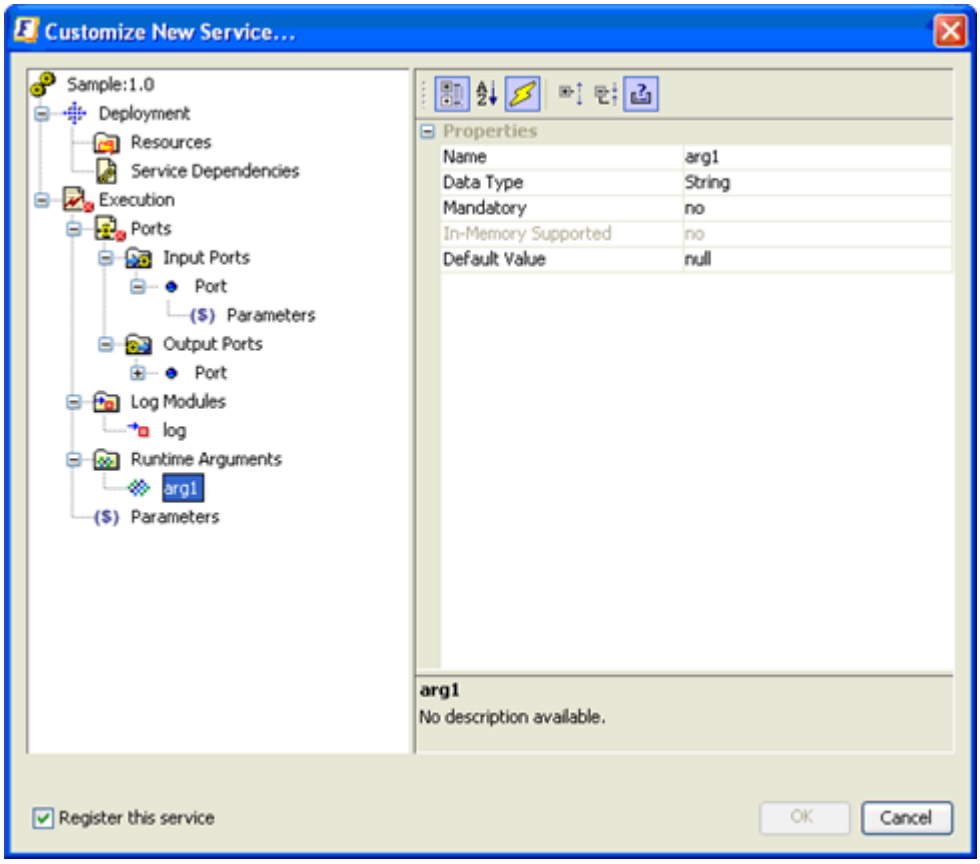


Figure 3.4.11: Modify details about of the argument added

3.4.4 Adding New Parameters to the Component

1. Right-click on **Parameters** in the left tree menu as shown in Figure 3.4.12 and select **Add Parameters** from the menu list.

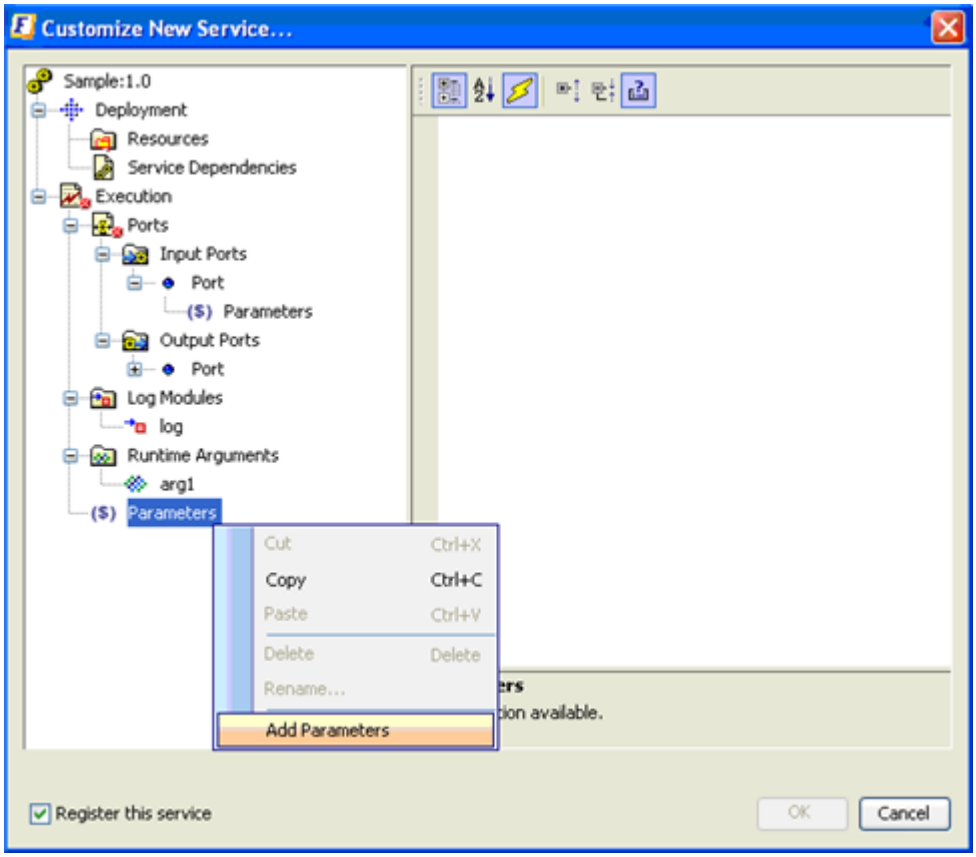


Figure 3.4.12: Menu to add additional parameters to execution

2. Provide the name of the parameter in the **Add Parameters** text box as shown in Figure 3.4.13.

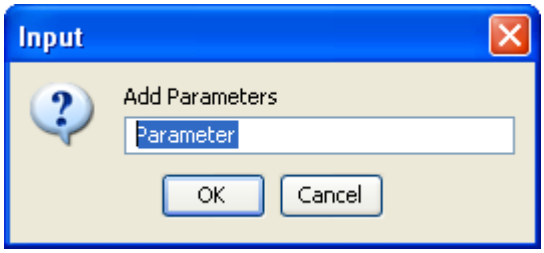


Figure 3.4.13: UI to provide the name of the parameter

3. Review the details and click the **OK** button to finish the UI for new component registration.

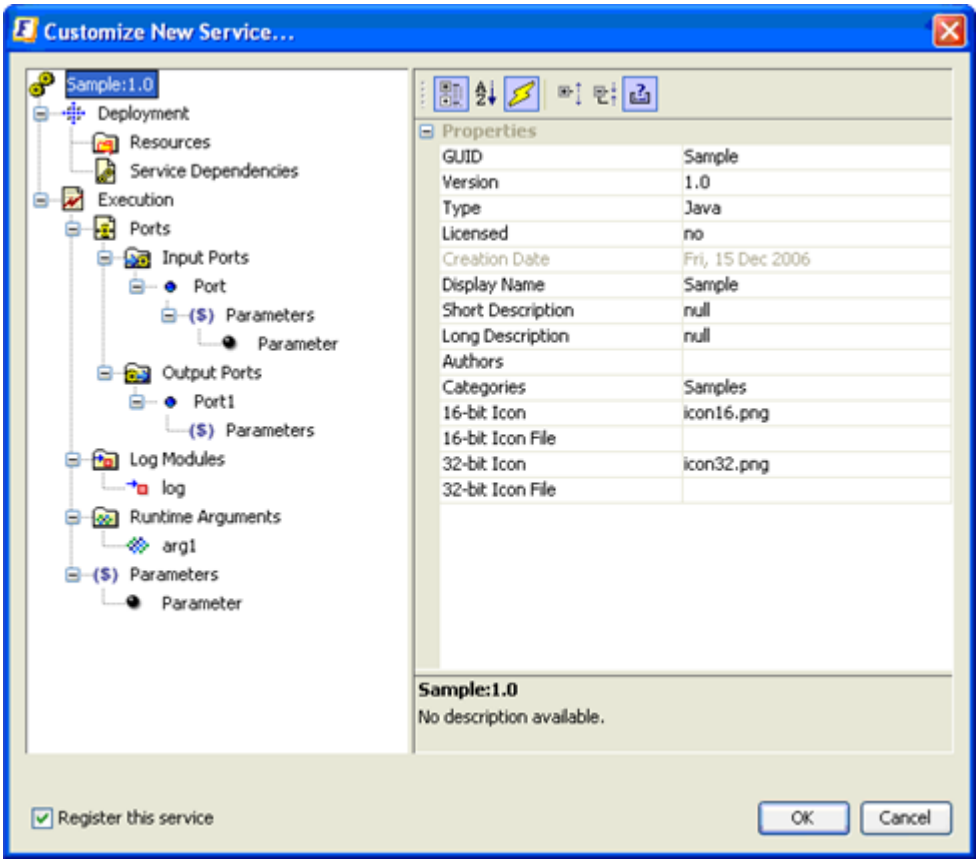


Figure 3.4.14: Review changes below clicking "OK"

If the **Register this service** checkbox is checked (as shown in Figure 3.4.14), then the component is registered with the Fiorano Enterprise Server and is immediately made available in the service palette for use within a Fiorano application.

A component used in a Fiorano Event Process needs to connect to a (local or remote) Fiorano Peer Server on launch. On most situations, the component is deployed onto a Fiorano Peer Server so it can be launched locally. The "Check Resources and Connectivity" and "Synchronize" functions in the Fiorano Studio automatically deploy components onto the named Fiorano peer server assigned to the component instance in the Fiorano Event Process. The Fiorano Peer Server on which a component needs to be launched is set using the "Nodes" property in the Properties window in the Fiorano Studio as shown below.

3.4.5 Adding Node Name to a Component Instance

1. Select the Component instance in the Fiorano Studio window and Click on the button in the **Nodes** property of a component as shown in Figure 3.4.15.

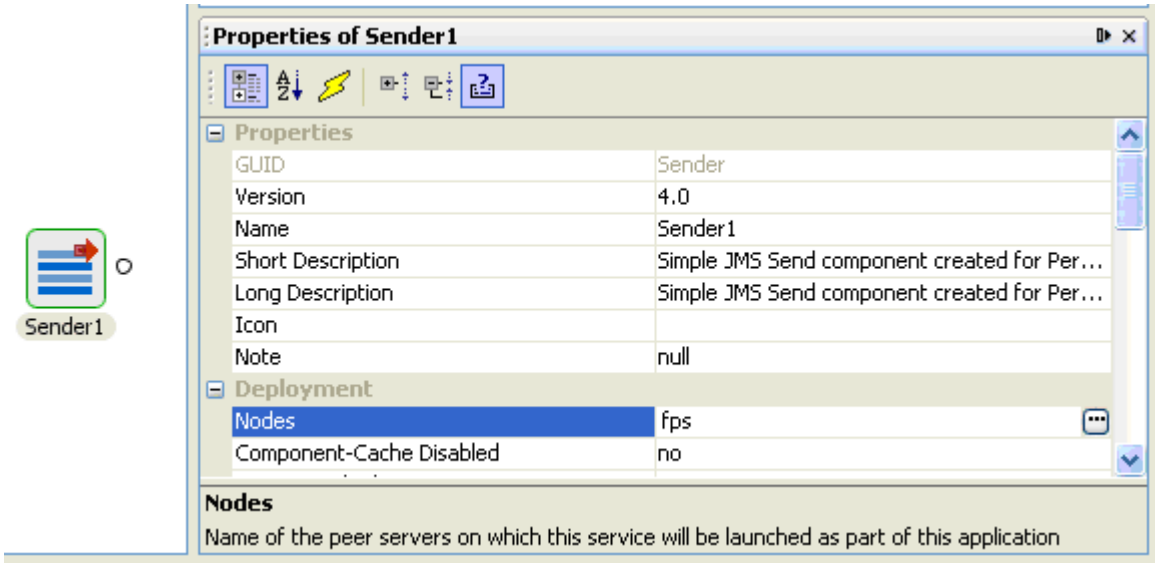


Figure 3.4.15: Nodes property for a component

2. Click on the **Add** button in the dialog that opens as shown in Figure 3.4.16. This displays a list of current peer servers available on the network; select the peer server on which the component is to and click the **OK** button. OS specific icons assist the user in identifying the Operating System on which the Fiorano Peer Server is running.

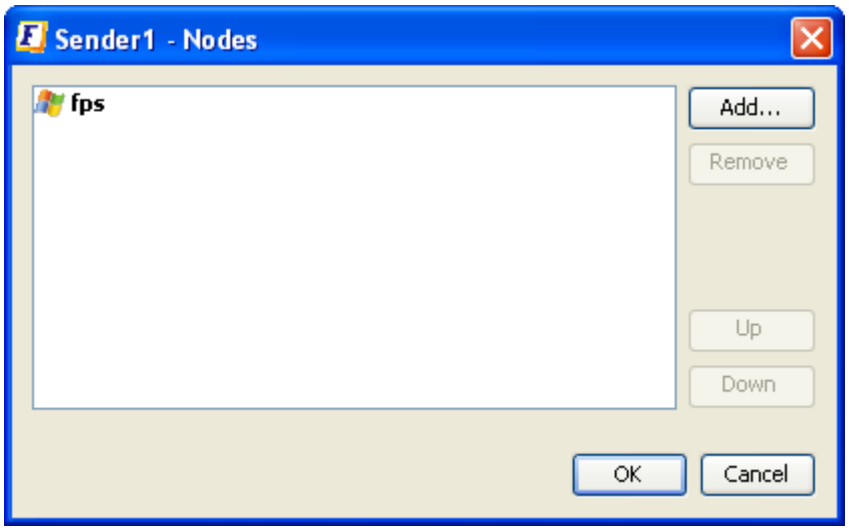


Figure 3.4.16: Fiorano Peer Server name on which the component will run

The Fiorano Studio menu includes the Check Resources and Connectivity and Synchronize features (as shown in Figure 3.4.17) to dynamically deploy the resources and dependencies of a component from the Fiorano Enterprise Server to the Fiorano Peer Server to launch and execute a component locally on the peer.

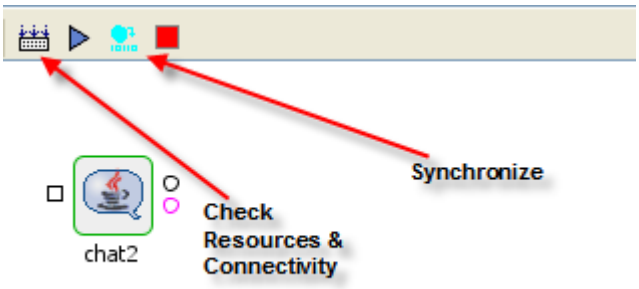


Figure 3.4.17: Fiorano Studio menu items for dynamic deployment

3.4.6 Manual Deployment

There are various ways in which a component can be configured and launched. One of supported methods is manual launch. A component's launch mode can be set to Manual as shown in Figure 3.2.2.

Manual launch is a mode in which the Fiorano peer server does not control the launch and stop of components. Fiorano provides two mechanisms for manual launch.

3.4.6.1 From Scriptgen Tool

Manual launch can be achieved via a launch script that can be generated via the Fiorano Studio.

3.4.6.1.1 To generate a manual launch script from the Fiorano Studio

1. Right-click on the component and select **Execution-> Save Launch Script** from the drop-down list as illustrated in Figure 3.4.18. These properties can be used to launch the component using the scriptgen tool available in <fiorano_install_dir>/esb/tools/scriptgen. Right-click to save the script.

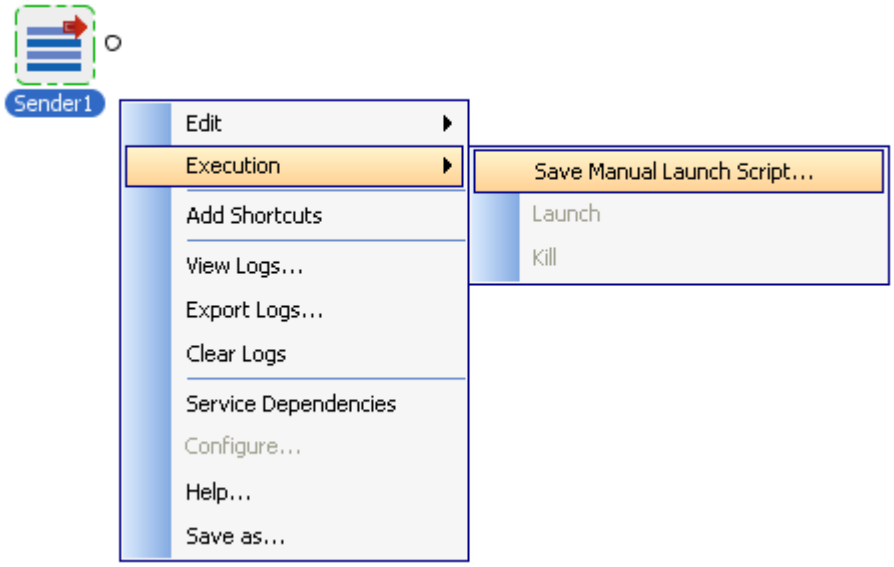


Figure 3.4.18: Menu to save the manual launch script using Fiorano Studio

2. Be sure to change the launch mode of the selected component to Manual in the Fiorano Studio as shown in Figure 3.2.2.
3. After saving the generated script, a dialog is shown with details on how to run the manual script as illustrated in Figure 3.4.19. Click the **OK** button.

Figure 3.4.19: Dialog showing the steps to execute a component using scriptgen

4. Executing the scriptgen.bat file (on windows) or scriptgen.sh file (UNIX platforms) sets the environment variables. Type the command **ant** and press enter to start the UI to execute the launch script saved above. This opens up an UI as shown in Figure 3.4.20.
5. Select the properties file and click on the **Load** button to set the appropriate properties. Click **Ok** to launch the component. Note that this Component-launch should only be done after the Event Process (within which the component is an instance) has been previously launched. Components that are marked "manual launch" will not be launched when the application is launched, since they need to be started manually, as described earlier.

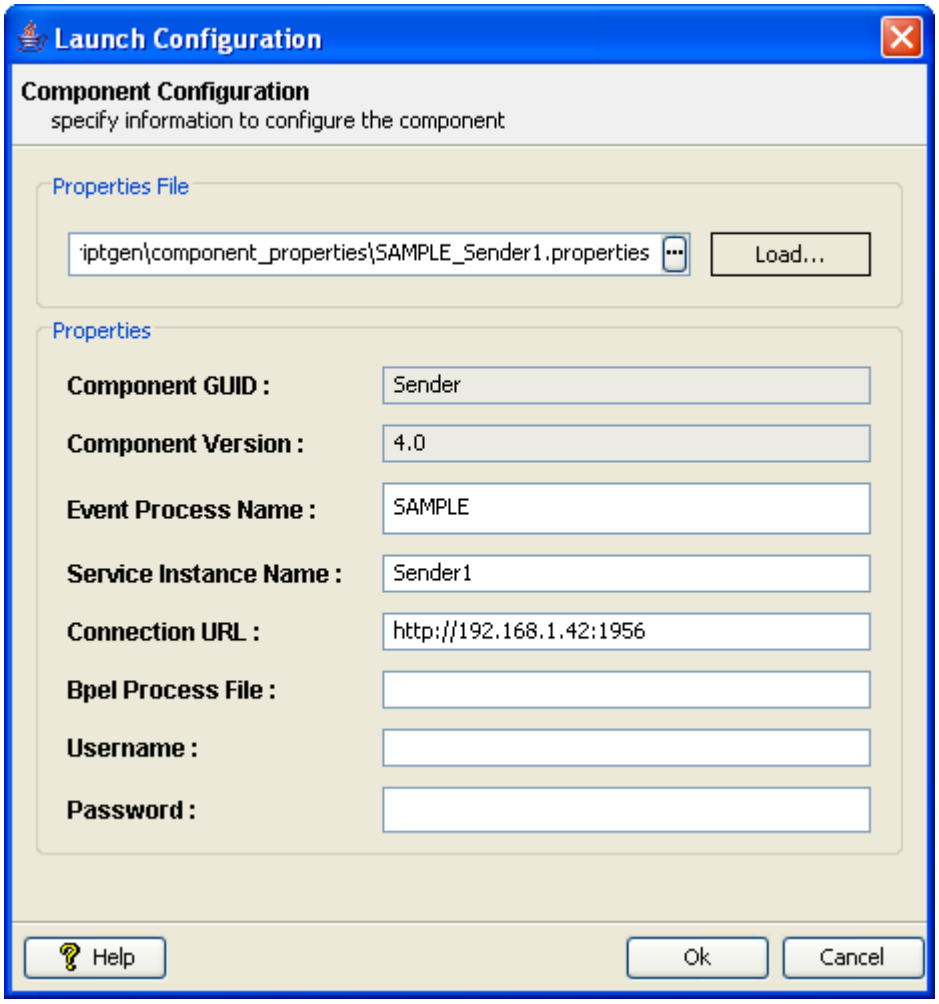


Figure 3.4.20: Scriptgen UI to load the component properties for manual launch

The console shows the progress of the launch of the component and the log statement during execution. Closing the console kills the component. A sample console with messages is shown in Figure 3.4.21.

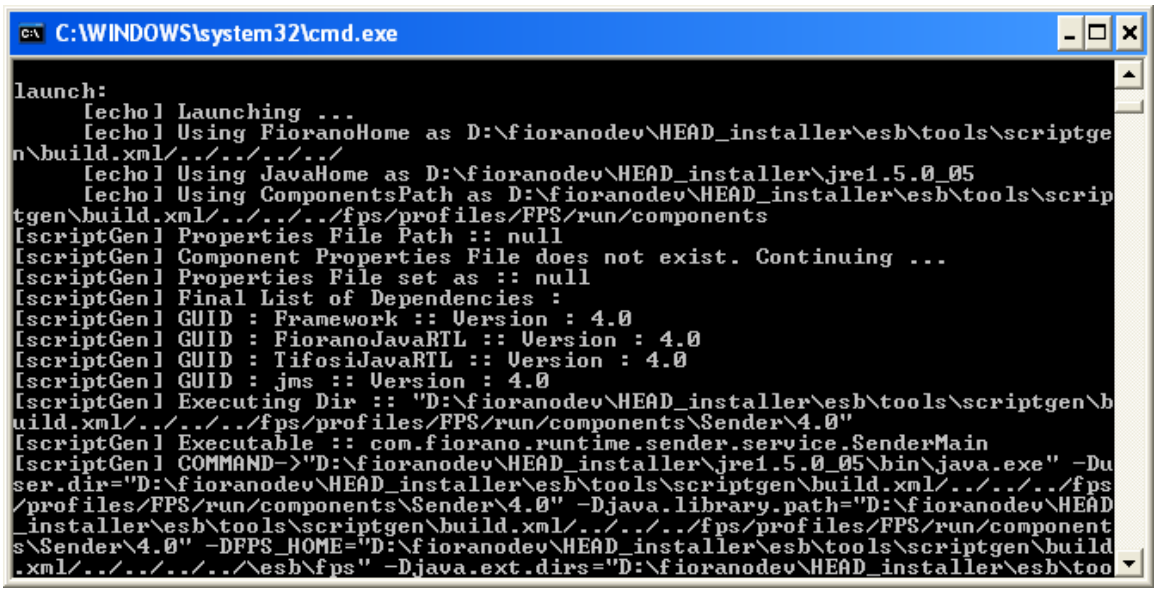


Figure 3.4.21: Console displaying the details of the launch of the component

3.4.6.2 From the configureBC and runBC utilities

Synchronous components (also refered to as **Business Components**) can be configured and executed using the configureBC and runBC utilities:

3.4.6.2.1 To configure and run Business Components

1. Navigate to the **<fiorano_install_dir>/bc/bin** directory as illustrated in Figure 3.4.22.

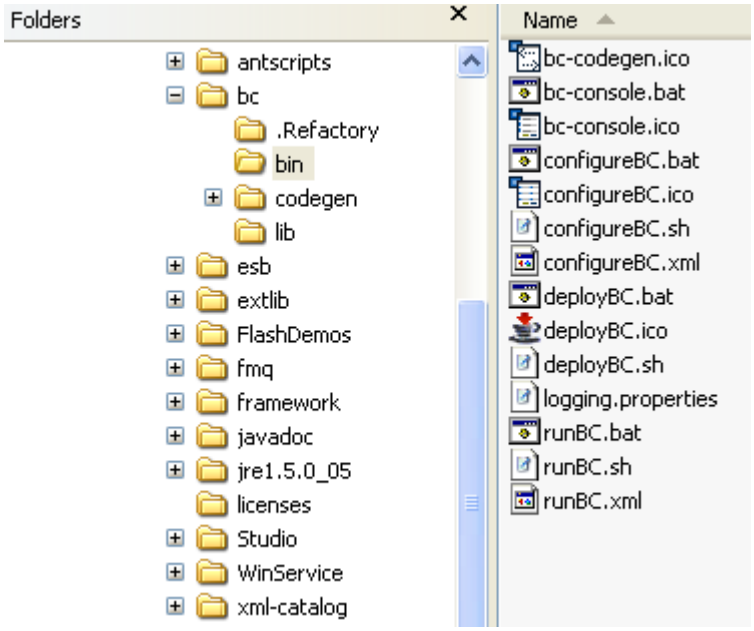


Figure 3.4.22: Location of configureBC and runBC script files

2. Execute the **BC.bat** file (for windows) or **BC.sh** (for UNIX and related platforms) to configure a component. The UI shown in Figure 3.4.23 is displayed. There is choice of creating a new configuration or opening an existing configuration for modification by changing the **Choose** parameter. Click the **Next** button.

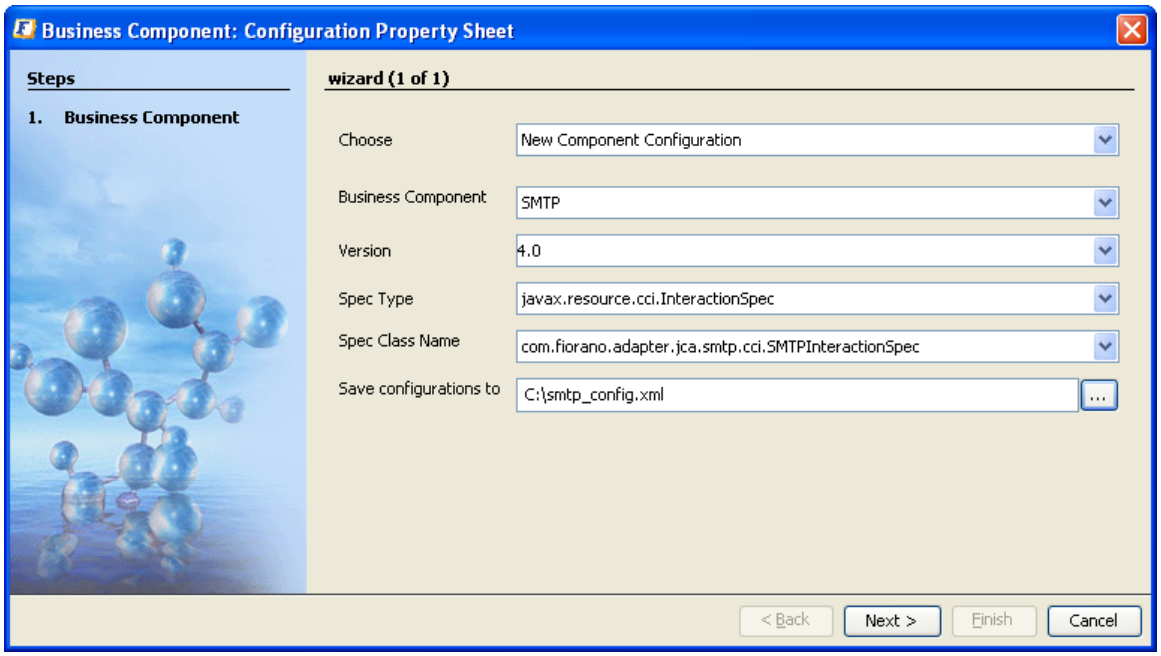


Figure 3.4.23: UI to configure a new component or load an existing configuration

3. Choose the appropriate business component and select the folder and XML file name to save the configuration. Click the **Next** button to display the Managed Connection Factory settings as shown in Figure 3.4.24. The wizard steps of the selected business component's CPS are shown.

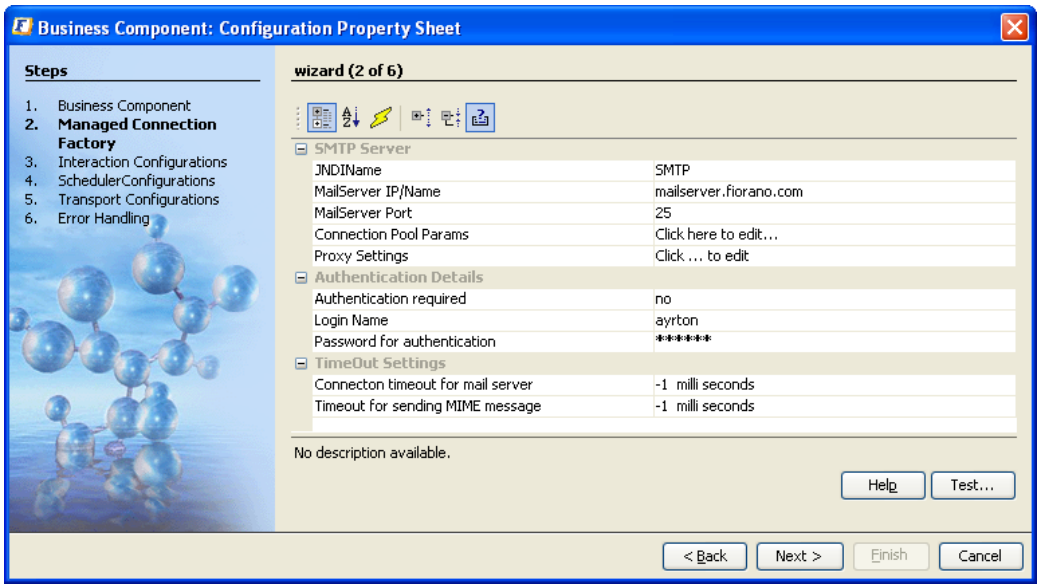


Figure 3.4.24: CPS of the component chosen are displayed

4. Move ahead in the wizard till you reach the **Transport Configurations** section which allows the component to send and receive messages with any JMS compliant messaging server or with the Tibco Rendezvous transport. Modify the Input Transport, Output Transport, and Error Transport sections to choose appropriate destination types and names. Also set appropriate configuration values to connect to these servers. This is illustrated in the Figure 3.4.25.

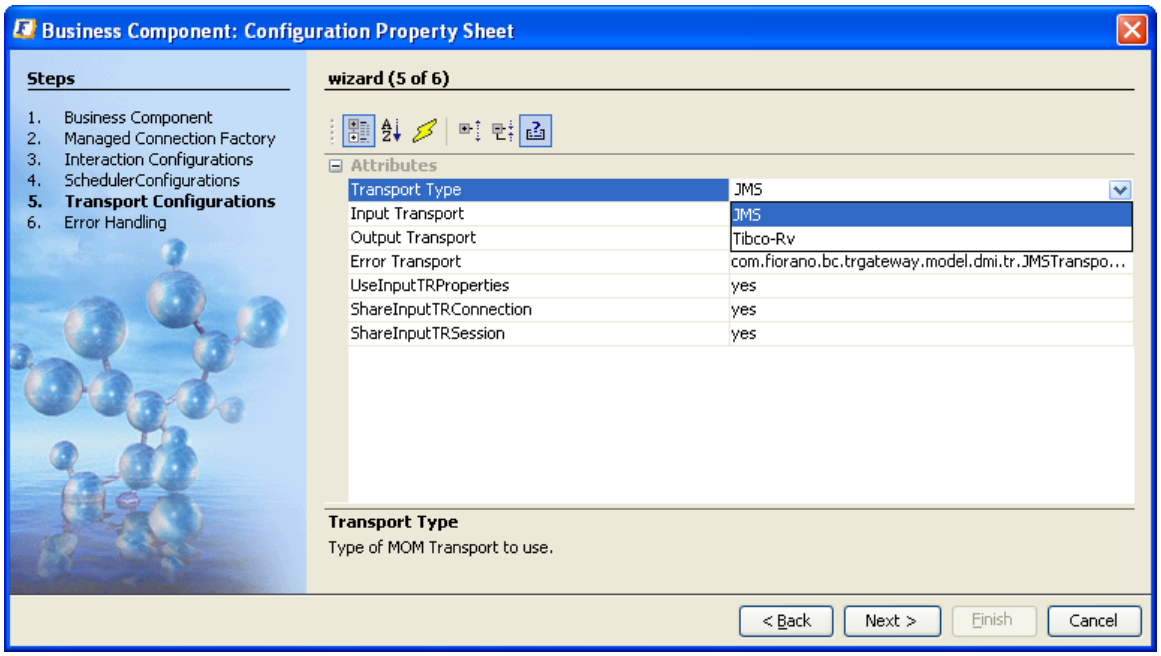


Figure 3.4.25: Transport properties for the component configuration

5. On completing the wizard, the XML file with contain the entire configuration for the Business Component. This configuration can now be used to run the specified component. Use the **runBC.bat** script (for windows) or **runBC.sh** script (for UNIX platforms) to execute the saved configuration, as illustrated in Figure 3.4.26.

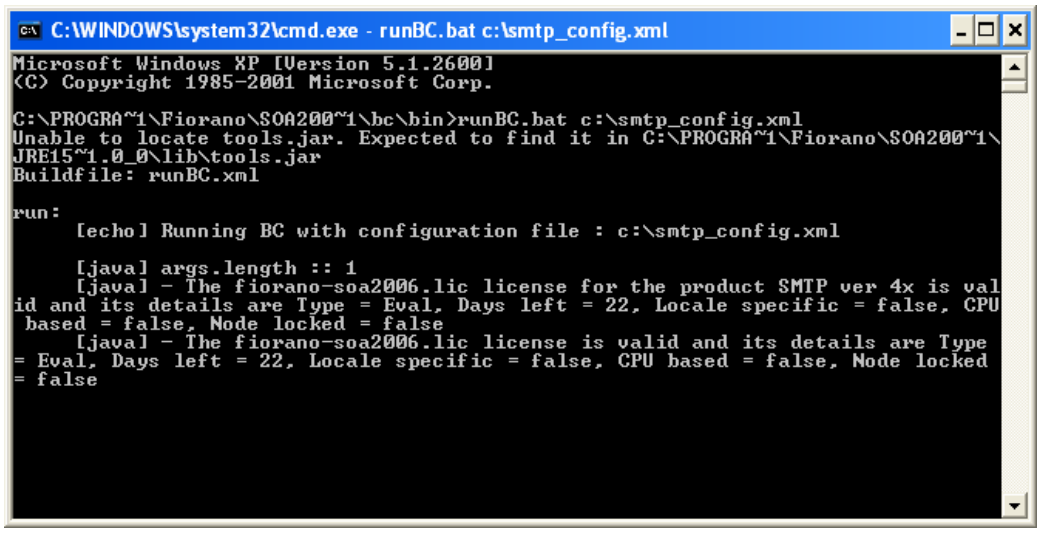


Figure 3.4.26: Command to run the saved configuration

6. Send a message to the configure destination in Input Transport section and verify for the response in the configured destination for Output Transport (if the request is valid) or the error in the configured destination for Error Transport (if the request is invalid or processing fails).

3.4.6.2.2 To modify a saved configuration XML file

1. Run the **configureBC** script as shown in Figure 3.4.27.

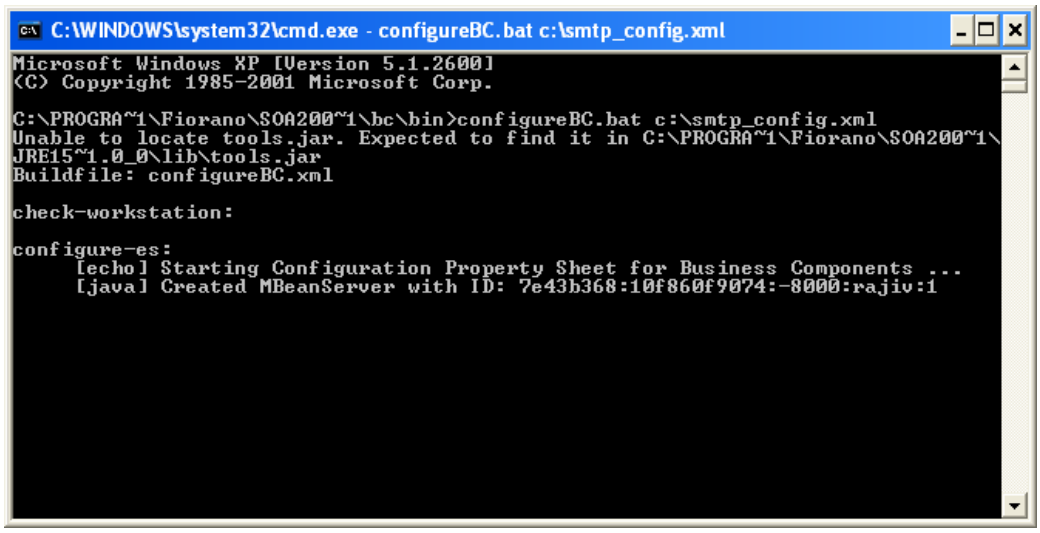


Figure 3.4.27: Command to modify an existing configuration

2. Alternatively, you can run the **configureBC** script file without the configuration XML file input and provide it through the UI as illustrated in Figure 3.4.28. Use the **Load Existing Configuration** option and choose the appropriate configuration file to modify it.

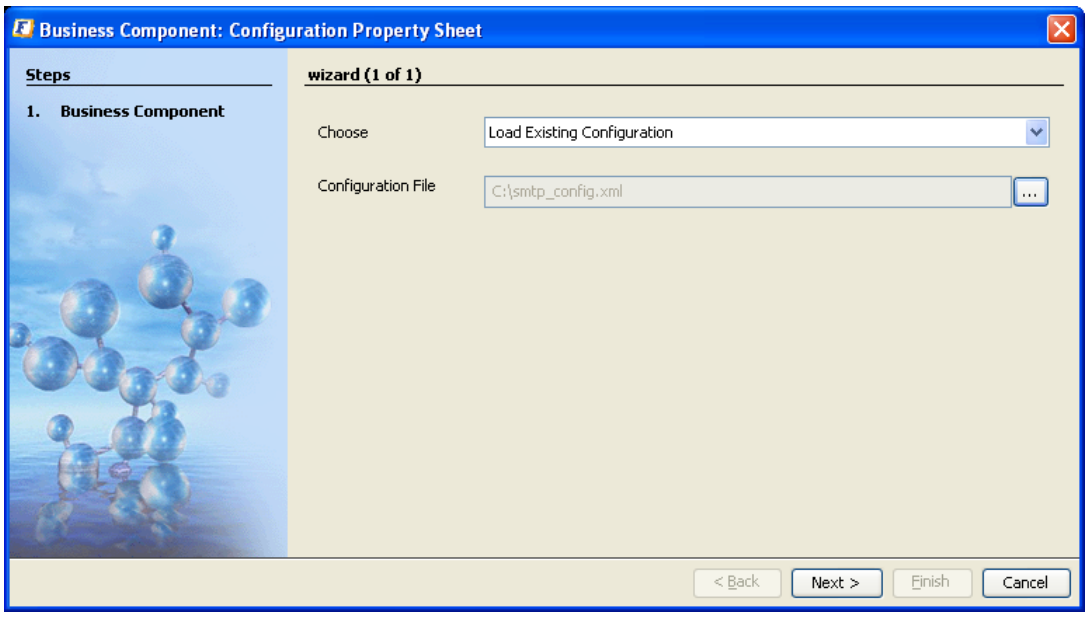


Figure 3.4.28: UI to load an existing configuration for modification

3.4.7 External Deployment

Synchronous components are fully compliant to the JCA version 1.0. Thus, they can be deployed in any J2EE JCA container which is compliant to JCA version 1.0. The following section describes how to deploy a synchronous component in a JBoss Application Server's JCA container.

3.4.7.1 Deploying a Synchronous Component in JBoss Application Server

Create a Resource Archive (RAR) of an adapter

The RAR file can be created using rar ant task. A sample build file that creates a RAR of an adapter is given below:

```
<project name="Rar" basedir="." default="createRar"
xmlns:fiorano="antlib:com.fiorano.ant">

        <property name="deploy.dir" value="D:/fioranodev_installer"/>

        <property name="component.guid" value="IWay"/>

        <property name="component.version" value="4.0"/>

        <property name="destDir" value="."/>

        <property name="rarFile" value="${component.guid}.rar"/>

        <property name="component.deploy.dir"
value="${deploy.dir}/esb/server/repository/components"/>

        <property name="comp.deploy.dir"
value="${component.deploy.dir}/${component.guid}/${component.version}"/>

        <property name="thirdparty" value="${deploy.dir}/extlib"/>



<target name="createRar">

        <fiorano:rar destfile="${rarFile}" destinationdir="${destDir}"
componentdir="${comp.deploy.dir}"

                componentrepositorypath="${component.deploy.dir}"
extlibspath="${thirdparty} duplicate="preserve"/>

</target>



<target name="cleanup">
```

```
        <delete file="${rarFile}"/>

</target>



</project>
```

Build file to create a RAR file of an adapter

The build file can be executed using command ant from the command prompt after making necessary modifications (see below)

- deploy.dir – FIORANO_HOME
- component.guid – Name of the component (case sensitive)
- component.version – Version no of the component
- destDir – Directory where the rar file needs to be created
- rarFile – Name of the Rar file

  Example: To set the destDir value to C:/Temp modify the build file as follows

  Before modification

  ```
  <property name="destDir" value="."/>
  ```

  After modification

  ```
  <property name="destDir" value="C:/Temp"/>
  ```

  Executing the build file creates a RAR of the adapter at the location destDir with name rarFile

Creating a –ds.xml for an adapter

A sample –ds.xml file generated for Iway adapter is given below. This is specific to JBoss Application Server. For other J2EE Application Servers, the format may vary.

```
<?xml version="1.0" encoding="UTF-8"?>

<connection-factories>

 <no-tx-connection-factory>

<jndi-name>fesb/cf/iWay1</jndi-name>

<rar-name>fiorano-iWay.rar</rar-name>

<connection-definition>javax.resource.cci.ConnectionFactory</connection-definition>

<config-property name="ConfigurationXML" type="java.lang.String">&lt;?xml version="1.0"
encoding="UTF-8"?&gt;
```

```
&lt;java version="1.5" class="java.beans.XMLDecoder"&gt;

 &lt;object
class="com.fiorano.adapter.jca.iway.spi.outbound.IwayManagedConnectionFactory"&gt;

&lt;void property="JNDIName"&gt;

 &lt;string&gt;iWay1&lt;/string&gt;

&lt;/void&gt;

&lt;void property="RARName"&gt;

 &lt;string&gt;4.0&lt;/string&gt;

&lt;/void&gt;

&lt;void property="state"&gt;

 &lt;int&gt;8&lt;/int&gt;

&lt;/void&gt;

 &lt;/object&gt;

&lt;/java&gt;

</config-property>

 </no-tx-connection-factory>

</connection-factories>
```

The above config-property ConfigurationXML is part of the configuration XML file which is generated when configureBC utility is used. Please refer to section 3.4.6 Manual Deployment for more information.

Deploying Adapter in JBoss

- Make sure the name mentioned in –ds.xml in the <rar-name> is same as the name of the RAR file generated. If they are not same, then do either of the following:
    - Change the name of name of RAR file to the name present in <rar-name> tag –ds.xml (or equivalently)
    - Change the name in <rar-name> tag to the name of RAR file (or equivalently)
    - The name of RAR file in the build.xml shown above should be mentioned to the name in <rar-name> tag.

      In the above the IWay adapter's –ds.xml contains <rar-name>fiorano-iWay.rar</rar-name> so the name of RAR file for the adapter should be fiorano-iWay.rar

- Copy the generated –ds.xml and RAR file to <JBOSS_INSTALL_DIR>/server/default/deploy
- Create a directory extlib in <JBOSS_INSTALL_DIR>/server/default and copy all the necessary common jars to <JBOSS_INSTALL_DIR>/server/default/extlib. The list of jars to be copied is given below
    - %FIORANO_HOME%/esb/server/repository/components/CompositeComponentEngine/4.0/cce.jar
    - %FIORANO_HOME%/esb/server/repository/components/Framework/4.0/Framework.jar
    - %FIORANO_HOME%/extlib/dom/dom.jar
    - %FIORANO_HOME%/extlib/jlicense/jlicense.jar
    - %FIORANO_HOME%/extlib/wsdl4j/wsdl4j.jar
    - %FIORANO_HOME%/extlib/saxon/saxon8.jar
    - %FIORANO_HOME%/extlib/saxon/saxon8-dom.jar
    - %FIORANO_HOME%/extlib/saxon/saxon8-jdom.jar
    - %FIORANO_HOME%/extlib/saxon/saxon8-sql.jar
    - %FIORANO_HOME%/extlib/saxon/saxon8-xom.jar
    - %FIORANO_HOME%/extlib/saxon/saxon8-xpath.jar
    - %FIORANO_HOME%/Studio/platform5/core/openide.jar
- add<classpath codebase="extlib" archives="*"/> to <JBOSS_INSTALL_DIR>/server/default/conf/jboss-service.xml
- add the following jars from <fiorano_install_dir>/extlib to <JBOSS_INSTALL_DIR>/lib/endorsed

dom.jar, jaxp-api.jar, resolver.jar, sax2.jar, xalan.jar, xerces.jar, xml-apis.jar

3.4.7.2 Additional Features for Component Administration

The Fiorano Studio provides monitoring support for components that are launched in a Fiorano application. The Fiorano Studio monitors the launch and kill of the component and allows the user to view logs of the component and the messages that queue up at the input of the component.

1. **Component status.** Refer to Figure 3.4.29 to 3.4.32 for all possible status values.

   A component which has not been launched has its name shown in black color. Refer to "chat2" text in Figure 3.4.29.

Figure 3.4.29: A component which is not launched

A component whose handle has been bound has its name shown in blue color. Refer to chat2 text in Figure 3.4.30.

Figure 3.4.30: A component which is bound

A component which has been launched has its name shown in green color. Refer to chat2 text Figure 3.4.31.

Figure 3.4.31: A component which is launched

A component which has been stopped (or killed) has its name shown in red color. Refer to chat2 text Figure 3.4.32.

Figure 3.4.32: A component which is stopped (or killed)

2. **View logs**. To view the Logs of a component using the Fiorano Studio

   a. Right-click on the component and select **View Logs** from the menu list as illustrated in Figure 3.4.33. You also use the same menu to clear and export logs as well.

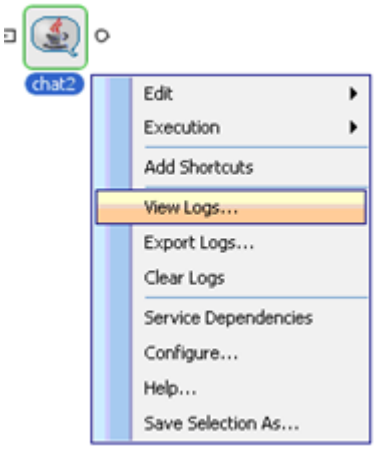


Figure 3.4.33: Menu showing the option to view component logs

   b. A dialog box appears which displays the output and error logs for the component, as shown in Figure 3.4.34. You can clear, export or refresh the logs. Also, using the same window you can change the service instance to view logs of another component

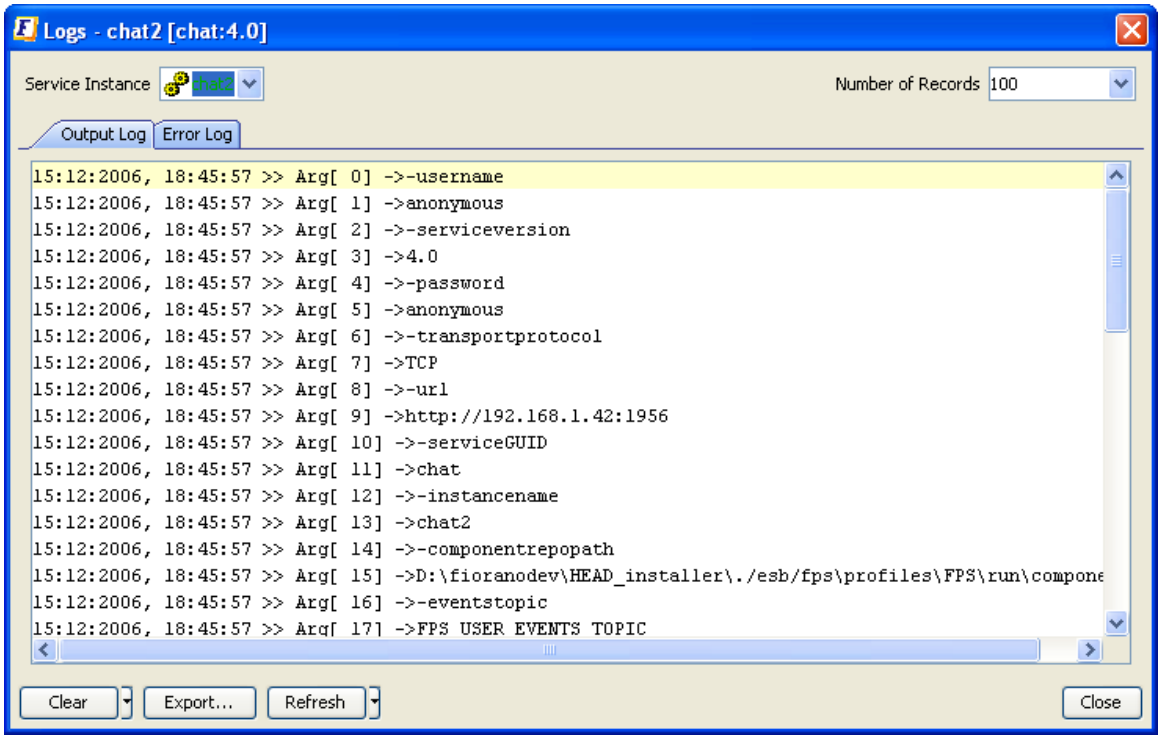


Figure 3.4.34: UI showing logs of the component

3. **View queued messages:** To view the queued messages at the input or output port of a component using Fiorano Studio:

   a. Right-click input port and select Browse Messages from the menu list as shown in Figure 3.4.35.

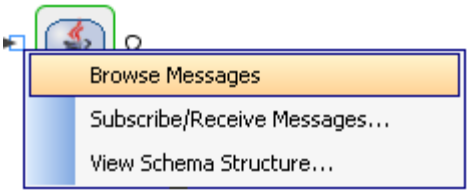


Figure 3.4.35: Menu to browse queued messages

   b. A dialog box appears to show the queued messages as shown in Figure 3.4.36. Click on the message to see its properties and the content.

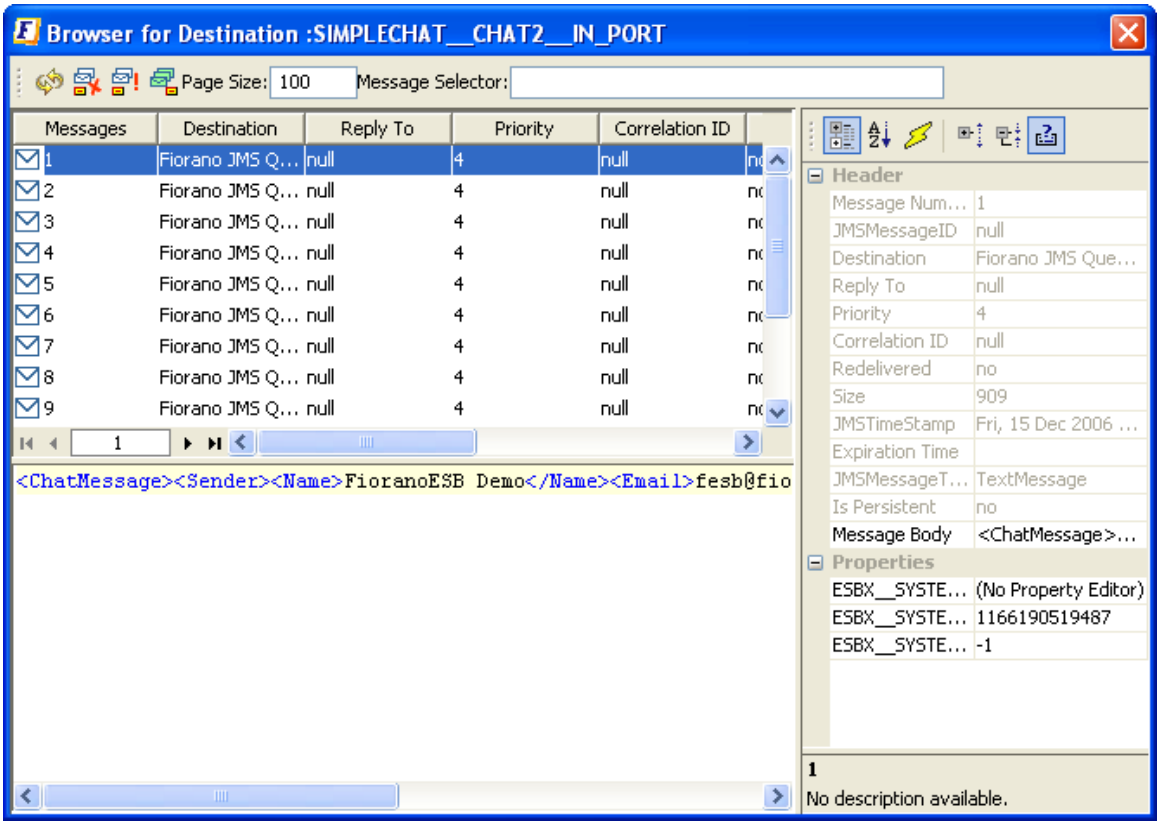


Figure 3.4.36: UI showing the queued messages on a queue

c. When you choose the Subscribe/Receive messages option, the messages are picked up from the queue/topic and are displayed as illustrated in Figure 3.4.37. Choose the appropriate message to view details.

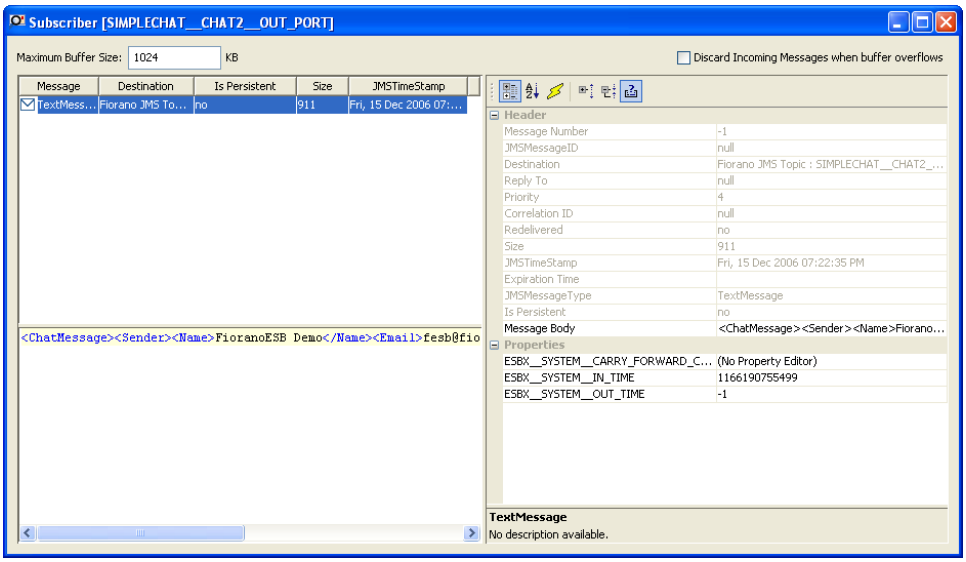


Figure 3.4.37: UI showing the queued messages on a topic

3.4.8 InMemory Launch

A component within an Event Process can be configured to be automatically launched within the same JVM as the peer server to which it is connected. Such a launch is called an In-Memory launch, since the component runs in the memory of the peer server JVM.

For each component launched in memory class loaders are created. The class loaders can be created in two ways, they are:

Cached Class Loader:

For each service, a class loader is created if it is not already created. For example, if three services a, b, and c with dependencies a->b->c.
When the component is launched in memory a different class loader is created for each service with resources required for execution by the service. And these class loaders are cached for other services using same libraries.

When calculating class loader of a service, it loads the resources of service as URL class path and maintains a list of class loaders of dependent services as parent class loaders. To resolve a particular class in classpath, the called class should be present in the URL classpath of class loader of the caller service or it should be present in parent class loaders. As we are caching the created class loaders, the memory occupied for multiple service instances can be reduced.

Problems in this approach

User has to set the dependencies of the services without any class path issues. If multiple instance of same component is launched in memory, as class loaders are cached, each instance use the same class loaders. Hence, any static variables in component are shared.

UnCached Class Loader

Unlike Hierarchical Class Loading, this uncached class loader creates a single URL class loader for the service including its dependencies. And these class loaders are not cached.

In this approach level of dependencies of the service does not matter and separate class loader is created for each instance. This solves the two problems addressed in cached class loader approach.

In this approach there would be an increase in memory usage of peer server as any class loaders are not cached. User has given an option to choose between these two approaches at peer server level.

1. Open Peer Server profile in Fiorano Studio.
2. Navigate to FPS->Fiorano->ESB->Peer->Launch->ClassLoaderManager.

Note: Cache class loaders property allows users to select option whether to cache the In Memory service class loaders or use separate class loaders, if it is true class loaders are cached, if false separate class loaders are used.

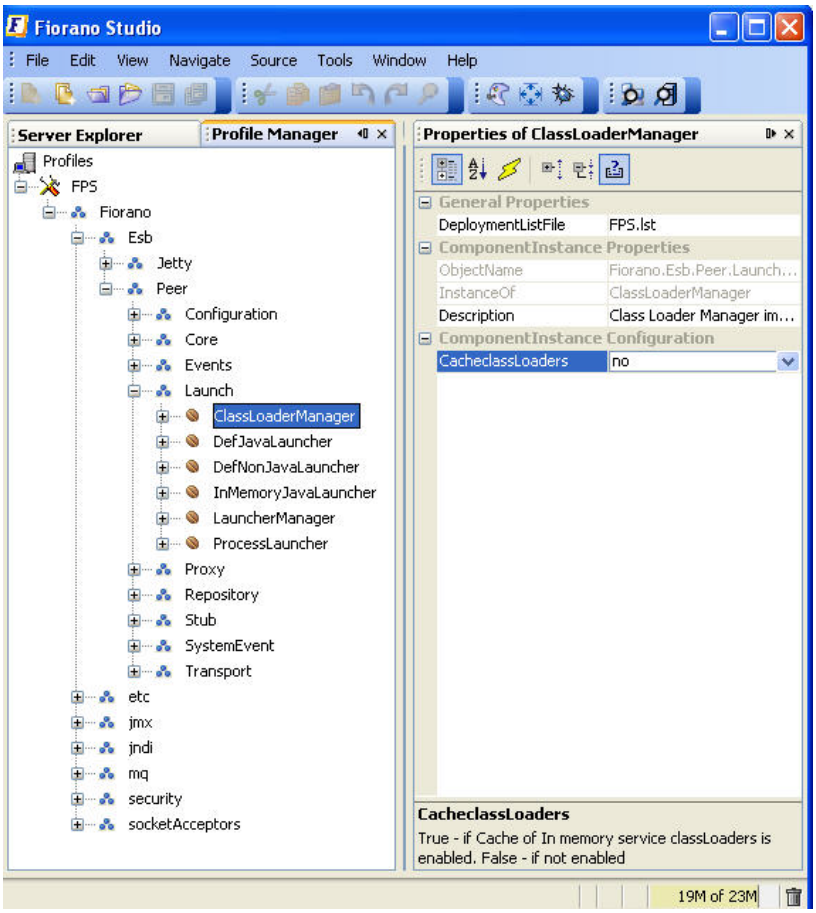


Figure 3.4.38: Properties of ClassLoaderManager

3. Save the profile and restart FPS.

3.5 Export and Import Service Components

The Fiorano Studio tool provides facilities to export and import components.

3.5.1 Exporting a Component

1. Right-click the component to be exported in **Service Repository->Registered Services** tree and choose **Export** from the drop-down list to export the component as shown in Figure 3.5.1.

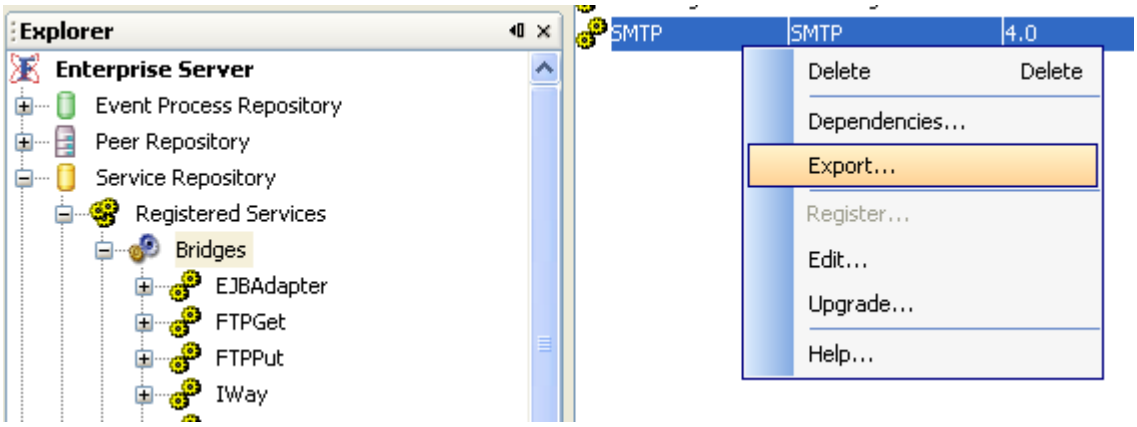


Figure 3.5.1: Menu to export the component from the Fiorano Studio

2. In the Customize Export SMTP: The 4.0 dialog box, provide the location where the file needs to be saved as shown in Figure 3.5.2. Also select any system library dependencies that you wish to export along with this component.

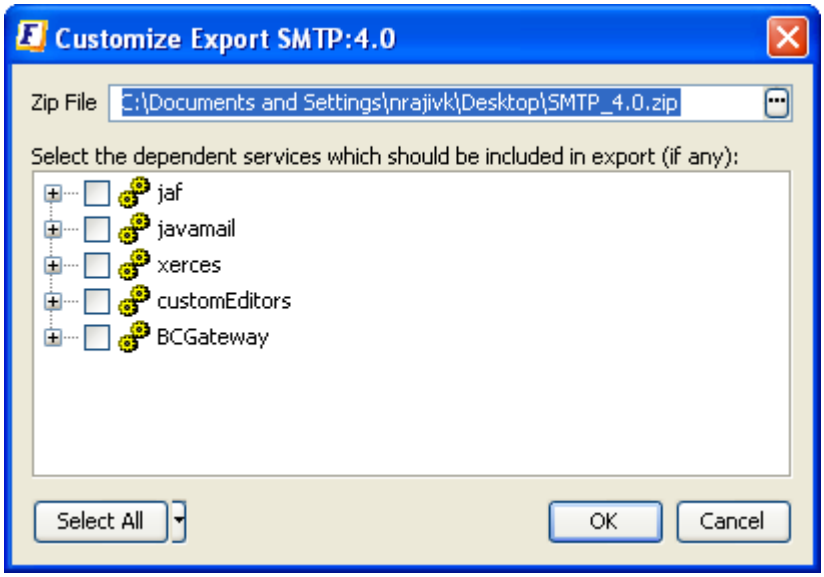


Figure 3.5.2: UI to save the exported component and its libraries.

3. Click **OK** to export the component as a zip file that is saved on the file system.

3.5.2 Importing a Component

The Import operation is similar to the export operation discussed above. An Import is typically performed by reading a .zip file containing an exported component, from the file system.

1. Right-click on **Service Repository->Registered Services** and choose **Import Service** from the menu list as shown in Figure 3.5.3.

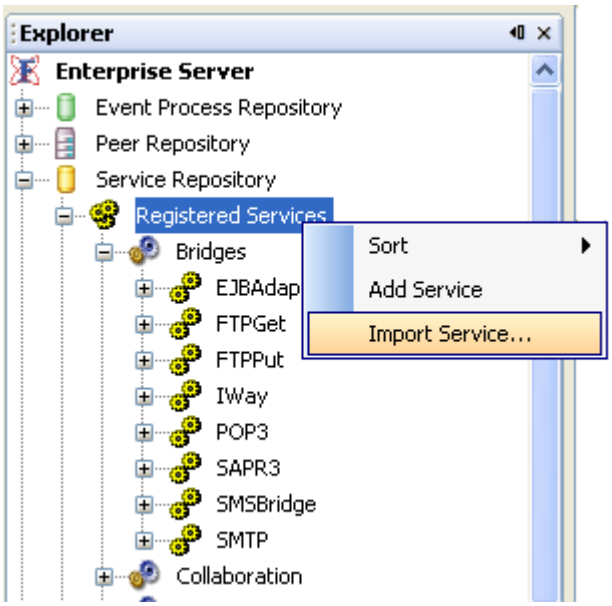


Figure 3.5.3: Menu to import a new component

2. Choose the **Import Service** from the menu list to import from the zip file. Click the **OK** button to import. In the file-chooser dialog, browse the file system and/or type in the path of the .zip file containing the component(s) to import.
3. Choose the .zip file containing the Component(s). The list of components in the .zip file is displayed, as shown in Figure 3.5.4. Choose the components to import and click **OK**.

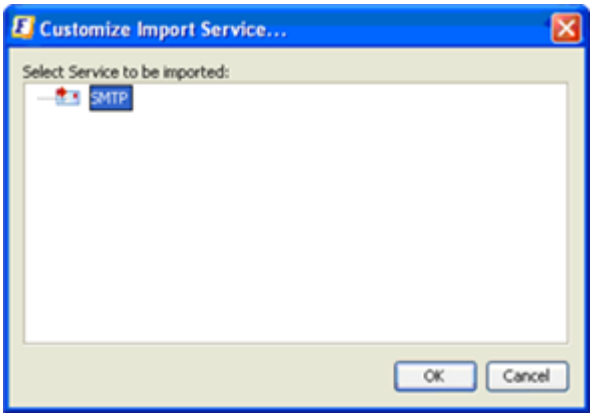


Figure 3.5.4: List of components to import

4. The Properties for the imported component are displayed as illustrated in Figure 3.5.5. Modify any details as necessary and click the **OK** button.

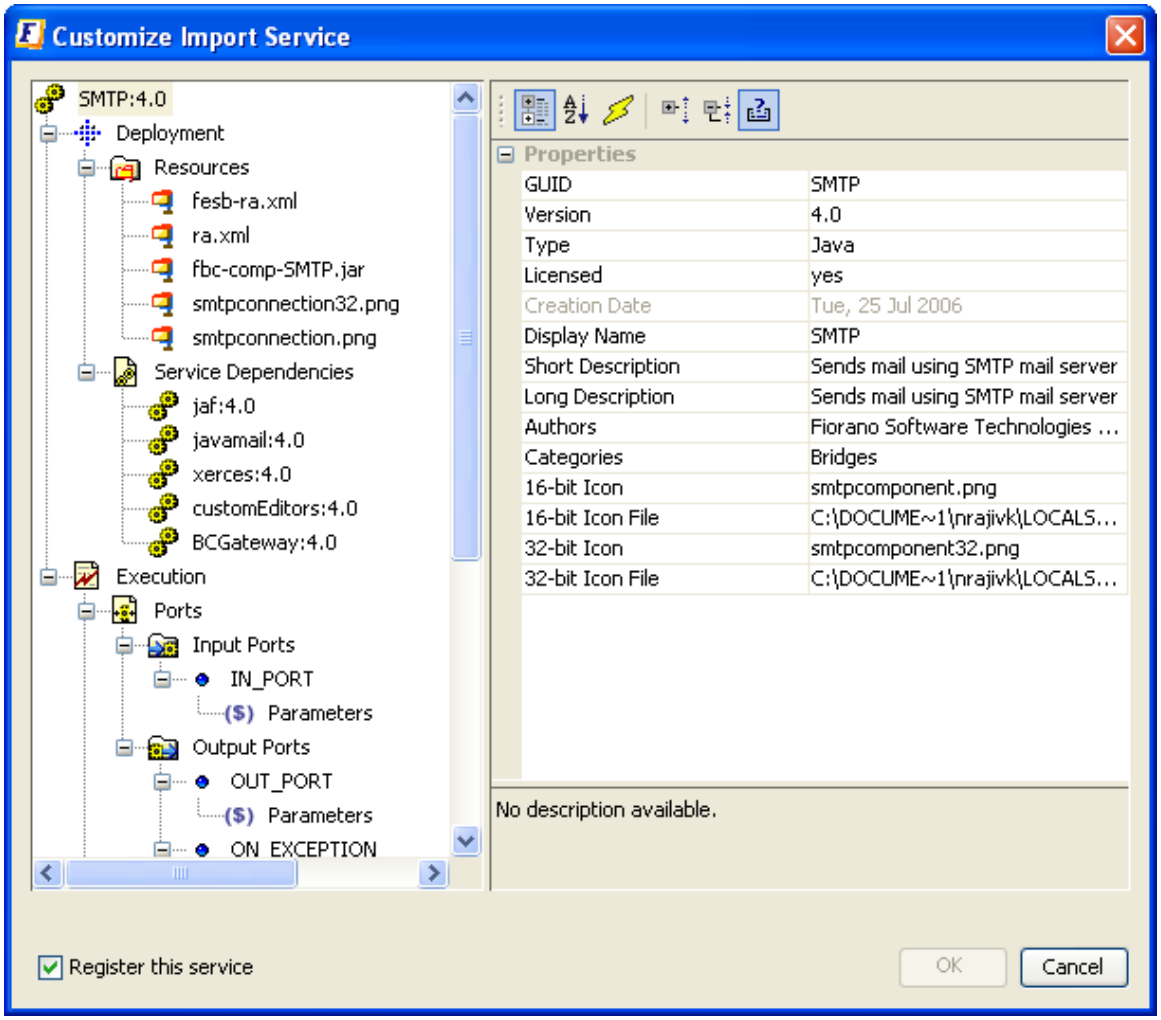


Figure 3.5.5: Properties of the imported component

Note: Exported components can be moved from one Fiorano Enterprise Server to another provided both the Fiorano Enterprise Servers are of same version and build.

3.6 Component Creation

Apart from the exhaustive list of pre-built components, custom components can be written, built, and deployed into Fiorano SOA Platform by developers. To aid developers in component creation, the platform provides a template engine to generate the skeleton code for custom components in Java, C, C++, C# (.Net). The following sections describe component creation in different languages

3.6.1 Template Engine

The Template engine is located at %FIORANO_HOME%\esb\tools\templates (this is referred as %TEMPLATE_ENGINE% going forward in this doc) directory in the installation. The templates and scripts used for component creation are organized as shown in the Figure 3.6.1.

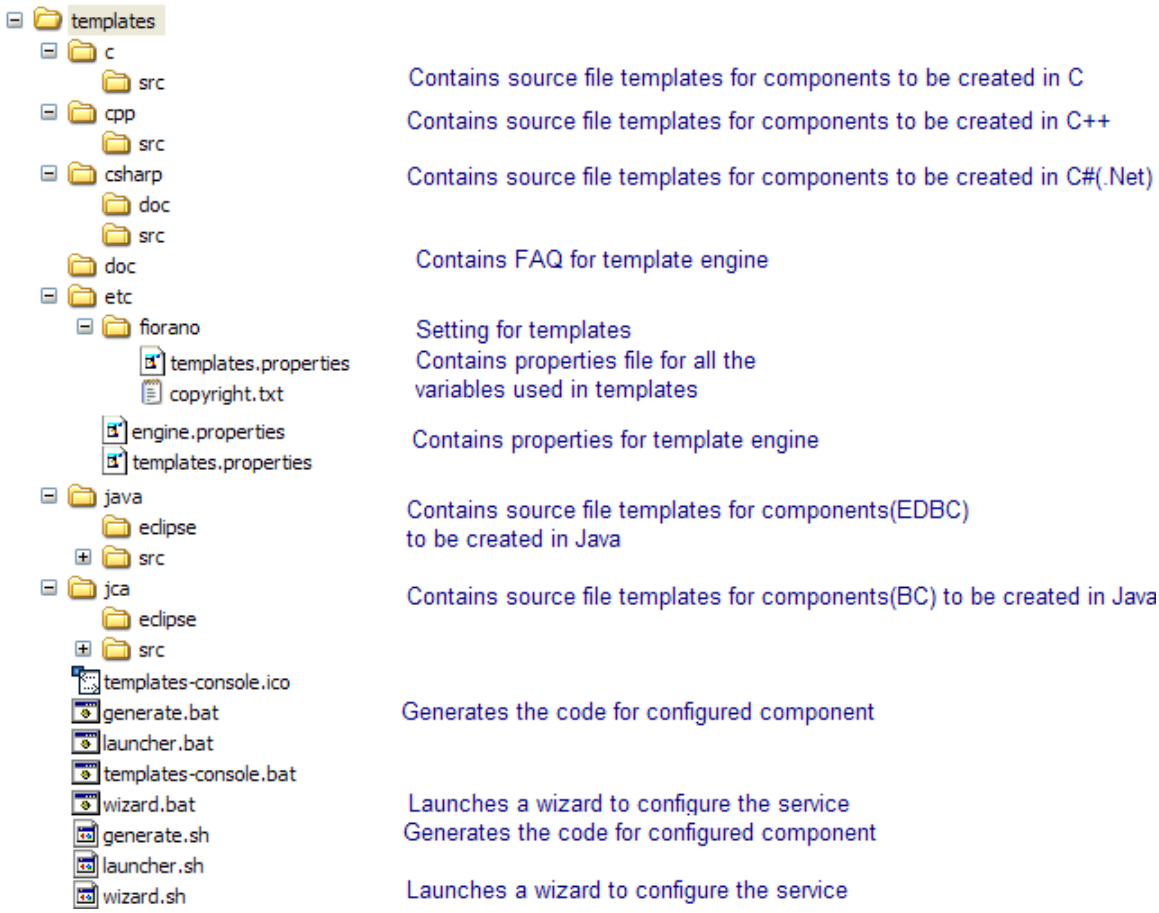


Figure 3.6.1: Structure of the Template engine

Component creation can be done in two ways:

- From the command line using scripts
- From the Fiorano Event Process Orchestration (Fiorano Studio)

3.6.1.1 Component Creation from the Command Line

The steps for component creation from command line are:

1. Create a new setting (optional)
2. Define new variables that can be used in templates (optional)
3. Modify the templates (optional)
4. Configure the service to be created.
5. Generate the code for the component.

6. Build the component.
7. Deploy the component.

Each of these steps is explained in the following sections.

3.6.1.2 Creating a Setting

A Setting contains the defined variables that can be used in the templates of source files and values that replace these variables during the source generation.

- Each folder in %TEMPLATE_ENGINE%/etc is a setting. The Fiorano setting can be seen in Figure 3.2.1.

A setting typically consists of the following files:

templates.properties

This is a file containing the variables that can be used in the templates of source files.

copyright.txt

This is a text file containing the copyright notice that should be included in the generated source files.

3.6.1.2.1 To create a new setting

- Create a directory in %TEMPLATE_ENGINE%\etc with the setting name required.
- Create the files templates.properties and copyright.txt.
- Define appropriate variables in templates.properties.

3.6.1.3 Variables

The template engine uses variables to substitute values during the code generation.

3.6.1.3.1 In-built variables

The following table illustrates in-built variables:

Variable Name	Description
guid	exact GUID of the service
serviceGUID	GUID of service with first letter as lowercase
ServiceGUID	GUID of service with first letter as uppercase
serviceguid	GUID of service in lowercase
version	version of the service

Variable Name	Description
inputPorts	java.util.ArrayList containing input port names
outputPorts	java.util.ArrayList containing output port names
inMemoryLaunchable	java.lang.Boolean specifying whether service can be launched in-memory
isSyncRequestType	java.lang.Boolean specifying whether SyncRequestType is enabled or not for input port
isBCConnEnabled	java.lang.Boolean specifying whether connection semantics is enabled for a JCA-complaint component
fioranoHome	Fiorano SOA Platform installation directory
copyright	copyright notice for source file (as javadoc comment) [content of copyright.txt from setting directory]
Date	current date
rootDir	root directory to resemble root package
BCRootDir	root directory to resemble root package for JCA-complaint components

The values for the above variables either picked up from pre-defined locations or computed based on other variables and hence the user does not have complete control of these variables.

Note: These variables are available independent of the Setting being used

3.6.1.3.2 Variables Defined in the fiorano Setting

The following table explains the variables present in the fiorano Settings:

Variable Name	Description
rootPackage	Name of the root package in which sources are generated
JCARootPackage	Root package for JCA-complaint Code generation
author	Developer name that appears in the source file
dateFormat	Date format used for date variable

3.6.1.3.3 Defining New Variables

New variables can be defined by adding an entry in the templates.properties file.

This is a java properties file containing:

- The variables defined that can be used in the templates of the source files (present in %TEMPLATE_ENGINE%\<lang>\src).
- The values for the defined variables which is substituted during the source code generation.

A typical file is shown in the Figure 3.6.2:

```
# developer name that appears in source file
author=Fiorano Software Technologies Pvt. Ltd.

# name of root package in which sources are generated
rootPackage=com.fiorano.edbc.${serviceguid}

# root package for JCA-complaint Code generation
JCARootPackage=com.fiorano.adapter.jca.${serviceguid}

# date format used for date variable
dateFormat=EEE, d MMM yyyy
```

Figure 3.6.2: templates.properties file containing defined variables and their values

To create a new variable with name e-mail and value user@domain.com, edit the templates.properties file as shown in the Figure 3.6.3.

```
# developer name that appears in source file
author=Fiorano Software Technologies Pvt. Ltd.

# name of root package in which sources are generated
rootPackage=com.fiorano.edbc.${serviceguid}

# root package for JCA-complaint Code generation
JCARootPackage=com.fiorano.adapter.jca.${serviceguid}

# date format used for date variable
dateFormat=EEE, d MMM yyyy

# email of the author
email=user@domain.com
```

Figure 3.6.3: Modified templates.properties file

For more Information regarding editing properties file, see http://java.sun.com/j2se/1.4.2/docs/api/java/util/Properties.html#load(java.io.InputStream)

3.6.1.4 Modifying the Templates

As shown in figure 3.6.1 source file templates for different languages are present at %TEMPLATE_ENGINE%\<lang>\src . These can be edited to use the variables defined in the previous section.

3.6.1.4.1 Using the Variables in Template

The variables defined can be referred in the template files by specifying as ${variable_name}. Shown in image:

```
/**
 * Execution class for ${ServiceGUID}
 * This class manages ( creating, caching andf destroying) all the JMS objects
 * It creates all the required listeners and listens to JMSExceptions.
 *
 * @author ${author}
 * @created ${date}
 */

public class ${ServiceGUID} implements{
```

Figure 3.6.4: Source template file showing use of variables

3.6.1.5 Defining Components

To communicate the details of component to the servers we need to define the component in such a way that servers understand the component. The language used for this purpose is XML. A component is defined using a XML file called ServiceDesriptor.xml. The definition includes properties of component such as, its name, operating system to which the component is complaint to, channels through which the component interacts with other components among other properties.

This section aims at understanding all such properties, defining them and see the affect such properties have.

A wizard to aid users to define the component to be created is provided in the template engine.

3.6.1.5.1 On Windows

- Browse to %TEMPLATE_ENGINE%
- Run templates-console.bat; a command prompt opens in directory pointed to %TEMPLATE_ENGINE%.
- Run the command wizard.bat specifying the destination directory as follows:

```
usage: wizard -dest <dir>
 -dest <dir>   destination project dir
```

Note: Run wizard.bat without any arguments to see the help.

3.6.1.5.2 On Linux

Browse to %TEMPLATE_ENGINE% in Linux console

Run the command wizard.sh specifying the destination directory as follows:

Usage: wizard.sh -dest <dir>

<dir> - The directory in which the component should be created.

Note: Run wizard.sh without any arguments to see the help.

3.6.1.6 Getting familiar with wizard and service configuration

In this section the different wizard steps of the configuration wizard are explained. Specific details with respect to language is mentioned in respective sections

3.6.1.6.1 Business Component Header Panel

This panel allows you to specify the type of component to be created and identification details of the component to be used in the tools. **ServiceGUID** uniquely identifies the Service.

The Category can be selected from a pre-existing list in the Category Selection Box; alternatively, a new category can be added directly in the text box (comma separated) or by clicking on the button at the right typing in the corresponding name.

Finally, enter the display name for the component; the Display Name is typically different from the ServiceGUID and is the name displayed by the Fiorano Studio for this Component.

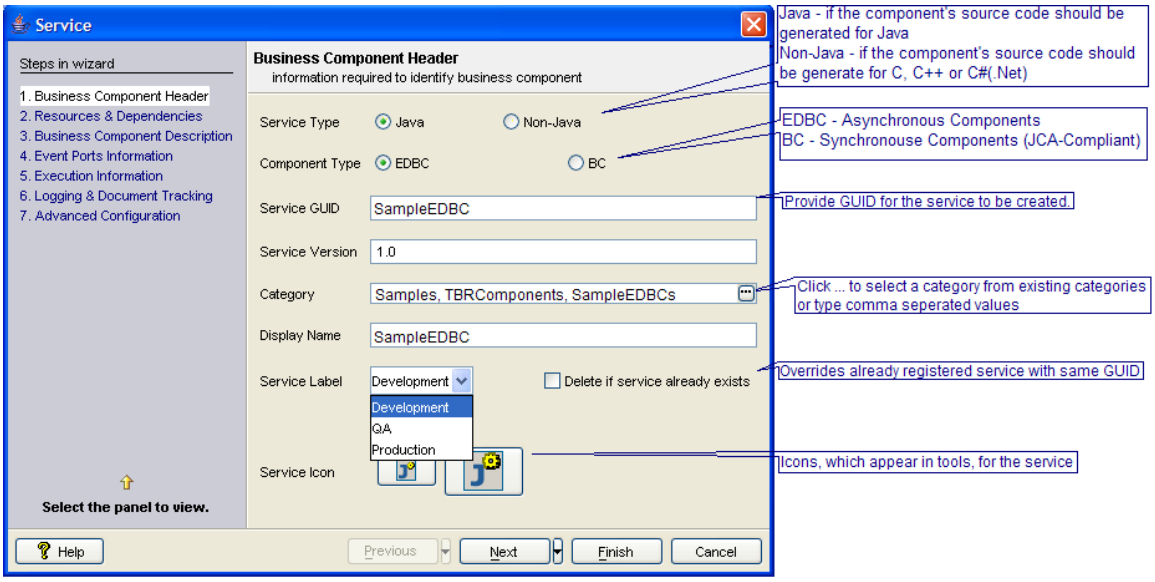


Figure 3.6.5: Business Component Header Panel

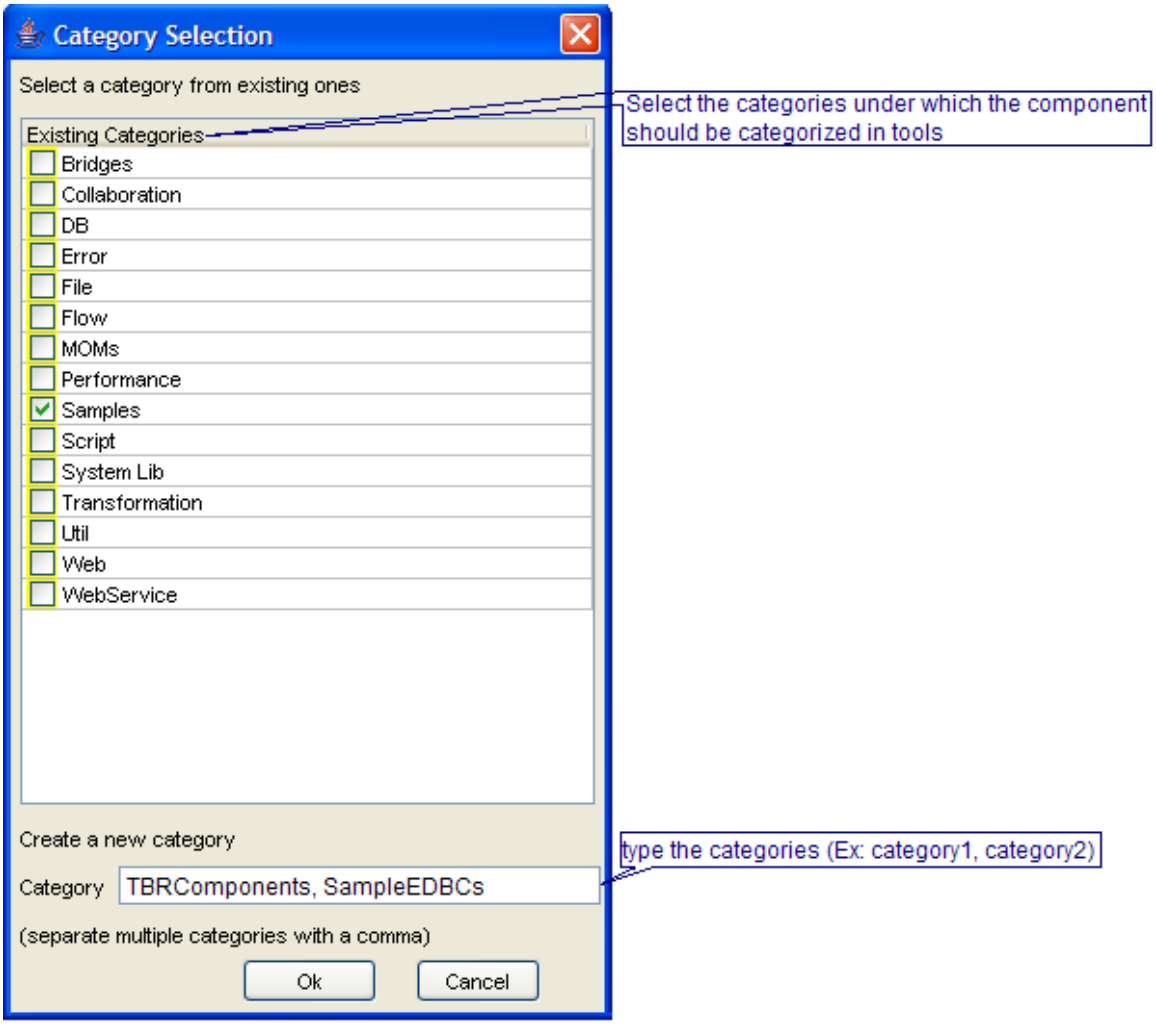


Figure 3.6.6: Specifying categories

The components that are registered, in the Enterprise server, are categorized under different categories for easier maintenance. The components are shown as grouped under the category they belong to in tools; Figure 3.6.8 illustrates image of organization of categories in Fiorano Studio.

```
<ServiceHeader canLaunchCPSInProcess="false"
    isErrorHandlingSupported="true" isLicensed="false"
    launchable="true" native="false">
    <Name>SampleEDBC</Name>
    <ServiceGUID>SampleEDBC</ServiceGUID>
    <Author>admin</Author>
    <CreationDate>17/12/2006</CreationDate>
    <Icon>icon.png</Icon>
    <LargeIcon>icon32.png</LargeIcon>
    <Version>1.0</Version>
    <Label>Development</Label>
    <Category>Samples</Category>
    <Category>TBRComponents</Category>
    <Category>SampleEDBCs</Category>
    <ShortDescription>A sample EDBC</ShortDescription>
    <LongDescription>A sample component created which listen
    component</LongDescription>
</ServiceHeader>
```

Figure 3.6.7: ServiceDescriptor.xml containing properties specified in the Business Component Header Panel

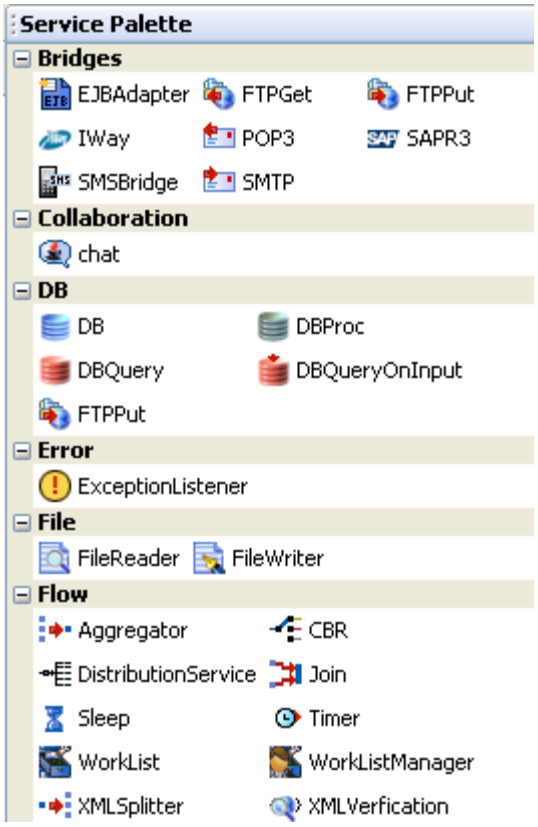


Figure 3.6.8: components categorized in Fiorano Studio

Note: Business Component Header panel is the only panel in which the details are mandatory. User can choose to finish the service definition any time after all the required details are specified in this panel.

3.6.1.6.2 Resources and Dependencies Panel

Any component created in general requires resources –third party libraries, Fiorano libraries or other files – required at the time of component configuration or execution.

Resources and dependencies both serve the same purpose; providing the component with libraries or files required. However they differ in the way these file or libraries are treated by Fiorano servers.

Resources can be any files which are used by the component. Typically resource files are of types – dll, zip, jar, so, exe. However there is no strict restriction on this, a file of any type can be added as a resource. The server makes a local copy of these files in the component's folder in the component repository (%FIORANO_HOME%\esb\fes\repository\components). So a same file added as a resource to two different components is copied into respective folders of both the components. Also when these components are launched in-memory of same peer server, resources is loaded into the respective class loaders of the components.

On the other hand Dependencies are predefined. Every component or system library registered can be added as a dependency. The dependencies are referenced from the existing location and are not copied locally. Another advantage is that dependencies are loaded only once when the components are launched in-memory of same peer server, there by reducing the memory footprint.

More details on registering components and system libraries are available in section **Service Component Characteristics, Configuration and Deployment**.

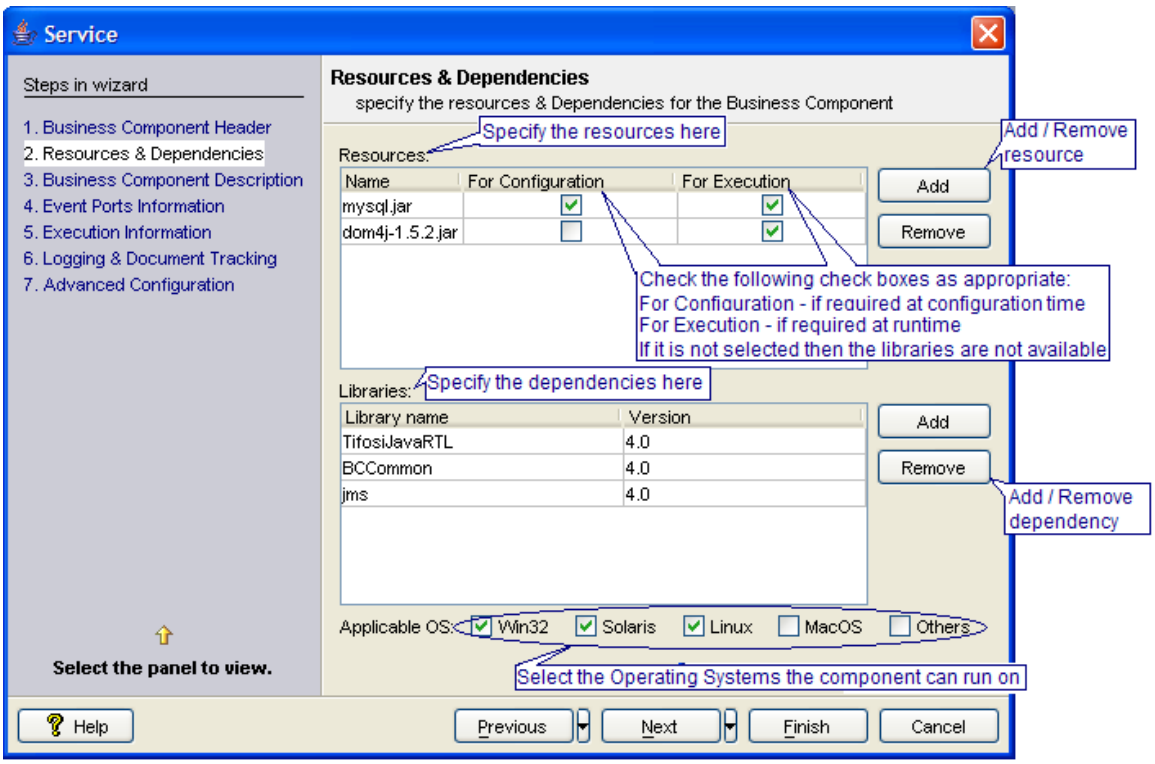


Figure 3.6.9: Resources & Dependencies panel

Resources

To Add: Click the **Add** button against Resources shown in Figure 3.6.9. Choose the required resources from the file dialog which opens up

To Remove: Select the resource to remove. Click the **Remove** button against Resources shown in Figure 3.6.9.

Dependencies

To Add: Click the **Add** button against Libraries shown in Figure 3.6.9. A window titled **Select Libraries for the Business Component** opens listing all the components and system libraries registered (shown in Figure 3.6.9). Select the required libraries and click the **OK** button.

Applicable OS

Select the operating systems to which the component is compatible.

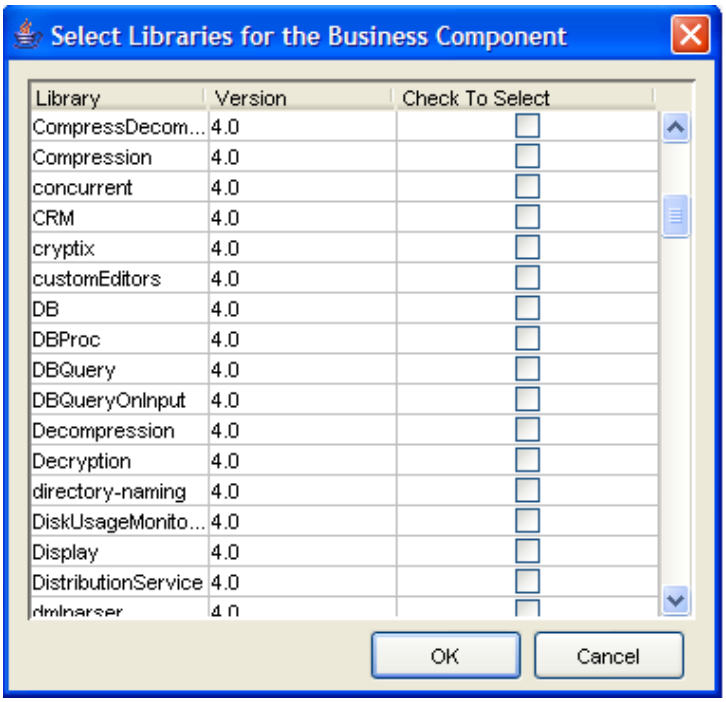


Figure 3.6.10: List of libraries

```
<Deployment deploymentStage="DEVELOPMENT" isAutoInstallable="true">
    <OS>Linux</OS>
    <OS>Solaris</OS>
    <OS>Win32</OS>
    <ServiceDependency>
        <ServiceGUID>TifosiJavaRTL</ServiceGUID>
        <Version>4.0</Version>
    </ServiceDependency>
    <ServiceDependency>
        <ServiceGUID>BCCommon</ServiceGUID>
        <Version>4.0</Version>
    </ServiceDependency>
    <ServiceDependency>
        <ServiceGUID>jms</ServiceGUID>
        <Version>4.0</Version>
    </ServiceDependency>
    <Resources>
        <Resource requiredForConfiguration="true" requiredForExecution="true">
            <Name>mysql.jar</Name>
        </Resource>
        <Resource requiredForConfiguration="false" requiredForExecution="true">
            <Name>dom4j-1.5.2.jar</Name>
        </Resource>
    </Resources>
</Deployment>
```

Figure 3.6.11: ServiceDescriptor.xml containing properties specified in the Resources and Dependencies Panel

3.6.1.6.3 Business Component Description Panel

The author and the description of the component can be provided in this following panel.

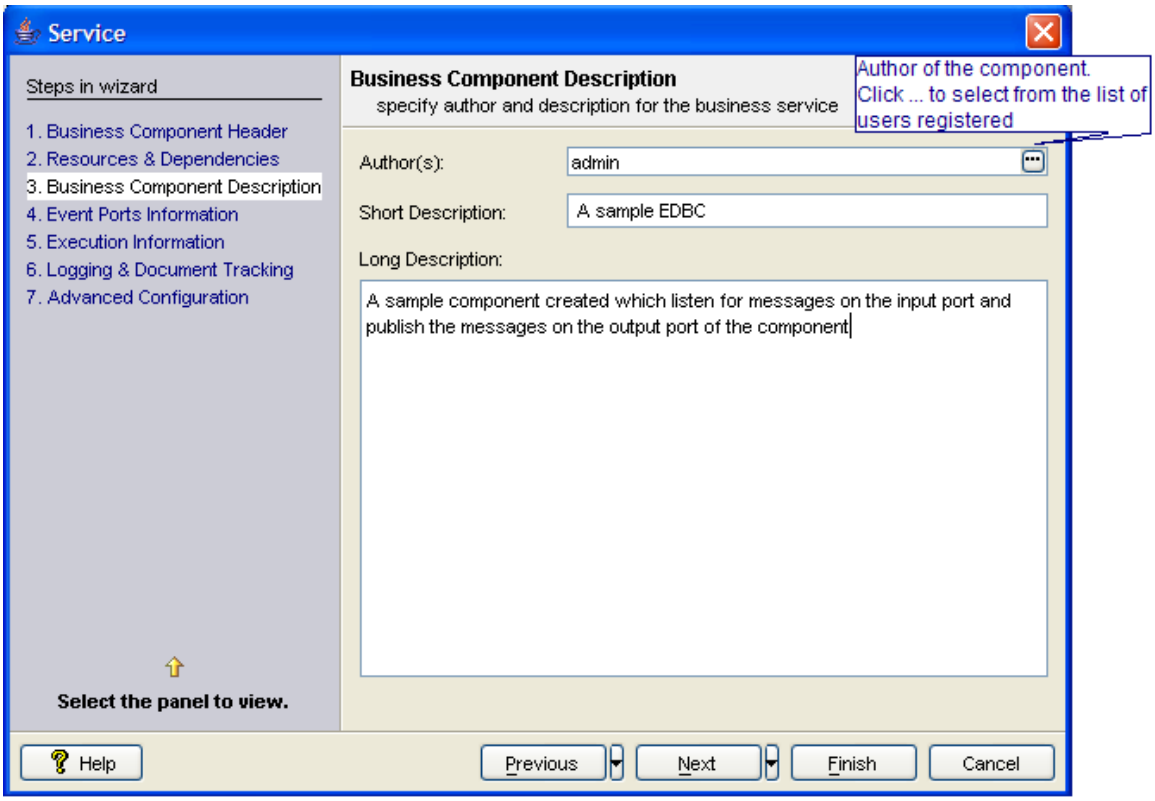


Figure 3.6.12: Panel containing description of the component

```
<ServiceHeader canLaunchCPSInProcess="false"
    isErrorHandlingSupported="true" isLicensed="false"
    launchable="true" native="false">
    <Name>SampleEDBC</Name>
    <ServiceGUID>SampleEDBC</ServiceGUID>
    <Author>admin</Author>
    <CreationDate>17/12/2006</CreationDate>
    <Icon>icon.png</Icon>
    <LargeIcon>icon32.png</LargeIcon>
    <Version>1.0</Version>
    <Label>Development</Label>
    <Category>Samples</Category>
    <Category>TBRComponents</Category>
    <Category>SampleEDBCs</Category>
    <ShortDescription>A sample EDBC</ShortDescription>
    <LongDescription>A sample component created which listen for messages
    on the input port and publish the messages on the output port of the
    component</LongDescription>
</ServiceHeader>
```

Figure 3.6.13: ServiceDescriptor.xml containing properties specified in the Business Component Description Panel

3.6.1.6.4 Event Ports Information Panel

Data transfer among components is done over JMS. So the components require data channels to receive the request and send the results. These data channels are called ports. Ports are JMS destinations; either topics or queues. A component can have any number of input ports and output ports. The port details are configured in this panel (Figure 3.6.14).

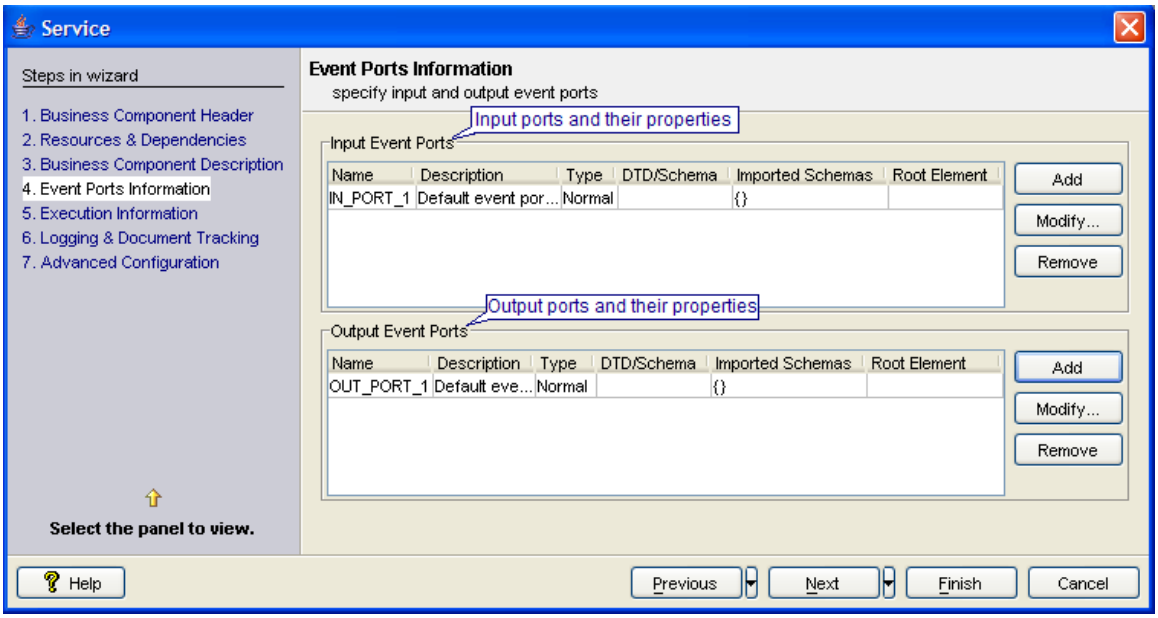


Figure 3.6.14: Configuring the port details for the component

Port Properties

Property	Description
Name	Name of the port that appears in the Fiorano Studio.

Property	Description
Description	A statement typically indicating the purpose of this port.
Type	Chose from Normal or Sync Request Reply.
DTD/Schema	If the component expects or sends out messages in XML format specify the DTD/XSD the message should be compliant. It can be used as an assertion to make sure that the component receives the message in the format it expects or that it sends out the message in the format it is supposed. Fiorano Studio checks for the format mismatches when the components are connected by a route and intimates them.
Imported Schemas	When definition is a XSD, use this field to define the elements that are resolved from a different schema. Multiple schemas can be provided here.
Root Element	The root element of the XML message.
Generate XMLBeans	This is an option to generate Java classes from the XSDs that are provided for ports. The developer of a custom component can use these to access the input data and generate the output without bothering about input message parsing and output XML generation. The schemas for which this option is selected are stored in schemasForXMLBeans folder in component directory.

Note: if the fields DT/Schema, imported Schemas and Root Element should be used only the message are of XML format.

Adding a Port

To add a port, click the **Add** button adjacent to **Input Event Ports** details or **Output Event Ports** to add an input or an output port respectively. The port details appear as shown in Figure 3.6.11.

Removing a Port

To remove a port, select the port to be removed and click the **Remove** button adjacent to **Input Event Ports** details or **Output Event Ports** as appropriate.

Modifying details of Port

Select the port to be modified and click on **Modify** button adjacent to **Input Event Ports** details or **Output Event Ports** as appropriate. This opens up a dialog where the details can be modified.

Output port details can be modified similarly except that for output port 'Normal' and 'Sync Request Reply' (in Figure 3.6.15) are not present.

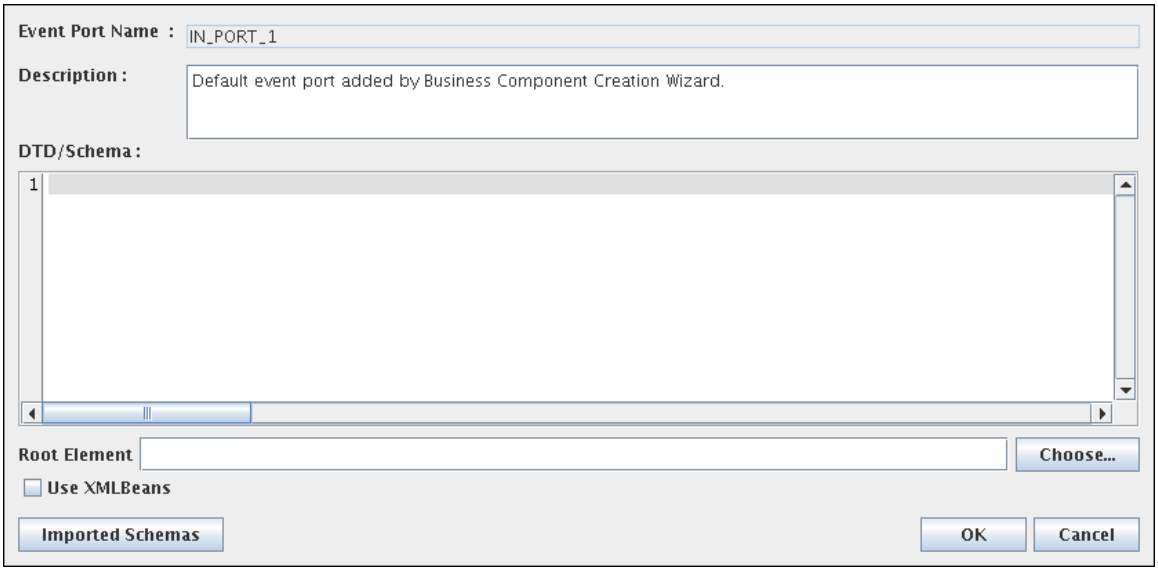


Figure 3.6.15: Modifying the event port properties

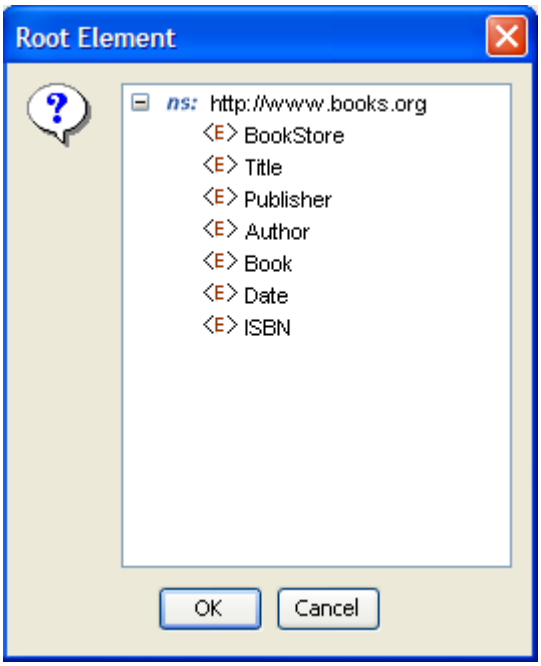


Figure 3.6.16: Selecting the root element

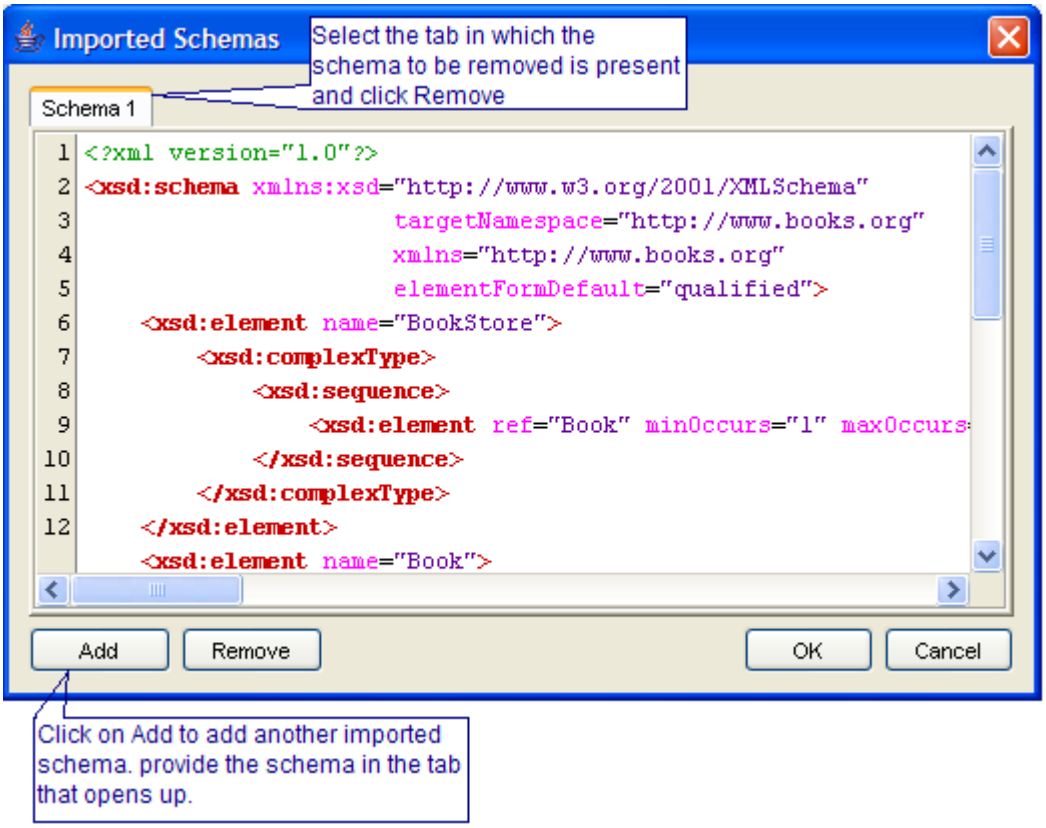


Figure 3.6.17: Adding and removing imported schemas

```
    <PortDescriptor>
        <InPort isSyncRequestType="false">
            <Name>IN_PORT_1</Name>
            <Description>Default event port added by Business Component Creation Wizard.</Description>
            <XSD><![CDATA[<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                      targetNamespace="http://www.books.org"
                      xmlns="http://www.books.org"
                      elementFormDefault="qualified">
    <xsd:element name="BookStore">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Book">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string"/>
    <xsd:element name="Date" type="xsd:string"/>
    <xsd:element name="ISBN" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>]]></XSD>
            <JavaClass/>
            <Param name="External_http://www.books.org">&lt;?xml version="1.0"?&gt;
&lt;xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                      targetNamespace="http://www.books.org"
                      xmlns="http://www.books.org"
                      elementFormDefault="qualified"&gt;
    &lt;xsd:element name="BookStore"&gt;
        &lt;xsd:complexType&gt;
            &lt;xsd:sequence&gt;
                &lt;xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/&gt;
            &lt;/xsd:sequence&gt;
        &lt;/xsd:complexType&gt;
    &lt;/xsd:element&gt;
    &lt;xsd:element name="Book"&gt;
        &lt;xsd:complexType&gt;
            &lt;xsd:sequence&gt;
                &lt;xsd:sequence&gt;
                    &lt;xsd:element ref="Title" minOccurs="1" maxOccurs="1"/&gt;
                    &lt;xsd:element ref="Author" minOccurs="1" maxOccurs="1"/&gt;
                    &lt;xsd:element ref="Date" minOccurs="1" maxOccurs="1"/&gt;
                    &lt;xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/&gt;
                    &lt;xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/&gt;
                &lt;/xsd:sequence&gt;
            &lt;/xsd:complexType&gt;
        &lt;/xsd:element&gt;
        &lt;xsd:element name="Title" type="xsd:string"/&gt;
        &lt;xsd:element name="Author" type="xsd:string"/&gt;
        &lt;xsd:element name="Date" type="xsd:string"/&gt;
        &lt;xsd:element name="ISBN" type="xsd:string"/&gt;
        &lt;xsd:element name="Publisher" type="xsd:string"/&gt;
    &lt;/xsd:schema&gt;</Param>
                <Param name="rootElementNS">http://www.books.org</Param>
                <Param name="rootElementName">BookStore</Param>
            </InPort>
            <OutPort isSyncRequestType="false">
                <Name>OUT_PORT_1</Name>
                <Description>Default event port added by Business Component Creation Wizard.</Description>
                <JavaClass/>
            </OutPort>
        </PortDescriptor>
```

Figure 3.6.18: ServiceDescriptor.xml containing properties specified in the Event Port Properties Panel

3.6.1.6.5 Execution Information Panel

This panel allows the user to specify the execution details of the component. A component while executing, might require parameters to execute different request or details of handling different request. There are two ways of passing this information to the component.

By configuring the details in the Configuration Property Sheet of the panel (discussed in component configuration section)

By defining the command line arguments that can be passed to the component during the launch of the component. These command line arguments are captured as runtime arguments in this panel.

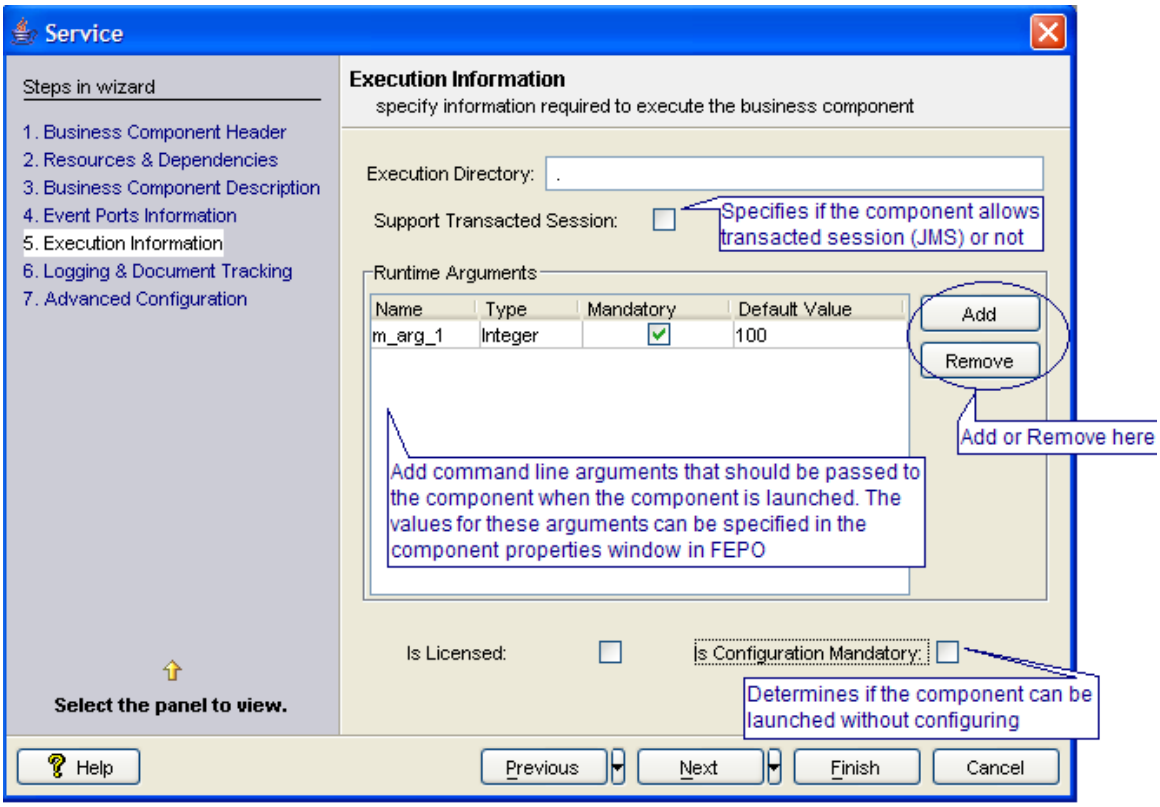


Figure 3.6.19: Configuring execution details of the component

Properties captured in Execution Information Panel

Property	Description
Execution Directory	Working directory of the component during the runtime
Is Licensed	Specifies if the component is licensed. If this is checked and component's license is not present in the license file the component will not be launched
Is Configuration Mandatory	Specifies whether the component requires to be configured before launching the component. If checked the component will not launch unless the component is configured using the configuration property sheet
Runtime Arguments	These are the command line arguments that are passed to the argument. The values for these can be provided in the properties window of the component in the Fiorano Studio.

```
<ServiceHeader canLaunchCPSInProcess="false"
    isErrorHandlingSupported="true" isLicensed="false"
    launchable="true" native="false">
    <Name>SampleEDBC</Name>
    <ServiceGUID>SampleEDBC</ServiceGUID>
    <Author>admin</Author>
    ....
    ....
    ....
</ServiceHeader>
    ....
    ....
    ....
<Execution inMemoryLaunchDefault="false"
    isAutoLaunchAvailable="true" isExternalLaunchAvailable="true"
    isFailoverSupported="true" isInMemoryLaunchable="true"
    manualLaunchDefault="false" transaction="None" type="java">
    <ExecutionDir>.</ExecutionDir>
    <Executable>$executable</Executable>
    <InMemoryLaunchImpl>com.fiorano.edbc.sampleedbc.SampleEDBC</InMemoryLaunchImpl>
    <LogModules>
        <Module defaultTraceLevel="INFO" maxTraceLevel="ALL" name="com.fiorano.edbc.sampleEDBC.sampleEDBC"/>
    </LogModules>
    <StatusTracking isSupported="true">
        <Status>Tracking_State_1</Status>
    </StatusTracking>
    <RuntimeArguments isVariableArgumentsSupported="false" supportsRuntimeArgsChange="false">
        <UserDefinedPropertySheet isConfigurationRequired="false" name="com.fiorano.edbc.sampleedbc.model.SampleEDBCPM"/>
        <Argument isInMemorySupported="true" isProfiled="false"
            isRequired="true" name="m_arg_1" type="Integer">100</Argument>
        <Argument isInMemorySupported="true" isProfiled="false"
            isRequired="false" name="JVM_PARAMS" type="Text"/>
    </RuntimeArguments>
</Execution>
```

Figure 3.6.20: ServiceDescriptor.xml containing properties specified in the Execution Information Panel

3.6.1.6.6 Logging and Document Tracking Panel

Logging is a practice of writing out the messages indicating the state of component, actions performed and any other related data. Logging is used for different purposes including

- Notifying users of important actions/changes or problems (exceptions/errors) that occur at runtime
- Aiding developers in debugging the application
- Understanding the flow of data among different method calls

For each of the above purposes the data and the details that should be logged vary. Log levels help in meeting the needs of different users.

This panel allows the users to define such logging and states for document tracking (Figure 3.6.21).

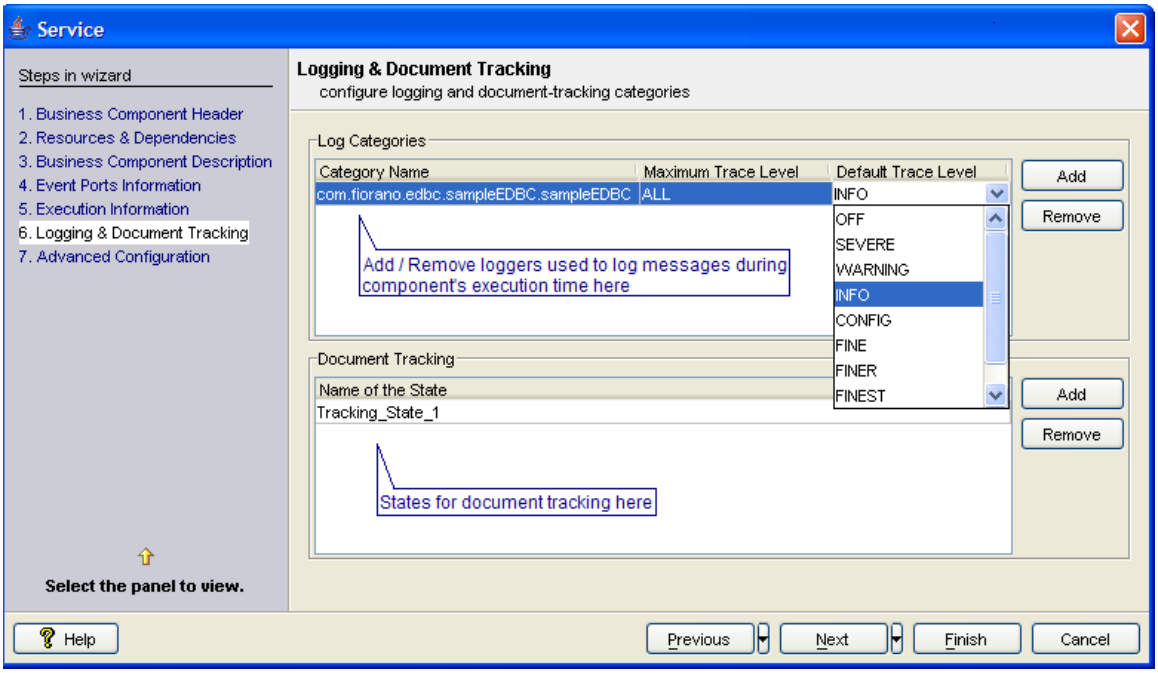


Figure 3.6.21: Configuring Logging and Document Tracking panel

```
<execution subtype="edbc">
        -
        -
        <logmodules>
                <logmodule name="ERR_HANDLER" level="WARNING" />
                <logmodule name="OUT_HANDLER" level="INFO" />
        </logmodules>
        -
        -
</execution>
```

Figure 3.6.22: ServiceDescriptor.xml containing properties specified in the Logging and Document Tracking Panel

3.6.1.6.7 Advanced Configuration Panel

This panel has the following properties:

Property	Description
Is auto launchable	Yes – Component can be launched in 'Separate Process' mode No – Component cannot launched in 'Separate Process' mode
Supports Error Handling	Yes – Error port is shown when 'Show Error Ports' is selected No – Error port is not shown even when 'Show Error Ports' is selected

Property	Description
Supports Failover to another Peer Server	Yes – When the Peer Server on which component is running goes down, the component keeps running on the next available Peer Server. No – When the Peer Server on which component is running goes down, the component will not keep running on the next available Peer Server.
Is manually launchable	Yes – Component can be launched in 'Manual' mode No – Component cannot launched in 'Manual' mode
Is inMemory launchable	Yes – Component can be launched in 'In Memory' mode No – Component cannot launched in 'In Memory' mode
InMemoryLaunch Class	A class which implements fiorano.esb.util.InMemoryLaunchable interface containing methods to launch and stop the component. This class is used by Peer Server to launch and stop component when the launch mode of the component is InMemory
Supports Component Control Protocol	Yes – Component listens, understands and responds to control events from Peer Server. Using this option allows components launched as separate process to cleanup when stopping No – Component does not handle control events from the Peer Server. The Peer Server will not send any control event to component. Component launched in separate process is issued a destroy command to stop and the component process will be killed instantly without any cleanup. This property will not editable while editing the service from Studio. For additional details on Component Control Protocol refer to section 3.12.
Generate Fiorano Installation as classpath variable	In the eclipse build files generated for the component, to generate fiorano home as a eclipse variable instead of a hardcoded path this option has to be selected. If selected, the developer has to define the variable 'FIORANO_HOME' and assign it to specific Fiorano Installation directory. The .classpath file will pickup the required libraries from the specified location.

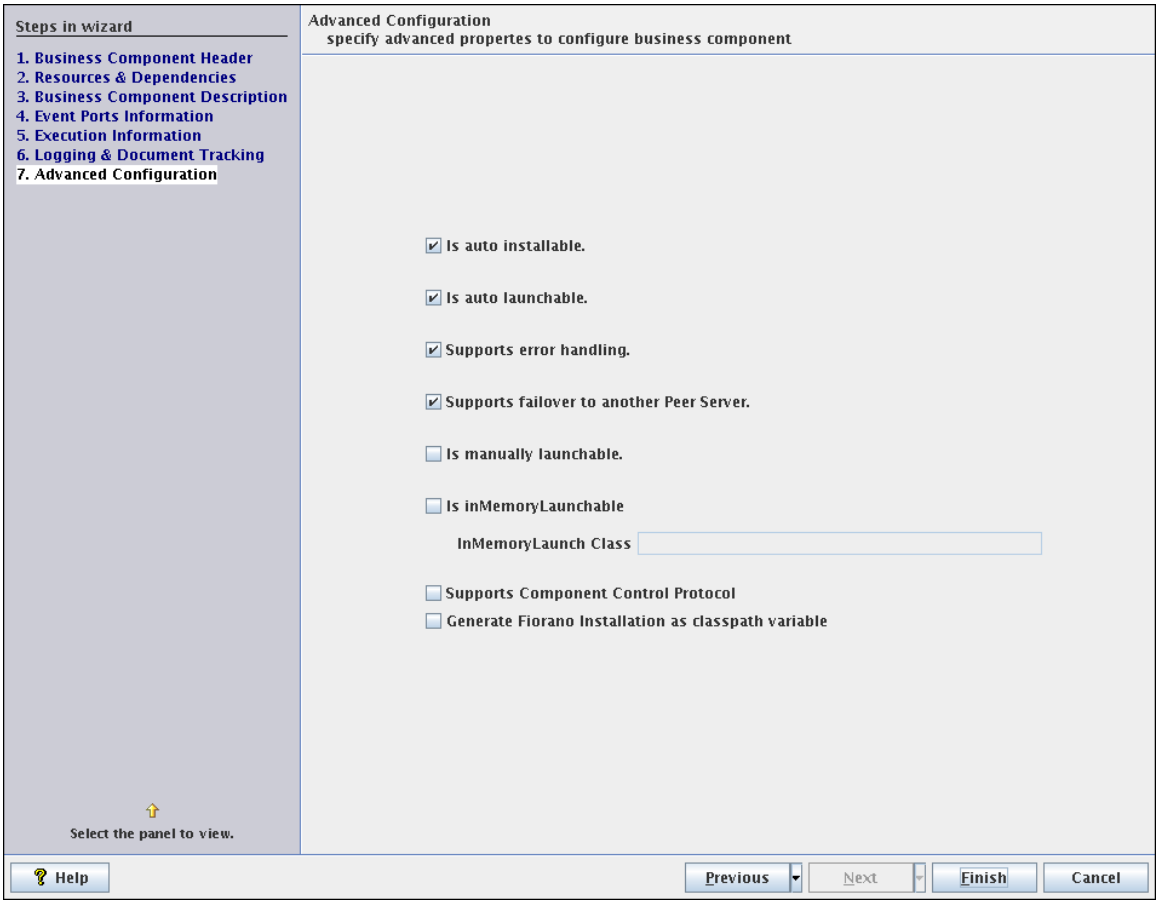


Figure 3.6.23: Advanced Configuration Panel

After the wizard is closed, it creates the following structure in the directory specified against the –dest option passed to the wizard.bat or wizard.sh.

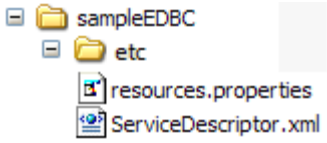


Figure 3.6.25: Directory structure after defining component

In case of synchronous Java Component a file named bc.properties is also created.

3.6.1.7 Generating Code for the Defined Component

To generate the source code for the component defined, use the generate command.

3.6.1.7.1 On Windows

- Browse to %TEMPLATE_ENGINE%
- Run templates-console.bat; a command prompt opens in directory pointed to %TEMPLATE_ENGINE%.

- Run the command generate.bat specifying the destination directory as follows:

```
usage:generate.bat [-language <name>] [-setting <name>] -dest <directory>
 -dest <directory>    target directory where source files are
                      generated
 -language <name>     launguage(c, cpp, csharp, java, jca) in
                      which source files are generated (default is java)
 -setting <name>      name of setting directory in etc folder
```

Figure 3.6.26: Run templates-console.bat

Note: Run generate.bat with out any arguments to see the help

3.6.1.7.2 On Linux

Browse to %TEMPLATE_ENGINE% in Linux console

Run the command generate.sh specifying the destination directory as follows:

usage: run [-language <name>] [-setting <name>] -dest <directory>

-dest <directory> target directory where source files are

Generated (same directory given while launching wizard)

-language <name>language (c, cpp, csharp, java, jca) in

which source files are generated (default is java)

-setting <name> name of setting directory in etc folder

Note: Run generate.sh with out any arguments to see the help

Executing the above command generates the sources under src directory specified in the directory specified against –dest argument. It also creates necessary files to build and deploy the components. build.properties, build.xml, common.xml are the common files that are created irrespective of language.

3.6.1.8 Building the Component

To build the component created, execute command ant in the component directory. This compiles the source files, builds the required archives and creates a file export.zip containing all the files /details required for the components. export.zip can be imported using FSSM or Studio, for details regarding importing of a business component see section 3.5 Export and Import Components.

3.6.1.9 Deploying the Component

Deploying the component means registering the component on a Fiorano Enterprise Server. Any component created before it can use in developing event process(s) has to be registered with the Fiorano Enterprise Server.

To deploy the developed component execute command **ant register** from the command line in the component directory.

The details of the enterprise server on which the component should deployed are specified in build.properties file

```
# location where FioranoSOA is installed
installer.dir=C:/fioranodev/HEAD_new_installer

# path to directory where distribution files are created
deploy.dir=build/dist

# FES server info
server=tsp_tcp\://localhost\:1947
user=admin
passwd=passwd

# uncomment following if you are using http proxy
#proxy.server=specifyProxyNameHere
#proxy.portt=80
#proxy.user=admin
#proxy.passwd=passwd
```

Figure 3.6.27: build.properties containing details of enterprise server on which component has to be deployed

Note: ant unregister command can be used to unrgister a custom component that is already registered with the enterprise server.

ant reregister command can be used to redeploy a component that is already registered with enterprise server in case any changes are made to the component.

3.6.2 Component Creation in Fiorano Studio

Fiorano provides a complete GUI based approach to define, build and deploy components from Fiorano Studio, apart from the scripts based approach from command line mentioned in the earlier section.

- The GUI for component creation can launched from Fiorano Studio →Tools → Create Service Component action (shown in Figure 3.6.28).

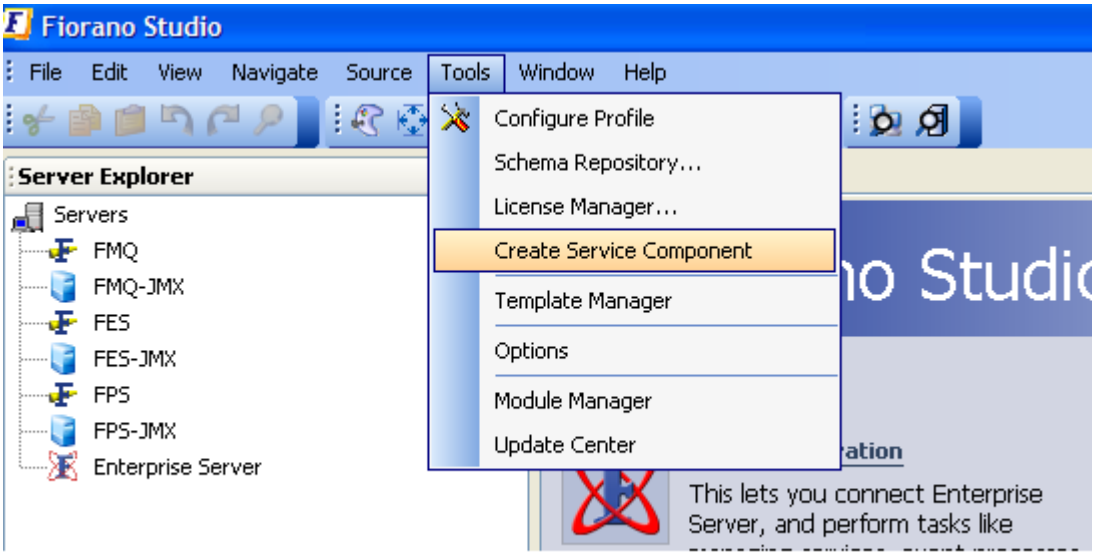


Figure 3.6.28: Menu containing Create Service Component action

- Invoking this action (click on the menu item) brings up a dialog (Figure 3.6.29). The dialog takes the destination folder in which the component has to be created. Note that the folder name given should not be existent.

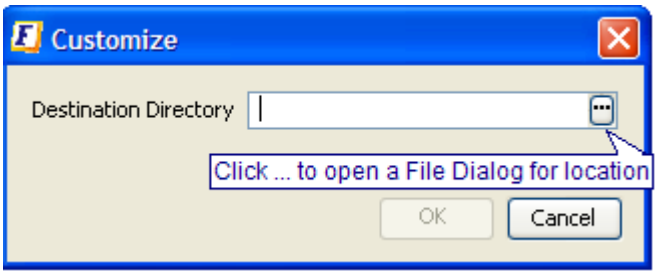


Figure 3.6.29: Input for destination folder in which the component has to be created.

- Provide a valid directory and click **OK** button.
- Define the component that is to be created comes up.
- Complete the component definition. On the completion of wizard, the **Customize** dialog comes up as shown in Figure 3.6.30.

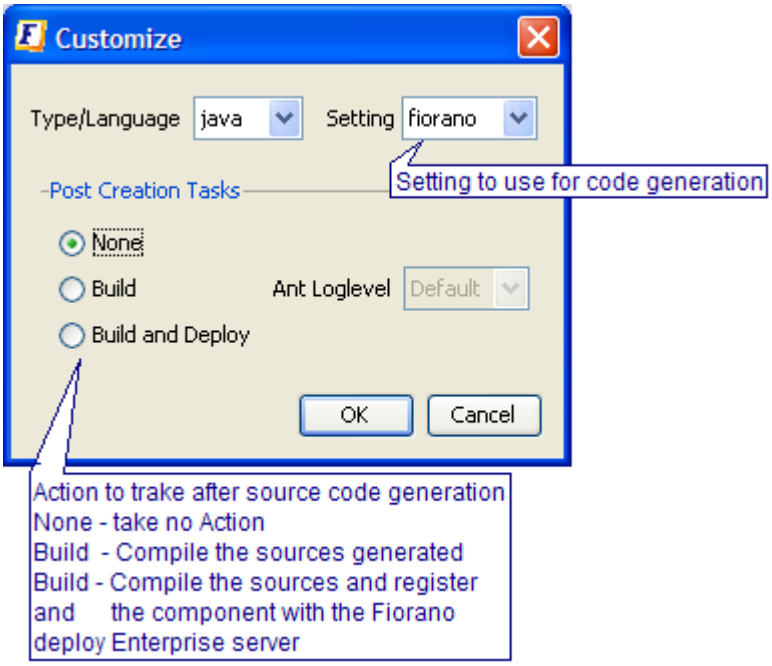


Figure 3.6.30: Component code generation, compilation and registration

Select the language, setting and post creation task (among the following tasks)

Post Creation Action	Action taken
None	Generate the source code and do nothing else
Build	Generate the source and build the sources
Build and Deploy	Generate the source, build the source and register with Fiorano Enterprise Server

3.6.3 Java Components

Java components can be of two types as described in section 3.1 Service Components. Characteristics:

- Asynchronous components (pure JMS)
- Synchronous components (JCA-Compliant)

3.6.3.1 Defining Asynchronous Component

To define an asynchronous components launch the component definition wizard as described in section 3.6.1.5 Defining Components and follow the steps described in section 3.6.1.6 Getting familiar with wizard and service configuration.

Example: wizard.bat –dest C:\SampleComponents\EDBC\SampleEDBC

3.6.3.1.1 Generating Code for Asynchronous Component

To generate code for asynchronous component follow the steps in section 3.6.1.7 Generating code for the defined component.

The generate command by default assumes that the code is generated for the asynchronous component. To particularly specify that the code generated should be for asynchronous component use –language option with value java

Example: generate.bat –dest C:\SampleComponents\EDBC\SampleEDBC –language java

The generated component code has the structure as shown in Figure 3.6.31

Figure 3.6.31: Structure of asynchronous component

Description/purpose for each of the files generated are also shown in Figure 3.6.31.

3.6.3.1.2 Adding business logic to asynchronous Component

By default the component functions are:

- Listens for the input message on the input port.
- Sends the output to JMSReplyTo if request reply is enabled on the input port which receives this message and JMSReplyTo is specified.

- If request reply is not enabled or JMSReplyTo is not specified then it sends the same message on all the output ports.

To add the business logic

First the control flow should be understood. The control flow works as follows

Component startup:

- Component's 'Executable' class (present in ServiceDescriptor.xml) is invoked with required arguments in case of 'separate process' launch. Or, startup (String [] args) method of the component's 'InMemoryLaunchImpl' class (present in ServiceDescriptor.xml) passing the arguments.
- Arguments are parsed. See figure 3.6.32.

```
/**
 * Main method is used to create an object of service class and calls a method for startup in external launch
 * @param args command
 */
public static void main(String args[]){
    try{
        SampleEDBC service = new SampleEDBC();
        service.common_startup(args);
    }catch (Exception ex){
        ex.printStackTrace();
    }
}

    /**
 * This method ent when it is configured to launch in inMemory
 */
public void startup(String [] args){
    try{
        common_startup(args);
    }catch (Exception ex){
        ex.printStackTrace();
    }
}
/**
 * Gets the component configuration information from the property model(PM) class and calls a method
 * to create JMS objects
 */
public void common_startup(String[] args) throws Exception {
        m_cmdLineArguments = new CommandLineParams(args);
        m_model = getConfiguration();
        createJMSObjects();
}
```

External launch control enters here

Arguments are passed to the component

InMemory launch control enters here

Arguments are parsed

Figure 3.6.32: `SampleEDBC.java` showing the control flow entry point and the args passed

- The configuration of the component is serialized and bound using JNDI during the configuration time. This serialized configuration is looked up during the start up and de-serialized to get the configuration object. See Figure 3.6.33.

```
/**
 * Loads Configuration object
 * @return configuration object
 */
private SampleEDBCPM getConfiguration() {
    String configXML = null;
    SampleEDBCPM configuration = null;

    try {
        // Lookup configuration object
        configXML = (String) _lookup(m_cmdLineArguments.getConfigName());
    } catch (NamingException ex) {
        configuration = new SampleEDBCPM();
        return configuration;
    }

    try {
        configuration = (SampleEDBCPM) BeanUtils.deserialiseBean(configXML);
    } catch (Throwable ex) {
        ex.printStackTrace();
        configuration = new SampleEDBCPM();
    }

    if (m_logger.isLoggable(Level.INFO))
        m_logger.log(Level.INFO, "Got Configuration " + configXML);

    return configuration;
}

/**
 * Lookup object from FMQ JNDI
 * @param objectName  used for lookup
 * @return object after lookup
 * @throws javax.naming.NamingException
 */
public Object _lookup(String objectName) throws NamingException {
    return m_cmdLineArguments.lookupObject(objectName);
}
```

Deserialize the configuration

lookup serialized configuration

Figure 3.6.33: `SampleEDBC.java`, JNDI lookup and deserialize the configuration

All the required JMS objects are created.

Component runtime:

- Once the component startup is complete, the component is ready to process the messages.
- Each of the input ports of the component is associated with a MessageListener (<Component_guid>MessageListner.java), which listens for the messages on the respective input port.
- On receiving a message, onMessage (Message msg) of MessageListener is called.
- The message is processed and sent on the output port(s) or JMSReplyTo destination as appropriate

```
/**
 * Processes the input request. By default it returns the input message.
 * @param request request string
 * @return response message.
 * @throws ServiceExecutionException if there is any exception in processing the request
 */
public String process(String request) throws ServiceExecutionException {
    logger.log(Level.INFO, RBUtil.getMessage(Bundle.class, Bundle.REQUEST_PROCESSED, new Object[]{request}));
    return request;
}

/**
 * Processes the input request. By default it returns the input object.
 * @param request request object
 * @return response object.
 * @throws ServiceExecutionException if there is any exception in processing the request.
 */
public Object process(Object request) throws ServiceExecutionException {
    logger.log(Level.INFO, RBUtil.getMessage(Bundle.class, Bundle.REQUEST_PROCESSED, new Object[]{request}));
    return request;
}
```

Add business logic here

Add business logic here and override onMessage(Message) in SampleEDBCMessageListener

Figure 3.6.34: RequestProcessor.java showing the business logic

```
/**
 * Processes the input request. By default it returns the input message.
 * @param request request string
 * @return response message.
 * @throws ServiceExecutionException if there is any exception in processing the request
 */
public String process(String request) throws ServiceExecutionException {
    logger.log(Level.INFO, RBUtil.getMessage(Bundle.class, Bundle.REQUEST_PROCESSED, new Object[]{request}));
    return request;
}

/**
 * Processes the input request. By default it returns the input object.
 * @param request request object
 * @return response object.
 * @throws ServiceExecutionException if there is any exception in processing the request.
 */
public Object process(Object request) throws ServiceExecutionException {
    logger.log(Level.INFO, RBUtil.getMessage(Bundle.class, Bundle.REQUEST_PROCESSED, new Object[]{request}));
    return request;
}
```

Add business logic here

Add business logic here and override onMessage(Message) in SampleEDBCMessageListener

Figure 3.6.35: SampleEDBCMessageListener.java containing logic to pass Object to request processor

Component shutdown:

When the application / component is killed the component's process is killed and the component is forcibly killed. However, if the component is launched in memory the shutdown (Object hint) is called. All the JMS objects, connections and so on are closed here.

For the generated component all of the above work is done and only the business logic needs to be added. Business logic includes creating any necessary objects or connections to external systems required for processing the messages, processing messages and cleaning up the objects, created at startup, during the shutdown. Business logic can be added in processMessage(Message message) method of<component_guid> MessageListener .java as shown in Figure 3.6.34.

3.6.3.1.3 Configuration Property Sheet of asynchronous component

The generated component by default has a Configuration Property Sheet (CPS), which can used to configure the adapter to perform required business logic. See section 3.6.3.2.7 Adding properties to Configuration Property Sheet to add additional properties on the CPS.

In addition to getters and setters and the jmx-operations described in the section 3.6.3.2.7 Adding properties to Configuration Property Sheet, asynchronous components configuration class can have the following methods (jmx operations)

```
/**

* @return

 * @jmx.managed-operation description="tests the connection or business logic"

 */

public void test() throws FioranoException
```

Add the logic to test any connection related or business operation related logic here. See WorkList component's CPS

```
/**

 * @return help url

 * @jmx.managed-operation description="Help set URL"

 */

public URL fetchHelpSetURL()
```

URL to the helpset file to display help for the component in the CPS

```
/**

 * @exception FioranoException

 * @jmx.managed-operation description=" Validates Configuration Properties "

 */

public void validate() throws FioranoException
```

Add logic to validate the component's configuration in this method. If the configuration is invalid throw a FioranoException to disallow user to finish the CPS.

3.6.3.1.4 Frequently Asked Questions

Question: What are runtime arguments?

Answer: Runtime arguments are name value pairs which are passed to a component as command line arguments during component launch.

Component's configuration is usually and preferably a part of component Custom Property Sheet (CPS). However, some configuration parameters may be passed as runtime arguments to the component in following cases:

- Component requires very few configuration parameters (typically up to 3).

- Configuration contains properties which may be changed very frequently and opening CPS every time should be avoided.

Question: How to access runtime arguments?

Answer: Runtime arguments are just another way to provide component configuration. Runtime arguments are accessible in ${ServiceGUID}.java as commandLineParams. To access runtime arguments in other classes, typically in classes which process request, following changes have to be done.

- **${ServiceGUID}PM.java** contains configuration for the component and this object is available with all the classes which process requests
- Create a getter and setter for an object of type **CommandLineParameters** in **${ServiceGUID}PM.java** as shown in Figure 3.6.36.

```
private CommandLineParameters commandLineParameters;

public CommandLineParameters getCommandLineParameters() {
    return commandLineParameters;
}

public void setCommandLineParameters(CommandLineParameters commandLineParameters) {
    this.commandLineParameters = commandLineParameters;
}
```

Figure 3.6.36: ${ServiceGUID}PM.java with getter and setter for runtime arguments

1. Override method **fetchConfiguration()** in **${ServiceGUID}.java** to set instance of **CommandLineParameters (commandLineParams)** on instance of **${ServiceGUID}PM.java** (configuration) as shown in Figure 3.6.37.

```
protected void fetchConfiguration() throws ServiceExecutionException {
    super.fetchConfiguration();
    ((SampleEDBCPM)configuration).setCommandLineParameters(commandLineParams);
}
```

Figure 3.6.37: Setting runtime arguments onto configuration in ${ServiceGUID}.java

2. Where ever required, fetch value of runtime arguments' from configuration object. As shown in Figure 3.6.38. Individual parameter objects can be accessed using **getParameter(String key)**

```
public RequestProcessor(Logger logger, IServiceConfiguration serviceConfiguration) {
    super(null, logger, serviceConfiguration);
    commandLineParameters = ((SampleEDBCPM) serviceConfiguration).getCommandLineParameters();
}

private void addMessageToCache(Message request) {
    if(SampleEDBCPM.FIFO.equals(getParamter("Order")))
        messages.add(request);
    else
        messages.add(0,request);
}

private String getParamter(String key) {
    return (String) commandLineParameters.getParameter(key);
}
```

Figure 3.6.38: Accessing runtime argument in RequestProcessor.java

Question: How to send messages onto specific output port(s)?

Answer: EDBC framework has support for following methods on JMSHandler class:

```
public void sendMessage(Message outputMsg) throws JMSException

public void sendMessage(Message outputMsg, Destination sendDest) throws JMSException

public void sendMessage(Message outputMsg, Destination sendDest, int deliveryMode, int priority, long ttl)

public void sendMessage(Message outputMsg, String outputPortName) throws JMSException
```

Figure 3.6.39: APIs for sending messages in JMSHandler

Sending messages only onto specific output port(s) can be done in one of the following methods:

Method 1: Determining output port while sending

Generated code uses sendMessage(Message outputMsg)API to send messages on all related output ports. To send messages only to a specific port use sendMessage(Message outputMsg, String outputPortName)as shown in Figure 3.6.40.

```
protected void sendResponse(Message message) throws ServiceExecutionException {
    try {
        jmsHandler.sendMessage(message, "OUT_PORT_2");
    } catch (JMSException e) {
        throw new ServiceExecutionException(e, ServiceErrorID.RESPONSE_GENERATION_ERROR);
    }
    getLogger().log(Level.INFO, RBUtil.getMessage(Bundle.class, Bundle.MESSAGE_SENT_SUCCESSFULLY,
}
```

Figure 3.6.40: Sending message to required output port in ${ServiceGUID}MessageListener.java

Method 2: Creating association between input ports and output ports

This method can be used when there is a strict one-to-one binding input and output ports.

Example: Consider a case where a component has two input ports (IN_PORT_1 and IN_PORT_2) and two output ports (OUT_PORT_1 and OUT_PORT_2) and all message received on IN_PORT_1 should sent on OUT_PORT_1 and all messages received on IN_PORT_2 should be sent on OUT_PORT_2.

Then the following can be done by passing a collection of only required OutputPortHandler objects when creating InputPortHandler in JMSObjects as shown in Figure 3.6.41.

```
protected AbstractInputPortHandler createInputPortHandler(Destination destination,
                                                          InputPortInstanceAdapter inputPortInstanceAdapter,
                                                          Collection outputPortHandlers,
                                                          EventGenerator eventGenerator, Session eventSession) {
    Collection outputPort = new ArrayList(1);
    for (Iterator iterator = outputPortHandlers.iterator(); iterator.hasNext();) {
        OutputPortHandler outputPortHandler = (OutputPortHandler) iterator.next();
        if("IN_PORT_1".equals(inputPortInstanceAdapter.getName())) {
            if("OUT_PORT_1".equals(outputPortHandler.getOutputPortInstanceAdapter().getName())) {
                outputPort.add(outputPortHandler);
            }
        } else {
            if("OUT_PORT_2".equals(outputPortHandler.getOutputPortInstanceAdapter().getName())) {
                outputPort.add(outputPortHandler);
            }
        }
    }
    return new InputPortHandler(destination, inputPortInstanceAdapter, outputPort, eventGenerator, eventSession);
}
```

Figure 3.6.41: Associating output ports with input ports in JMSObjects.java

Question: How to release resources in RequestProcessor class?

Answer: RequestProcessor is designed to handle requests and should usually contain only logic for handling requests. Responsibility of passing required resources to RequestProcessor lies usually with the class creating RequestProcessor object as shown in StoreNForward sample provided in %FIORANO_HOME%\esb\samples\components\EDBC\StoreNForward.

Number of RequestProcessor objects in the component also largely depends on component implementation. For the generated if resources are being initialized in RequestProcessor class, then they can release by making following changes:

Add a method close() in RequestProcessor which does the required cleanup.

```
public void close() {
    //todo: release resources here
}
```

Figure 3.6.42: close method to release resources in RequestProcessor.java

Cache ${ServiceGUID}MessageListener class in JMSHandler.java and provide a close() method which does necessary cleanup.

```
    private SampleEDBCMessageListener messageListener;

    protected MessageListener createMessageListener(ServiceExceptionHandler serviceExceptionHandler) {
        return messageListener = new SampleEDBCMessageListener(this, serviceExceptionHandler);
    }

    public void close() {
        //todo: add any cleanup logic here
        ((RequestProcessor)messageListener.getRequestProcessor()).close();
    }
```

Figure 3.6.43: Caching ${ServiceGUID}MessageListener and releasing resources in JMSHandler.java

Add a close() method in InputPortHandler.java to do necessary cleanup and close all JMSHandler objects

```
public void close() {
    //todo: do any additional cleanup here
    AbstractJMSHandler[] jmsHandlers = getJmsHandlers();
    for (int i = 0; i < jmsHandlers.length; i++) {
        JMSHandler jmsHandler = (JMSHandler) jmsHandlers[i];
        jmsHandler.close();
    }
}
```

Figure 3.6.44: close() to close all JMSHandler objects in InputPortHandler.java

Override destroy() method in JMSObjects.java to close InputPortHandlers

```
public void destroy() {
    Collection inputPortHandlers = getInputPortHandlers();
    for (Iterator iterator = inputPortHandlers.iterator(); iterator.hasNext();) {
        InputPortHandler inputPortHandler = (InputPortHandler) iterator.next();
        inputPortHandler.close();
    }
    super.destroy();
}
```

Figure 3.6.45: closing InputPortHandler objects in JMSObjects.java

Question: How to use a custom CPS instead of default CPS after the component creation?

Answer: By default, the component created from generated sources has a dynamically created name-value pair CPS. Additional properties can be added to CPS as described in section 3.6.3.2.7 Adding properties to Configuration Property Sheet (CPS).

However, if a manually written CPS class (an instance of TifosiCustomPropertySheet or PropertyEditorSupport) from earlier SOA version is present then it can used instead of the default dynamically created CPS as shown in Figure 3.6.46.

Edit etc\ServiceDescriptor.xml to use the complete class name for required CPS class

(Say, com.fiorano.edbc.sampleedbc.cps.PropertySheet) instead of default CPS class name (com.fiorano.edbc.${serviceguid}.model.${ServiceGUID}PM) at /service/execution/cps/launcher as shown in Figure 3.6.46:

```
<service guid="sampleEDBC" version="1.0"><display name="sampleEDBC" categorie
"06-12-2007 02:37:04" last-modified="06-12-2007 02:36:29" authors="admin" />
="4.0" />><serviceref guid="EDBCEngine" version="4.0" />><serviceref guid="jms"
supported="7" />><separate-process executable="$executable" />><inmemory execut
<cps><launcher>com.fiorano.edbc.sampleedbc.cps.PropertySheet</launcher></cps>
Business Component Creation Wizard.</description></port></input-ports><output
Component Creation Wizard.</description></port></output-ports><logmodules><lo
"INFO" />></logmodules><runtime-arguments><runtime-argument name="JVM_PARAMS"
</value></runtime-argument></runtime-arguments></execution></service>
```

Figure 3.6.46: ServiceDescriptor.xml showing the change in CPS class name

Edit common.xml file to provide new CPS class name in service-export ant task (present in the target deploy) as shown in Figure 3.6.47.

```
<target name="deploy" depends="module.deploy">
    <service-export
            destfile="export.zip"
            servicefile="etc/ServiceDescriptor.xml"
            resourcesfile="etc/resources.properties"
            executionClass="com.fiorano.edbc.sampleedbc.SampleEDBC"
            cpsClass="com.fiorano.edbc.sampleedbc.cps.PropertySheet"
            cpsMandatory="true"
            inMemoryClass="com.fiorano.edbc.sampleedbc.SampleEDBC"
            defaultLogModule="com.fiorano.edbc.sampleedbc.sampleEDBC">
        <fileset dir="${deploy.dir}">
            <include name="${jar.path}"/>
        </fileset>
    </service-export>
</target>
```

Figure 3.6.47: common.xml, service-export ant task showing change in CPS name

Question: How to use a custom service class?

Answer: Similar to CPS class above, if a pre-existing service class has to be used instead of default generated service class ${ServiceGUID}.Note that the class should implement InMemoryLaunchable interface.

Edit common.xml file to provide new service class name in service-export ant task (present in the target deploy) as shown in Figure 3.6.48.

```
<service-export
        destfile="export.zip"
        servicefile="etc/ServiceDescriptor.xml"
        resourcesfile="etc/resources.properties"
        executionClass="com.fiorano.edbc.sampleedbc.SampleEDBC"
        cpsClass="com.fiorano.edbc.sampleedbc.cps.SampleEDBCPropertySheet"
        cpsMandatory="true"
        inMemoryClass="com.fiorano.edbc.sampleedbc.SampleEDBC"
        defaultLogModule="com.fiorano.edbc.sampleedbc.sampleEDBC">
    <fileset dir="${deploy.dir}">
        <include name="${jar.path}"/>
    </fileset>
</service-export>
```

Figure 3.6.48: common.xml, service-export ant task showing change in service name

Note: defaultLogModule value should also be changed to reflect correct value(new service class name being used)

Question: How to create a component handling asynchronous requests?

Answer: Default generated sources handle requests synchronously, i.e. for each input there is atleast one corresponding output. However, asynchronous request handling can be achieved by making following changes in the generated source code.

Modify the constructor of RequestProcessor to pass a handle to JMSHandler

```
private JMSHandler jmsHandler;

/**
 * creates an instance of Request Processor.
 * @param schema If there's any schema on the input port to be validated.
 */
public RequestProcessor(ESBRecordDefinition schema, JMSHandler jmsHandler) {
    super(schema, jmsHandler.getLogger(), jmsHandler.getServiceConfiguration());
    this.logger = jmsHandler.getLogger();
    this.jmsHandler = jmsHandler;
}
```

Figure 3.6.49: Modifying RequestProcessor.java to pass handle for JMSHandler in constructor

Override sendResponse(Message message)and prepareResponse(Message requestMessage, String response) in ${ServiceGUID}MessageListener with empty implementations.

```
protected void sendResponse(Message message) throws ServiceExecutionException {
}

protected Message prepareResponse(Message message, String string) throws ServiceExecutionException {
    return null;
}
```

Figure 3.6.50: Empty implementations in ${ServiceGUID}MessageListener.java

Pass reference of JMSHandler to RequestProcessor

```
public SampleEDBCMessageListener(JMSHandler jmsHandler, ServiceExceptionHandler exceptionHandler) {
    this.jmsHandler = jmsHandler;
    ESBRecordDefinition recordDefinition = jmsHandler.getInputPortHandler().getInputPortInstanceAdapter().getSchema();
    requestProcessor = new RequestProcessor(recordDefinition, jmsHandler); //passing JMSHandler reference
    this.exceptionHandler = exceptionHandler;
}
```

Figure 3.6.51: Passing JMSHandler to RequestProcessor from ${ServiceGUID}MessageListener.java

Return null in process (String request) in RequestProcessor

```
public String process(String request) throws ServiceExecutionException {
    logger.log(Level.INFO, RBUtil.getMessage(Bundle.class, Bundle.REQUEST_PROCESSED, new Object[]{request}));
    return null; //return null for asyncronous requests
}
```

Figure 3.6.52: Return null after processing request in RequestProcessor.java

When required condition is satisfied send the message from RequestProcessor process(String request) to required output port using JMSHandler as described in FAQ 3 above.

Question: How to access a property set on the incoming JMS Message?

Answer: Generated code calls RequestProcessor process(String request) for processing message. However, message properties from input message can be accessed in RequestProcessor by changing generated code as follows:

Override handleMessage(Message requestMessage) in ${ServiceGUID}MessageListener to call Message process(Message) API in RequestProcessor as shown in Figure 3.6.53.

```
public void handleMessage(Message requestMessage) throws ServiceExecutionException {
    IRequestProcessor requestProcessor = getRequestProcessor();
    Message response;
    String request = null;
    try {
        request = MessageUtil.getTextData(requestMessage);
    } catch (JMSException e) {
        throw new ServiceExecutionException(RBUtil.getMessage(com.fiorano.edbc.framework.service.jms.Bundle.class,
                                            Bundle.FAILED_TO_FETCH_REQUEST), e, ServiceErrorID.TRANSPORT_ERROR);
    }
    if (requestProcessor != null) {
        try {
            requestProcessor.validate(request);
        } catch (ServiceExecutionException e) {
            handleException(e, requestMessage);
        }
        response = requestProcessor.process(requestMessage);     //change here to use Message process(Message) API
    } else {
        response = requestMessage;
    }
    sendResponse(response);
}
```

Figure 3.6.53: Overriding handleMessage in ${ServiceGUID}MessageListener.java

Override Message process (Message) method in RequestProcessor as shown in Figure 3.6.54.

```
public Message process(Message request) throws ServiceExecutionException {
    logger.log(Level.INFO, RBUtil.getMessage(Bundle.class, Bundle.REQUEST_PROCESSED
    String sampleProperty = null;
    try {
        sampleProperty = request.getStringProperty("SampleProperty");
    } catch (JMSException e) {
        logger.log(Level.INFO, RBUtil.getMessage(Bundle.class, Bundle.ERROR_IN_GETT
    }
    return request;
}
```

Figure 3.6.54: Overriding API to handle Message in RequestProcessor.java

Note: StoreNForward sample provided in %FIORANO_HOME%\esb\samples\components\EDBC\StoreNForward shows handling requests asynchronously and using API which handle Message instead of String

Question: How is exception handling done?

Answer: Exception handling in generated code can be easily done by just setting predefined error codes when exceptions are thrown. For each error code, a set actions can be taken (if enabled). These actions can be configured in the CPS generated.

Following error codes are supported by EDBC framework:

- ServiceErrorID.INVALID_REQUEST_ERROR
- ServiceErrorID.REQUEST_EXECUTION_ERROR
- ServiceErrorID.TRANSPORT_ERROR

If an error happen when a request is being processed, throw a ServiceExecutionException with appropriate error code (as shown in the Figure 3.6.55) and the framework handles it based on the actions enabled in CPS.

```
private void retrieveMessageFromCacheAndSend(Message request) throws ServiceExecutionException {
    if(messages.size() > 0) {
        try {
            messageSender.send((Message) messages.remove(0),((TextMessage)request).getText());
        } catch (JMSException e) {
            throw new ServiceExecutionException(e, ServiceErrorID.REQUEST_EXECUTION_ERROR);
        }
    }
}
```

Figure 3.6.55: Throwing a ServiceExecutionException with appropriate error code

Question: How can resources be initialized or cleaned up in service class?

Answer: Override createServiceObjects() and stop()in ${ServiceGUID}.java to handle resource initialization and resource cleanup respectively

```
protected void createServiceObjects() throws ServiceExecutionException {
    //todo: initialize resources here
}

public void stop() {
    //todo: cleanup resources here
    super.stop();
}
```

Figure 3.6.56: Handling resource initialization and cleanup in ${ServiceGUID}.java

Question: How can the configuration be made optional for service?

Answer: Configuration can be made optional for a service by making following changes:

Edit common.xml file and change value of cpsMandatory property to false in service-export ant task (present in the target deploy) as shown in Figure 3.6.57.

```
<target name="deploy" depends="module.deploy">
    <service-export
            destfile="export.zip"
            servicefile="etc/ServiceDescriptor.xml"
            resourcesfile="etc/resources.properties"
            executionClass="com.fiorano.edbc.storenforward.StoreNForward"
            cpsClass="com.fiorano.edbc.storenforward.cps.StoreNForwardPropertySheet"
            cpsMandatory="false"
            inMemoryClass="com.fiorano.edbc.storenforward.StoreNForward"
            defaultLogModule="com.fiorano.edbc.storenforward.storeNForward">
        <fileset dir="${deploy.dir}">
            <include name="${jar.path}"/>
        </fileset>
    </service-export>
</target>
```

Figure 3.6.57: Modfiying common.xml to mark configuration as optional

- Override isConfigurationMandatory() in ${ServiceGUID} to return false
- Override createDefaultServiceConfiguration() in ${ServiceGUID} to return default configuration

```
/**
 * Specifies whether the configuration is mandaory or not.
 * @return true - if configuration is mandatory.
 *         false - if configuration is not mandatory
 */
protected boolean isConfigurationMandatory() {
    return false;
}

/**
 * Creates a new Property Model object
 */
protected void createDefaultServiceConfiguration() {
    //default configuration to use when configuration is not mandatory
    configuration = new SampleEDBCPM();
}
```

Figure 3.6.58: Overriding required methods to make configuration optional in ${ServiceGUID}.java

3.6.3.2 Creating a Synchronous Component

Fiorano provides a framework to develop synchronous components. The user creating a synchronous component cannot and will not have to, make any coding effort to create or handle JMS objects. Handling of JMS objects is taken care of in the framework provided.

- Synchronous components are JCA-Complaint (1.0 spec) and hence user needs to implement a few interfaces on top of the framework if the synchronous component has to be created from scratch. However, to make it easier the template engine generates the implementation of JCA interfaces as well and all that the user would be required to handle is the business logic, like in case of asynchronous component.

3.6.3.2.1 Defining synchronous Component

To define a synchronous component launch the component definition wizard as described in section 3.6.1.5 Defining Components.

Example: wizard.bat –dest C:\SampleComponents\BC\SampleBC

- Fiorano framework for synchronous components allows the only one input port and one output port and it handles the launch / stop of the component automatically. Hence, the synchronous component definition wizard contains only the first three panels described in section 3.6.1.6 Getting familiar with wizard and service configuration. Refer to Figure 3.6.59.

In the **Business Component Header** panel select the component type as **BC**. On selecting BC, a check box **Enable Connection Semantics** appears. This check box has to be selected if the component makes a connection to external system. Enabling connection semantics also allows the component to pool the connections and reuse them. Refer to Figure 3.6.59.

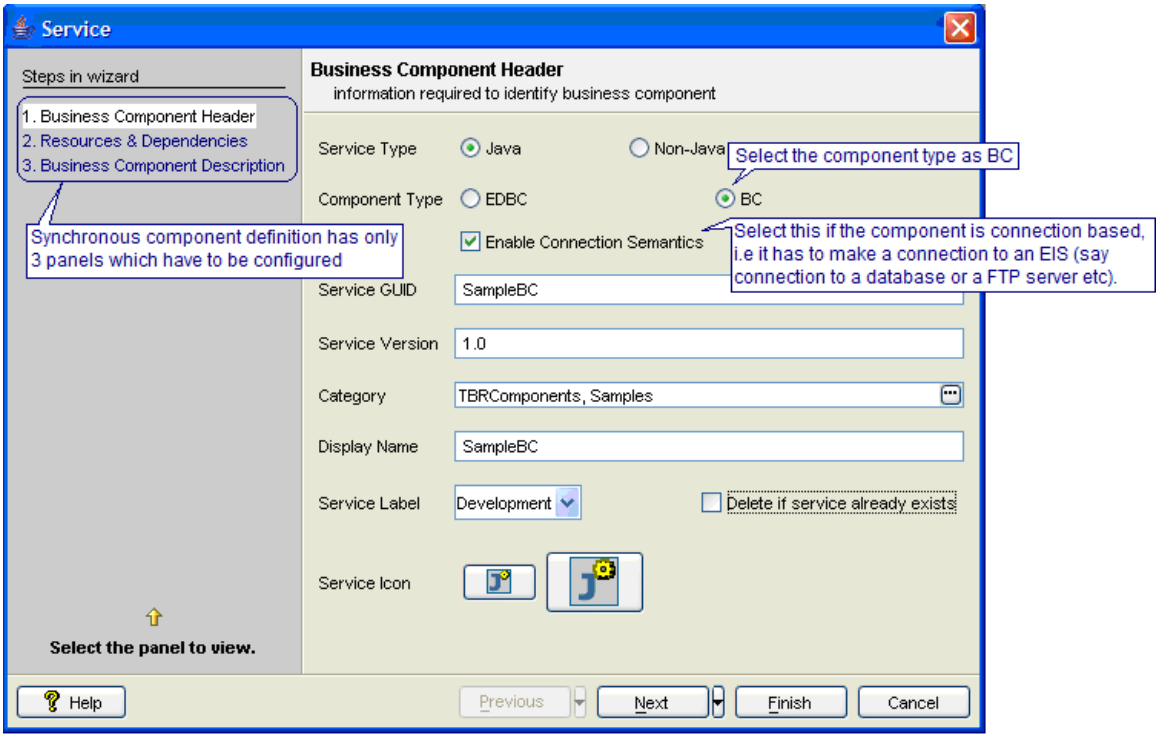


Figure 3.6.59: Business Component Header Panel for synchronous component

- **Resources and Dependencies** panel and **Business Component Description** panel can be configured as described in section 3.6.1.6 Getting familiar with wizard and service configuration.

3.6.3.2.2 Generating Code for synchronous Component

To generate code for synchronous component follow the steps as described in section 3.6.1.6 Getting familiar with wizard and service configuration specifying –language option with jca value as shown below:

Example: generate.bat –dest C:\SampleComponents\EDBC\SampleEDBC –language jca

The generated component code has the structure as shown in Figure 3.6.60.

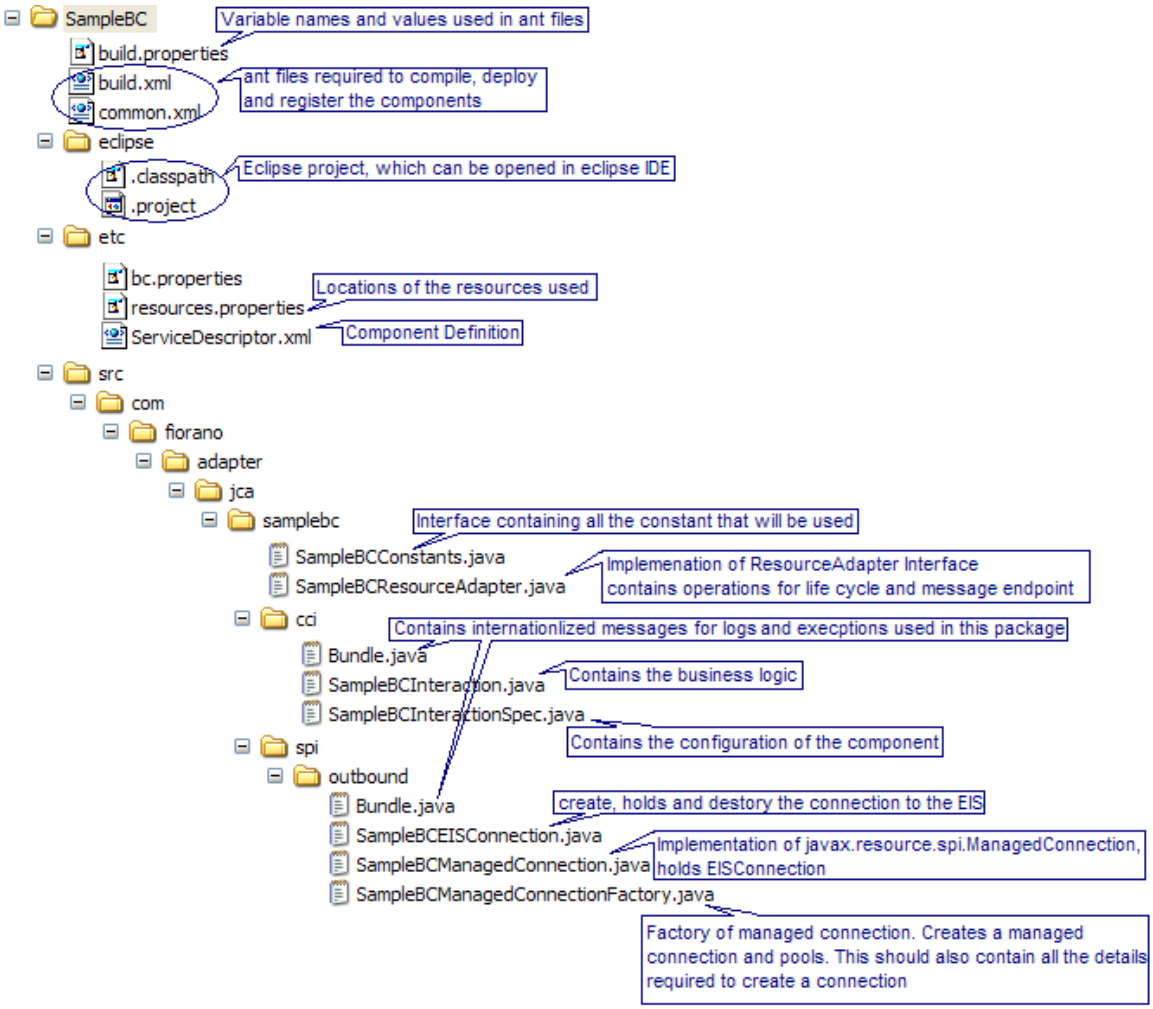


Figure 3.6.60: Structure of synchronous component

3.6.3.2.3 Adding Business Logic to Synchronous Component

The business logic of synchronous component lies in the <component_guid>Interaction.java. The methods and their purpose /description are given below:

```
String execute (ESBInteractionSpec interactionSpec, String inputMessage)
```

This method contains the business logic of the component.

Parameters:

- **interactionSpec –** the configuration object; holds all the details specified during the configuration time of the component in the Configuration Property Sheet (CPS).
- **inputMessage –** the inputMessage received on the component's input port

Return value:

Returns the string which should be sent as output message on the output port of the component.

```
ESBRecordDefinition getInputRecordDefinition(ESBInteractionSpec interactionSpec)
```

This method sets the schema (XSD/DTD) of the expected input message on the input port of the component

Parameters:

- interactionSpec **–** the configuration object; holds all the details specified during the configuration time of the component in the Configuration Property Sheet (CPS).

Return value:

Returns a ESBRecordDefinition object which contains the schema details like, the structure, structure type (XSD/DTD), root element, targetnamespace if the structure is a XSD and the imported XSDs if any in case the structure type is XSD.

```
ESBRecordDefinition getOutputRecordDefinition(ESBInteractionSpec interactionSpec)
```

This method sets the schema (XSD/DTD) of the expected output message on the output port of the component

Parameters:

- interactionSpec **–** the configuration object; holds all the details specified during the configuration time of the component in the Configuration Property Sheet (CPS).

Return value:

Returns a ESBRecordDefinition object which contains the schema details like, the structure, structure type (XSD/DTD), root element, targetnamespace if the structure is a XSD and the imported XSDs if any in case the structure type is XSD.

```
/**
 *  Contains the execution logic for this component.
 *
 *@param  interactionSpec        interactionspec object
 *@param  inputMessage           input message
 *@return                        output message
 *@exception  ResourceException  Business logic here cessing input
 */
public String execute(ESBInteractionSpec interactionSpec, String inputMessage)
         throws ResourceException
{
    m_logger.log(Level.FINER, Bundle.INPUT_MESSAGE, inputMessage);
    m_logger.log(Level.FINE, Bundle.PROCESSING_MESSAGE);
    /**
     *@todo  Implement the processing logic of input. Currently the output is same as that of input.
     */
    String responseMessage = inputMessage;
    m_logger.log(Level.FINER, Bundle.OUTPUT_MESSAGE, responseMessage);

    return responseMessage;
}
/**
* Returns input record definition value
* @param  interactionSpec                        Description of the Parameter
* @return                                        The inputRecordDefinition value
* @exception  javax.resource.Resource Define input port schema here ion of the Exception
*/
public ESBRecordDefinition getInputRecordDefinition(ESBInteractionSpec interactionSpec)
        throws javax.resource.ResourceException
{
     /**
     *@todo    Implement this
     */
    return null;

}
/**
*  Gets the outputRecordDefinition attribute of the Interaction object
*
*@param  interactionSpec Description of the Parameter
*@return   The outputRecordDefinition value
*@exception  javax.resource.ResourceException  Description of the Exception
*/
public ESBRecordDefinition getOutputRecordDefinition(ESBInteractionSpec interactionSpec)
        throws javax.resource.ResourceException
{
    /**
     *@todo    Implement this
     */
    return null;
}
```

Define output port schema here

Figure 3.6.61: SampleBCInteraction.java showing the business logic

Adding properties on response message

To add properties on to the response (output message) for a request make the following change in the <component_guid>Interaction.java

Add the following code in the <component_guid>Interaction class as shown in Figure 3.6.62.

```
/**
 * Processes the input Message. The input request is expected to be wrapped in
 * ESBRecord.
 * The default implementation extract the request payload (can be XML) and
 * delegates the call to {@link #execute(ESBInteractionSpec, String)}
 * The response payload (can be XML) returned is wrapped into {@link ESBRecord}
 * and is returned as the return parameter
 * <p/>
 * @param interactionSpec {@link ESBInteractionSpec} object passed from the client application
 * @param inputRecord {@link ESBRecord} containing the Request to be executed
 * @return response {@link ESBRecord}
 * @exception ResourceException if an error occurs while executing the EIS interaction.
 */
public ESBRecord execute(ESBInteractionSpec interactionSpec, ESBRecord inputRecord)
    throws ResourceException
{
    ESBInteractionSpec esbInteractionSpec = (ESBInteractionSpec) interactionSpec;
    String inputContent = null;
    try{
        //get the input record properties using        Retrieve properties from input message here
        //inputRecord.getProperty(<propertyName>)
        if (inputRecord != null)
            inputContent = inputRecord.getContentAsString();
    }
    catch (ESBException tfe) {
        m_logger.logrb(Level.SEVERE, AbstractESBInteraction.class.getName(), "execute",
                       Bundle.class.getName(), Bundle.ERROR_IN_PARSING_INPUT_RECORD, tfe);
        throw new ESBResourceException(Bundle.class, Bundle.ERROR_IN_PARSING_INPUT_RECORD, tfe).
    }
    String responseContent = execute(interactionSpec, inputContent);
    if (m_logger.isLoggable(Level.FINER))
        m_logger.logrb(Level.FINER, AbstractESBInteraction.class.getName(), "execute",
                Bundle.class.getName(), Bundle.RECEIVED_RESPONSE, responseContent);
    //create the response record
    ESBRecord esbResponseRecord = null;
    try{
        esbResponseRecord = (ESBRecord) inputRecord.clone();
    }
    catch (Throwable e){
        esbResponseRecord = new ESBRecord();
    }
    if (getOutputRecordDefinition(interactionSpec) == null)
        esbResponseRecord.setContentType(ESBRecord.CONTENT_TYPE_TEXT);
    esbResponseRecord.loadContent(responseContent);
    //set properties on outputRecord here using        Set properties here
    //esbResponseRecord.setProperty(<propertyName>,<propertyValue>)
    return esbResponseRecord;

}
```

Figure 3.6.62: code snippet to be added in SampleBCInteraction.java to read/set properties on the message

- Javadoc for API to add the properties on the ESBRecord is present at the following location %FIORANO_HOME%/javadoc/BCDK/index.html

Returning multiple responses for a single request

To return multiple responses for a single request

- Create a class which implements EnumerationRecord and implement hasMoreElements () and nextElement () method (shown in Figure 3.6.63)

`hasMoreElements ()` should return if more output messages exist

`nextElement ()` should contain logic for processing/retrieving next output message

```
public class SampleBCEnumerationRecord
      implements EnumerationRecord
{

    private ESBRecord currentRecord = null;

    /**
    * C'tor a File enumeration record.
    */
    public SampleBCEnumerationRecord() {

    }

    public boolean hasMoreElements(){
        //implement this
        return false;
    }

    public Object nextElement(){
        //implement this
        throw new NoSuchElementException();
    }

    /**
     * Sets the first response message.
     *
     * @param responseMessage the first response message
     */
    public void setFirstResponseMessage(ESBRecord responseMessage)
    {
        currentRecord = responseMessage;
    }

}
```

Figure 3.6.63: EnumerationRecord Implementation

- Add the code snippet shown (Figure 3.6.62) in **Adding properties on response message section** make the following change in the <component_guid>Interaction.java
- Add the following code in the <component_guid>Interaction class as shown in Figure 3.6.64.

```
/**
 * Description of the Method
 *
 * @param interactionSpec Description of the Parameter
 * @param inputRecord Description of the Parameter
 * @return Description of the Return Value
 * @throws ResourceException
 */
public Record execute(InteractionSpec interactionSpec, Record inputRecord)
        throws ResourceException {

    SampleBCEnumerationRecord outputRecord = new SampleBCEnumerationRecord();
    execute(interactionSpec, inputRecord, outputRecord);
    return outputRecord;

}

/**
 * Process the input with the specified configuration.
 *
 * @param interactionSpec the interaction specs
 * @param inputRecord the input record
 * @param outputRecord the output record
 * @return true if the record is processed
 * @throws ResourceException if there is an error then throw an resource
 *                           exception
 */
public boolean execute(InteractionSpec interactionSpec, Record inputRecord,
        Record outputRecord) throws ResourceException {

    if (_execute(interactionSpec, inputRecord, outputRecord))
        return true;

    if (inputRecord != null && inputRecord.size() > 0)
        ESBRecord inputESBRecord = (ESBRecord) ((IndexedRecord) inputRecord).get(0);

    ESBRecord outputESBRecord = execute( (ESBInteractionSpec)interactionSpec, inputESBRecord);
    ((SampleBCEnumerationRecord) outputRecord).setFirstResponseMessage(outputESBRecord);
    return true;

}
```

Figure 3.6.64 Code to be added in SampleBCInteraction.java for supporting multiple responses

3.6.3.2.4 Handling Connection

<component_guid>EISConnection.java creates and holds (in m_physicalConnection object) the component's connection to EIS. Figure 3.6.65 shows the code snippet where the connection creation has to be added. The user does not take to implement connection pooling. The framework automatically allows pooling of connections.

```
/**
* Creates EIS connection
* @exception javax.resource.ResourceException
*/
public void init()
    throws javax.resource.ResourceException
{
    /**
     * @todo add the code to create EIS Connection
     */
     m_physicalConnection = null;
}
/**
* Returns the EISProduct name for object
* @return EIS product name
*/
public String getEISProductName() throws javax.resource.ResourceException
{
    /**@todo Implement this*/
    return "SampleBCEIS";
}
/**
* Returns the UserName for object
* @return username string
 * @exception javax.resource.ResourceException
*/
public String getUserName() throws javax.resource.ResourceException
{
     /**@todo Implement this*/
    return "Anonymous";
}
/**
* Returns the EISProduct Version for object
* @return product version if EIS
* @exception javax.resource.ResourceException
*/
public String getEISProductVersion() throws javax.resource.ResourceException
{
    /**@todo Implement this*/
    return "1.0";
}
```

Create the connection here and assign to m_physicalConnection

m_physicalConnection should hold the connection object

provide details here which will be shown in the test result of MCF if connection is successfully established

Figure 3.6.65: SampleBCEISConnection.java showing location for connection creation

3.6.3.2.5 Configuration Property Sheet of Synchronous Components

The generated component by default has a Configuration Property Sheet (CPS), which can used to configure the adapter to perform required business logic. See section 3.6.3.2.7 Adding properties to Configuration Property Sheet to add additional properties in Managed Connection Factory and Interaction Configurations of the CPS.

A synchronous component's CPS has the following wizard steps:

Managed Connection Factory

Connection related configuration is captured in this panel. This panel is shown only if the component is defined to be connection based. To add properties in this panel add attributes in <component_guid>ManagedConnectionFactory.java

See javadoc of AbstractESBManagedConnectionFactory (super class of <component_guid>ManagedConnectionFactory.java) for all the methods and attributes that are available. Javadoc is located at the following location %FIORANO_HOME%/javadoc/BCDK/index.html

Interaction Configurations

Configuration details related to business logic are captured in this panel. To add properties in this panel add attributes in <component_guid>InteractionSpec.java

See javadoc of ESBInteractionSpec or ConnectionLessInteractionSpec (super class of <component_guid>InteractionSpec) for all the methods and attributes that are available. Javadoc is located at the following location %FIORANO_HOME%/javadoc/BCDK/index.html

Scheduler Configurations

Configuration details related to scheduling the execution of business components – like start time for scheduling, interval at which execution have to scheduled, number of times etc –are captured in this panel.

Transport Configurations (shown only if scheduling is enabled in Scheduler Configurations)

This panel contains transport related configurations such as type of JMS session (transacted or non-transacted), session count, acknowledge type for messages and so on.

Error Handling

Actions to be taken on event of errors during the component execution should be defined in this panel.

Adding Validation to MCF and Interaction Configurations panels

To enable validation in the MCF panels and Interaction Configurations panels override public void validate () throws ESBResourceValidationException method in AbstractESBManagedConnectionFactory and ESBInteractionSpec respectively and implement the validation.

Generating Sample Input

Fiorano synchronous component's framework generates sample input, which can be used to test the business logic in the Interaction Configurations panel of the CPS as explained in section 3.7.1.1 Testing in Configuration Property Sheet (CPS), using the schema returned by public ESBRecordDefinition getInputRecordDefinition (ESBInteractionSpec interactionSpec) throws ResourceException method in the <component_guid>Interaction class. However, it might be required to generate input using custom logic for the adapter, like in case of Text2XML. To generate sample input using custom logic override and implement the following method in <component_guid>InteractionSpec. public void generateSampleInput(java.io.OutputStream outStream) throws ESBException

Showing proxy related configuration details in the MCF panel

Sometimes component might have to make connection through a proxy. In such cases to take proxy related configuration details change the <component_guid>ManagedConnectionFactory.java as shown in Figure 3.6.66 and Figure 3.6.67.

```
public class SampleBCManagedConnectionFactory extends AbstractESBManagedConnectionFactory
```

Figure 3.6.66: Generated SampleBCManagedConnectionFactory.java

```
public class SampleBCManagedConnectionFactory extends AdvancedAbstractESBManagedConnectionFactory
```

Figure 3.6.67: Modified SampleBCManagedConnectionFactory.java to allow configuring proxy details

3.6.3.2.6 Exception handling

The logic for handling exceptions (as configured in Error Handling panel of CPS) is in-built in the synchronous component's framework. However, the user needs to throw appropriate exceptions with appropriate error codes for the framework to interpret exceptions correctly and handle them as configured.

Exceptions

There are two exception classes defined in the framework of synchronous component.

- ESBResourceException

  This exception class (subclass of javax.resource.ResourceException) should be used when throwing exceptions from classes in cci package (files in src\com\fiorano\adapter\jca\<component_guid>\cci folder) to classes in framework. Example in execute method of <component_guid>Interaction.java throw ESBResourceException with appropriate error codes.

- ESBResourceAdapterInternalException

  This exception class (subclass of javax.resource.ResourceException) should be used when throwing exceptions from classes in spi package (files in src\com\fiorano\adapter\jca\<component_guid>\spi folder) to classes in framework. Example in init method of <component_guid> EISConnection.java throw ESBResourceException with appropriate error codes.

Error codes

- INVALID_REQUEST_ERROR

  This is the error code to be used if the request (input message) is invalid (does not confirm to schema or is unexpected). This is used in cci package in general.

- RESPONSE_GENERATION_ERROR

  This is the error code to be used when an exception occurs while response (output message) is being created. This is used in cci package in general.

- REQUEST_EXECUTION_ERROR

  This is the error code to be used when an exception occurs while processing the request and the request cannot be executed. Typically for used exception in business logic. This is used in cci package in general.

- RESOURCE_CONNECTION_LOST_ERROR

  This is the error code to be used when the connection to EIS is lost while processing the request. This is used in both cci and spi packages in general.

- RESOURCE_CONNECT_ERROR

  This is the error code to be used when the attempt to connect to EIS fails. This is used in spi package in general.

3.6.3.2.7 Adding properties to Configuration Property Sheet (CPS)

The logic for creating and displaying CPS is inbuilt in the Fiorano framework. User can specify the properties that have to be displayed in the CPS (Figure 3.6.69) by specifying JMX descriptors (Figure 3.6.70) in the java files containing the configuration details. Such java files are java bean type classes which have getter and setter for all the fields and an empty constructor.

To add a new property that should be shown in the CPS

1. Add a getter and setter defined for the property.
2. Add jmx.managed-attribute descriptors above the method.
3. Add other jmx descriptors describing how the property has to be rendered or handled in the CPS.

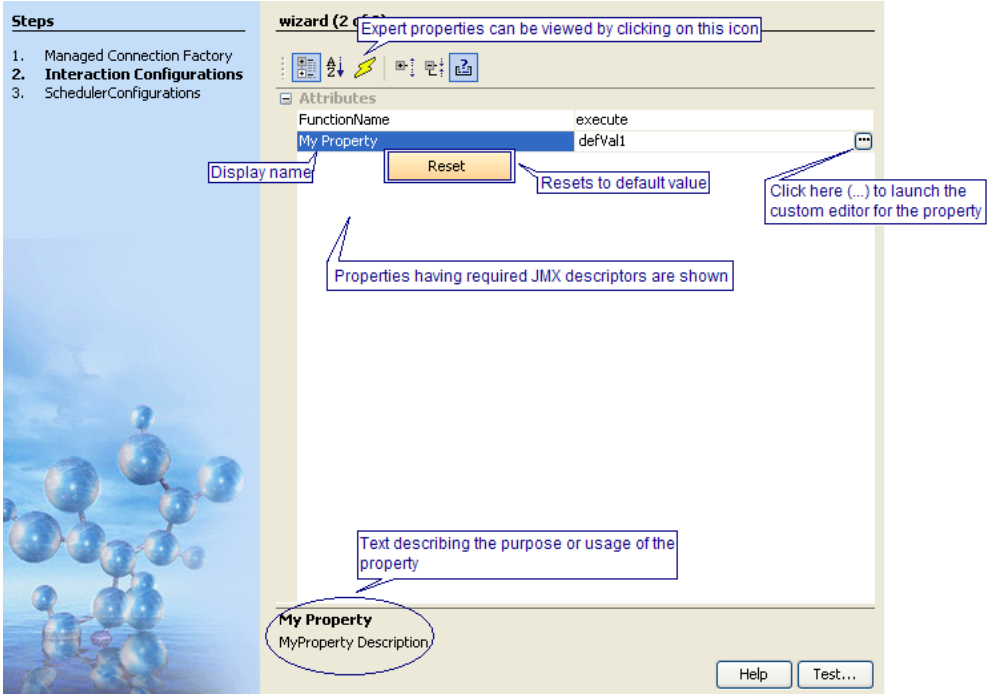


Figure 3.6.68: CPS containing a property, My Property, required for component configuration

```
 * @fiorano.xmbean
 * @jmx.mbean
 * @fesb.jca-interactionSpec-object description="SampleBCComponent"
 * name="SampleBC"
 * icon="bc.png"
  */
public class SampleBCInteractionSpec extends ESBInteractionSpec
{
    private String m_myProperty = "defVal";
    /**
     * @jmx:managed-c                                    'default constructor"
     */
    public SampleBCInteractionSpec()
    {
    }

    /**
    *@jmx.managed-attribute access="read-write" description="MyProperty Description"
    *@jmx.descriptor name="displayName" value="My Property"
    *@jmx.descriptor name="index" value="1"
    *@jmx.descriptor name="defaultValue" value="defVal"
    */
    public String getMyProperty()
    {
        return m_myProperty;
    }

   /**
    *@jmx.managed-attribute
    * Sets the property  value
    */
    public void setMyProperty(String property)
    {
        m_myProperty = property;
    }

}
```

Property required for configuration

Allows the property value to be changed

Text describing the purpose or usage of the property

Name of property that should be shown in the CPS

Default value to reset to

This attribute is jmx managed and shown in CPS

Figure 3.6.69: Java file showing the JMX descriptors for the property

All the JMX descriptors that can be added on the components getters are shown in the following table.

JMX descriptor	Description	Default	Example
propertyEditor	JMX descriptor to specify the editor class which should be launched when … is clicked in the CPS. By default basic editors are shown for basic return type example: if the getter returns a File then a file chooser is shown		@ jmx.descriptor name="propertyEditor" value="com.fiorano.adapter.jca.db.editors.SQLConfigurationEditor"
Expert	JMX descriptorTo make a specific Property Advanced so that its not visible unless 'expert' view is switched on in the BCDK	false	@jmx.descriptor name="expert" value="true"
displayName	By default, in the GUI the display name for a property is introspected from the class.		@jmx.descriptor name="displayName" value="My Display Name"

JMX descriptor	Description	Default	Example
	in case you want to specify a more meaningful display name to a property, then that can be done using the jmx.descriptor		
Optional	To notify the editor that this property is optional. Optional property names are rendered in new Color(0, 128, 0) in CPS.	false	@jmx.descriptor name="optional" value="true"
Password	To inform that this property is password and render as "*****"	false	@jmx.descriptor name="password" value="true"
Index	Define the order in which properties appear in Studio. Index should be a valid value is integer; Indexes start from 0(zero) If you don't specify this descriptor index is assumed to be 0 (zero) The order of attributes is indeterminate, if they have same index.	--	@jmx.descriptor name="index" value="1"
legalValues	If the property can take only a value from a list of some permissible values, then same should be provided as a JMX descriptor. The values must be provided as comma separated list.		@jmx.descriptor name="legalValues" value="Value1,Value2"
canEditAsText	if this property is not specified default value is false if legalValues are present else true if prop has legal values and canEditAsText is true, then you will see an editable combo box	false	@jmx.descriptor name="canEditAsText" value="true"
Refresh	If it is set to true for an attribute then all the attributes of the MBean are refreshed when the value changes	false	@jmx.descriptor name="refresh" value="true"

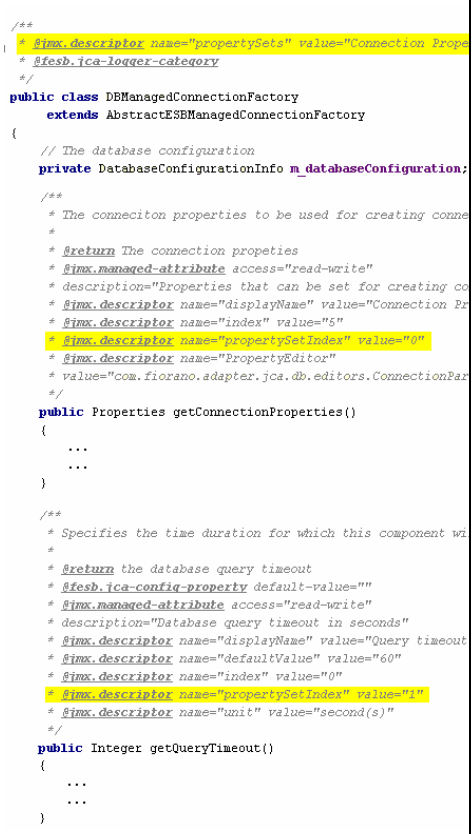
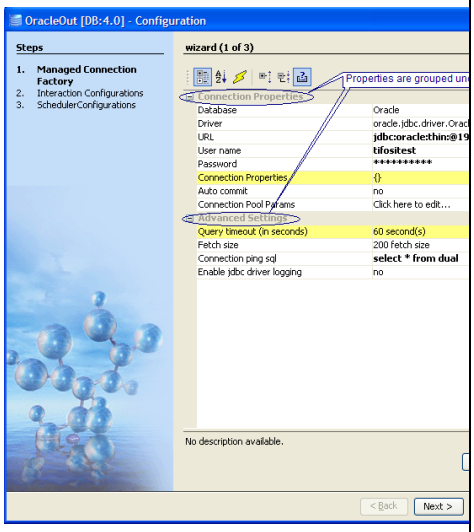
JMX descriptor	Description	Default	Example

hidesProperties	JMX descriptor to indicate that change in the value of this attribute hides some other properties.
	
	This descriptor also requires implementing method java.util.List fetchHiddenProperties()
	
	Example:
	
	class SampleBCConnection{
	
	getProxyUsed (boolean use);
	
	getProxyUser (String user);
	
	getProxyPassword (String pwd);
	
	}
	
	To show properties ProxyUser and ProxyPassword only when ProxyUsed is true; change that above code as follows:
	
	class SampleBCConnection{
	
	/**
	
	*@jmx.managed-attribute access="read-write" description="Determines whether proxy should be used to connect"
	
	*@jmx.descriptor name="displayName" value="Use proxy?"
	
	* @jmx.descriptor name="hidesProperties" value="true"
	
	**/
	
	getProxyUsed(boolean use);
	
	getProxyUser(String user);
	
	getProxyPassword(String pwd);
			@jmx.descriptor name="hidesProperties" value="true"

JMX descriptor	Description	Default	Example

	/**
	
	* @return
	
	* @jmx.managed-operation descriptor="get Hidden Properties"
	
	*/
	
	public java.util.List fetchHiddenProperties()
	
	{
	
	return isProxyUsed() ? Collections.EMPTY_LIST: Arrays.asList(new String[] { "ProxyUser", "ProxyPassword" });
	
	}
	
	}		

propertySets and propertySetIndex	These descriptors are used to categorize or group properties under different propertySets. Figure 3.6.70 shows the use propertySets and propertySetIndex descriptors to define the propertySets and group properties under propertySets respectively.		

JMX descriptor	Description	Default	Example
	 Figure 3.6.70: code snippet showing the use of propertySets and propertySetIndex  Figure 3.6.71: CPS containing		

```
/**
 * @jmx.descriptor name="propertySets" value="Connection Prope
 * @fesb.jca-logger-category
 */
public class DBManagedConnectionFactory
     extends AbstractESBManagedConnectionFactory
{
    // The database configuration
    private DatabaseConfigurationInfo m_databaseConfiguration;

    /**
     * The conneciton properties to be used for creating conne
     *
     * @return The connection propeties
     * @jmx.managed-attribute access="read-write"
     * description="Properties that can be set for creating co
     * @jmx.descriptor name="displayName" value="Connection Pr
     * @jmx.descriptor name="index" value="5"
     * @jmx.descriptor name="propertySetIndex" value="0"
     * @jmx.descriptor name="PropertyEditor"
     * value="com.fiorano.adapter.jca.db.editors.ConnectionPar
     */
    public Properties getConnectionProperties()
    {
        ...
        ...
    }

    /**
     * Specifies the time duration for which this component wi
     *
     * @return the database query timeout
     * @fesb.jca-config-property default-value=""
     * @jmx.managed-attribute access="read-write"
     * description="Database query timeout in seconds"
     * @jmx.descriptor name="displayName" value="Query timeout
     * @jmx.descriptor name="defaultValue" value="60"
     * @jmx.descriptor name="index" value="0"
     * @jmx.descriptor name="propertySetIndex" value="1"
     * @jmx.descriptor name="unit" value="second(s)"
     */
    public Integer getQueryTimeout()
    {
        ...
        ...
    }
```

JMX descriptor	Description	Default	Example
	properties grouped under propertySets The order of attributes within property set is still derived from "index" If "propertySets" is defined for mbeans; all attributes with "propertySetIndex" not set lies in propertySet 0 (zero) any property whose propertySetIndex is out-of-range lies in propertySet 0 (zero) the overlapping of propertySets among multiple mbeans is undefined;		
warningMessage	A warning Dialog pops up whenever the value of the property changes. The Message to be shown in the warning dialog should be set using value		@jmx.descriptor name="warningMessage" value="The Message comes here"
defaultValue	The default value which should be set for the property when “Reset” is clicked (Comes up on the right click on property, Figure 3.6.64)		@jmx.descriptor name="defaultValue" value="10"
minValue	Minimum value acceptable for the property		@jmx.descriptor name="minValue" value="0"
maxValue	Maximum value acceptable for the property		@jmx.descriptor name="maxValue" value="1000"
primitive	Used for Boolean properties, By Default a property specified Boolean shows “yes”, “no”, “none” in the combo box. If this property is set to true, “none” will not be shown	false	@jmx.descriptor name="primitive" value="true"

3.6.4 Non-Java components

Non-Java components can only be of the type Event Driven Business (EDBC) Components.

3.6.4.1 Defining a C component

To define a asynchronous component using C language, launch the component definition wizard as described in section 3.6.1.5 Defining Components. In the business Header panel, Service Type has to be chosen as "Non-Java" and Component type must be chosen as C as shown in figure 3.6.72.

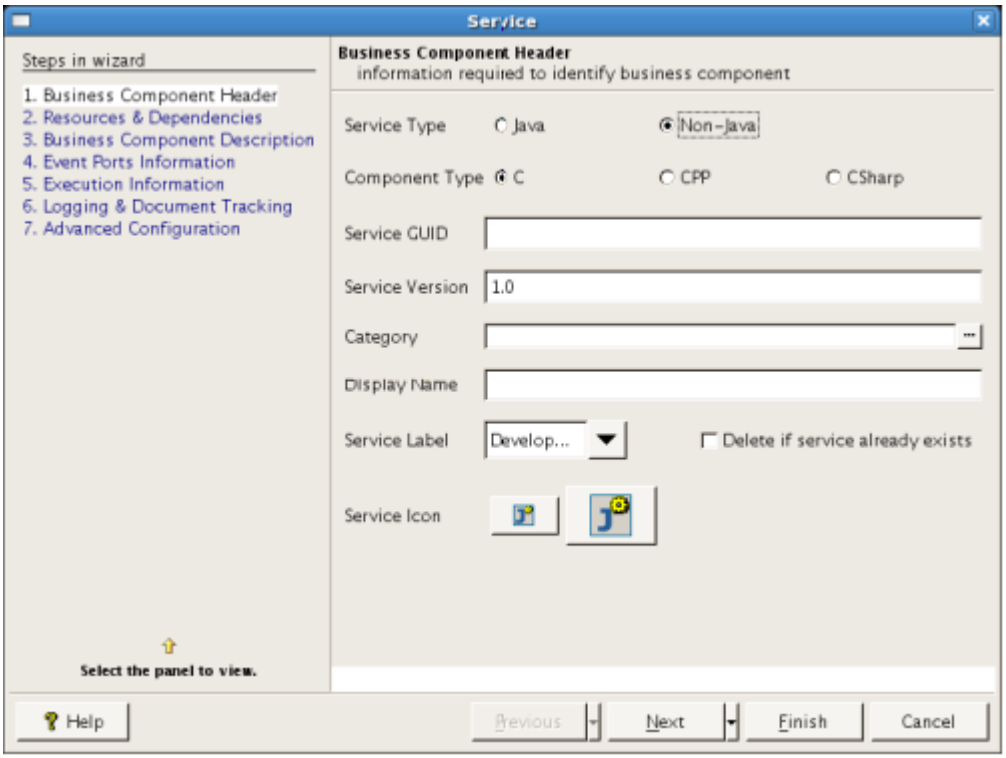


Figure 3.6.72 Specifying Service Type and Component Type

In the resources and dependencies panel no resources or dependencies needs to be specified for C/C++ components as they are built using static libraries and do not need any resources or libraries at runtime as shown in figure 3.6.73.

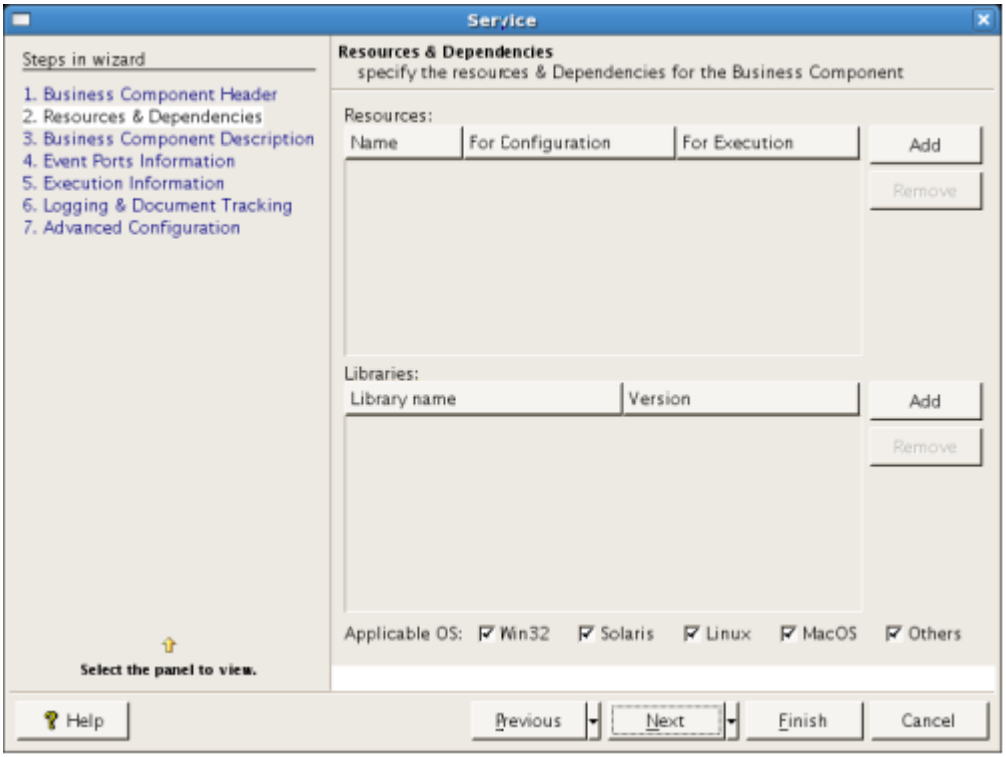


Figure 3.6.73 No resources or dependencies for C/C++ components

Follow the steps described in section 3.6.1.6 Getting familiar with wizard and service configuration to complete other panels. The wizard can be launched by typing in the following command from a terminal.

Example: wizard.bat –dest C:\SampleComponents\C_Sample\SampleC

3.6.4.1.1 Generation of code

To generate code for asynchronous component follow the steps in section 3.6.1.7 Generating code for the Defined Component.

Use -language option when generating sources to specify the language used for generating code for the component. A sample command is as shown below.

Example: generate.bat/sh –dest C:\SampleComponents\C\SampleC –language c

Figure 3.6.74 shows the structure of the code generated for an asynchronous component defined in C.

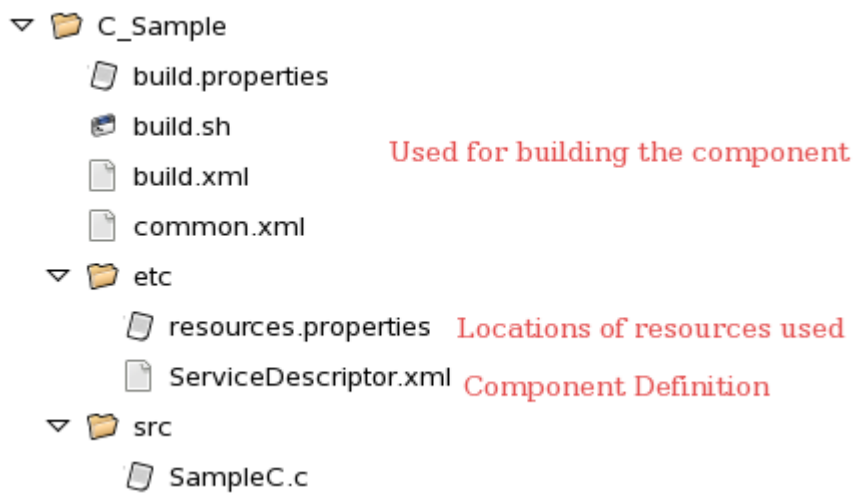


Figure 3.6.74: Structure of asynchronous component

3.6.4.1.2 Adding Business Logic

The purpose of the source files generated is explained below

The {ServiceGUID}.c contains the main() function which creates a connection with the peer server by calling the method **initializeConnection.**

When a message is received on the input port of the component, the function **onMessage** is called which in turn calls the function **processMessage** to process the message.

```
void onMessage(Message msg, void* param)
{
        char* temp = NULL;
        char *sendMessage = NULL;
        int i = 0;

        TextMessage tmsg = NULL;//(TextMessage)msg;
        temp = FTMSG_getText(msg);


        /* <Execute the component functionality > */
        sendMessage = processMessage(temp);
        /* <END> */
```

```
char* processMessage(char* msg)
{
        char* inMsg, *outMsg;
        outMsg = (char*)malloc(1000);
        inMsg = msg;
        /* < Write the message processing functionality (Forward input to Output)> */
        strcpy(outMsg, inMsg);
        /* < END > */
        return outMsg;
}
```

Figure 3.6.75: onMessage and processMessage methods

The functionality of the business component can be defined in the processMessage method.

3.6.4.1.3 Deploying the Component

To deploy the created component, make sure the Enterprise server is running and VC_HOME is set as environment variable or is defined in build.bat, such that %VC_HOME%/Bin points to the location of compiler (cl.exe). In Linux environment, setting this variable is **not** necessary since **gcc** will be used to compile the generated sources. The component can be deployed onto the server by giving the command **ant register** from the location where the component sources are generated**.** This will deploy the component on the enterprise server. Refer section 3.6.1.9 Deploying the Component for more information regarding the commands to register/unregister a component from the server.

For further instructions to generate/build from studio refer section 3.6.2 Component Creation in Fiorano Studio.

3.6.4.2 Creating a C++ component.

An asynchronous component using C++ language can be created similar to the steps described in section 3.6.4.1 Defining a C component. All the steps remain the same apart from setting the Component Type as **CPP** in Business Header Panel shown in figure 3.6.72. A sample command to launch the wizard is shown in the figure below.

Example: wizard.bat –dest C:\SampleComponents\cpp\SampleCpp

3.6.4.2.1 Generation of code for C++ component

To generate code for asynchronous component follow the steps in section 3.6.1.7 Generating code for the Defined Component.

Use `-language` option to specify the language used for generating code for the component.
Example: `generate.bat/sh –dest C:\SampleComponents\cpp\SampleCpp –language cpp`

Figure 3.6.76 shows the structure of the code generated for an asynchronous component defined in C++.

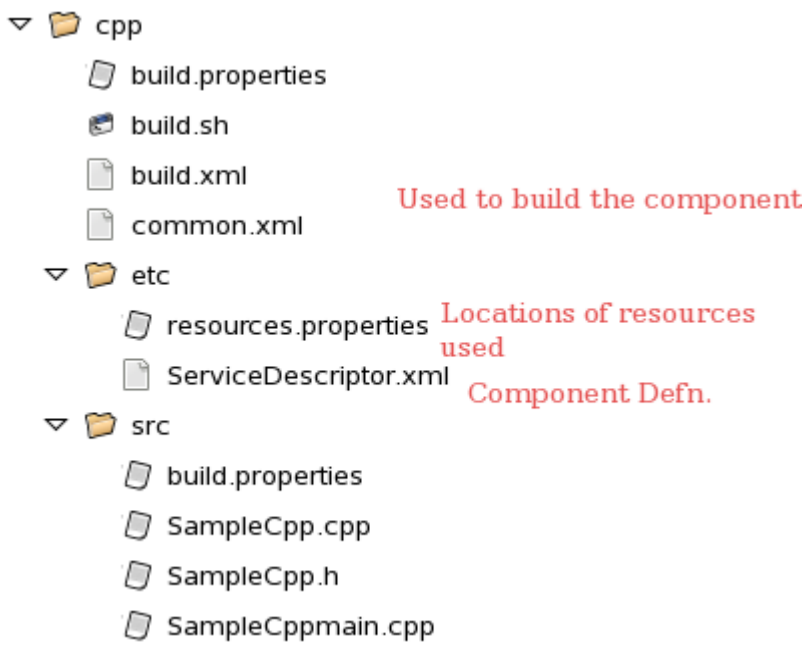


Figure 3.6.76: Structure of asynchronous component

3.6.4.2.2 Adding Business Logic

The purpose of the source files is explained below along with the control flow

The file {ServiceGUID}main.cpp contains the main function where all the required JMS objects are created and the service is started.

The generated main method is shown below:

```
try
{
                int i = 0;

                char* commandLine = NULL;
                commandLine = (char*)malloc(sizeof(char)*1000);
                memset(commandLine,0,1000);

                for(i = 0; i < argc - 1; i++)
                {
                    strcat(commandLine,args[i + 1]);
                    strcat(commandLine," ");

                }

                Service *service= new Service(commandLine);
                service->initializeValues();
                service->createJMSObjects();
                //service->Run();
                getchar();
}
```

When a message is received on the input port of the component, the function **onMessage** defined in {SerivceGUID}.cpp will be called and the business logic of the component can be defined in this method. (See Figure 3.6.75)

```
void CMListener::onMessage(CMessage *msg)
{
                try
        {
                        CTextMessage *forwardMsg;
                        CTextMessage *cmsg = (CTextMessage *)msg;

                        String message = (char*)cmsg->getText();
                        // Add the business process functionality here

                        if(m_ts != NULL)
                                forwardMsg = m_ts->createTextMessage();
                        else if(m_qs != NULL)
                                forwardMsg = m_qs->createTextMessage();
```

Figure 3.6.75: Default contents of the onMessage method of {ServiceGUID}.cpp

3.6.4.2.3 Deploying the Component

To deploy the created component, make sure the Enterprise server is running and VC_HOME is set as environment variable or is defined in build.bat, such that %VC_HOME%/Bin points to the location of compiler (cl.exe). In Linux environment, setting this variable is **not** necessary since **g++** will be used to compile the generated sources. The component can be deployed onto the server by giving the command **ant register** from the location where the component sources are generated**.** This will deploy the component on the enterprise server. Refer section 3.6.1.9 Deploying the Component for more information regarding the commands to register/unregister a component from the server.

For further instructions to generate/build from studio, refer section 3.6.2 Component Creation in Fiorano Studio.

3.6.4.3 Creating a C# Component

Important: To create a CSharp EDBC component, ensure that .NET 2003 or above is installed on the machine that hosts the peer server on which the component will finally be deployed.

An asynchronous component using C# language can be created similar to the steps described in section 3.6.4.1 Defining a C Component. Component Type should be chosen as CSharp in Business Header Panel shown in figure 3.6.72.

In the Resources and Dependencies panel, by default a resource **fmq-csharp-native.dll** is added as shown in figure 3.6.76. Any **additional dlls** necessary for the component can be added in this panel.

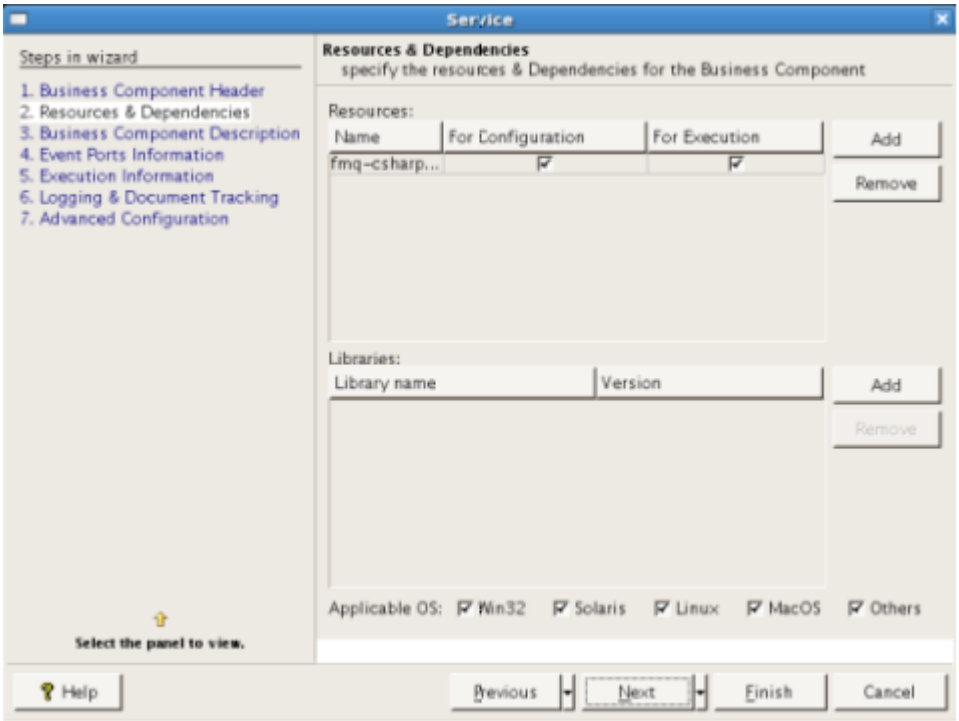


Figure 3.6.76 Adding dlls to cSharp component.

Example: wizard.bat –dest C:\SampleComponents\CSharp\CSSample

3.6.4.3.1 Code Generation

To generate code for asynchronous component follow the steps in section 3.6.1.7 Generating code for the Defined Component.

Use -language option to specify the language as csharp for generating code for the component.

Example: generate.bat/sh –dest C:\SampleComponents\CSharp\CSSample –language csharp

3.6.4.3.2 Adding Business Logic

By default, the component sends the messages received on its input port to the output port. Custom component generation wizard creates a visual studio solution file (.sln) (see Figure 3.6.77) which can be opened using Microsoft Visual Studio 2003 or above.

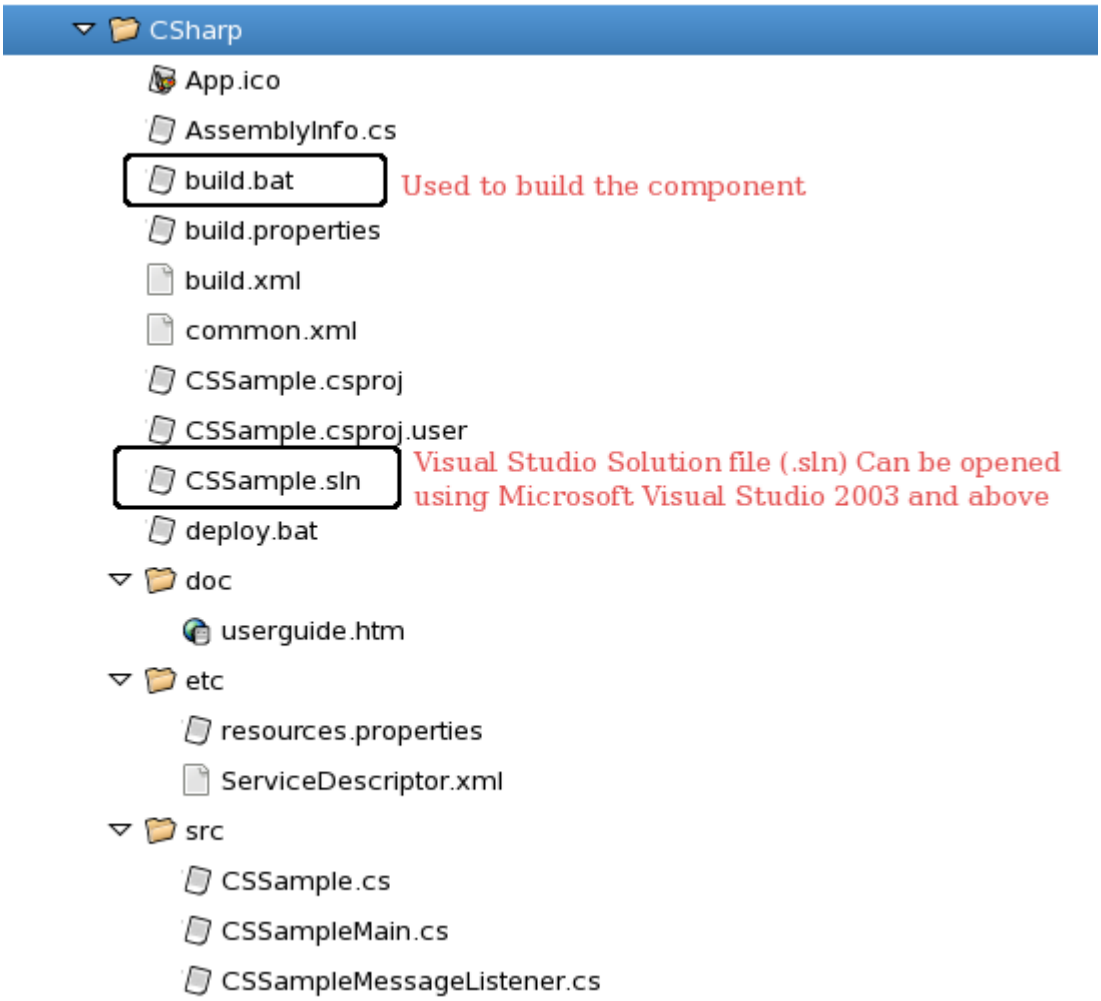


Figure 3.6.77: Structure of asynchronous component

The main method in {**ServiceGUID}Main** creates all the required JMS objects and starts the component. The Message Listener (**{ServiceGUID}MessageListener**) contains the callback method **onMessage.** For each message received on the input port of the component, **onMessage** function is called and this calls **processMessage** function. The business logic can be provided in the method **processMessage.**

```
public void onMessage(Message msg)
{
        try
        {
                TextMessage textMsg = (TextMessage)msg;
                TextMessage forwardMsg = null;
                string message = textMsg.getText();
                // Add the business process functionality here
                string procMsg = processMessage(message);
                // End

forwardMsg = sess.createTextMessage();
                forwardMsg.setText(procMsg);
                prod.send(forwardMsg);
```

3.6.4.3.3 Deploying the component

To deploy the created component, make sure the Enterprise server is running and DOT_NET_FRAMEWORK_HOME is set as environment variable or is defined in build.bat, such that it points to the location of csc.exe.

Run **build.bat,** this will create executable (CSSample.exe in this case) in the "build" folder of the custom component directory (C:\SampleComponents\CSharp\CSSample in our example).

```
C:\EDBC\CsEDBC>build.bat
Using DOT_NET_FRAMEWORK_HOME C:\WINDOWS\Microsoft.NET\Framework\v3.5
Step 1: Compiling, Linking and generating exe module...
Completed !

C:\EDBC\CsEDBC>_
```

Run **deploy.bat.** This opens up a new command window which connects to the Fiorano Enterprise server and deploys the component.

Alternatively **ant register** can also be used to register the component. Refer section 3.6.1.9 Deploying the Component for more information regarding the commands to register/unregister a component from the server.

For further instructions to generate/build from studio, refer section 3.6.2 Component Creation in Fiorano Studio.

3.7 Service Component Testing

This sections talks about how the functionality of the component can be tested. A very simple way to test the component is to use it the way it is intended to be used! That is to create a flow with the component and run the flow to check if desired results are obtained. Unfortunately, at times the result may not be the desired one for no fault in the component logic but for external reasons. So Fiorano provides a more isolated way to test the synchronous components' logic.

3.7.1 Testing Synchronous Components

Synchronous can be tested

- During the configuration time using the 'test' button in the CPS
- Using JUnit test cases

3.7.1.1 Testing in Configuration Property Sheet (CPS)

For all the synchronous components the Configuration Property Sheet (CPS) contains a 'test' button on the Managed Connection Factory (MCF) panel (in case of a connection based component, Figure 3.7.2) and the interaction step panel (Figure 3.7.4).

3.7.1.1.1 Testing the Connection

To test the connection which the component makes with the underlying EIS, specify the connection details in the Managed Connection Factory panel and click 'test'.

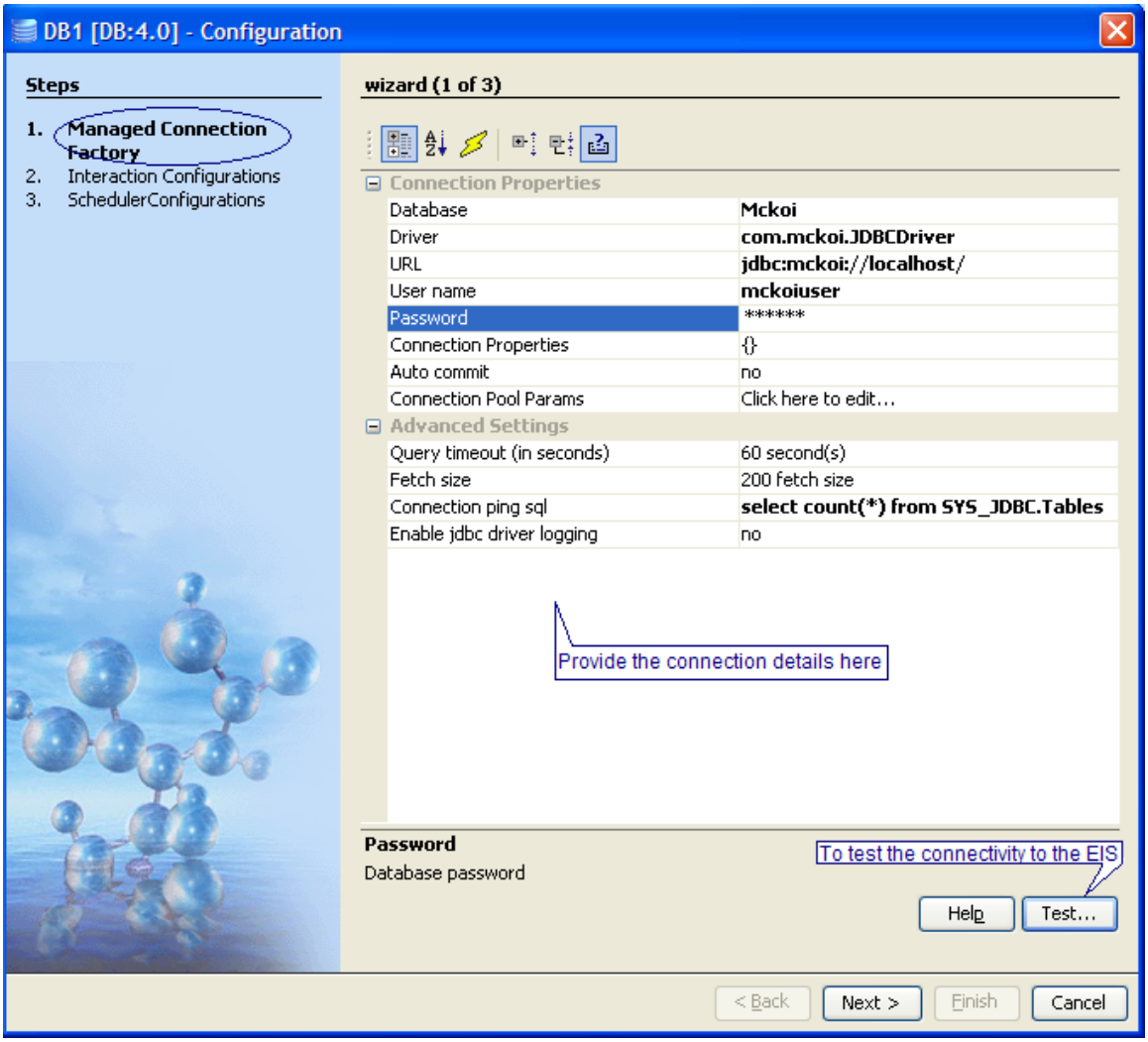


Figure 3.7.1: Managed Connection Factory (MCF) panel showing the connection details and Test button

A dialog pops up showing the result of the operation (connecting to EIS) performed. The result can be one of the following

A text/XML result showing the connection details of the EIS to which the component successfully connects to (Figure 3.7.2)

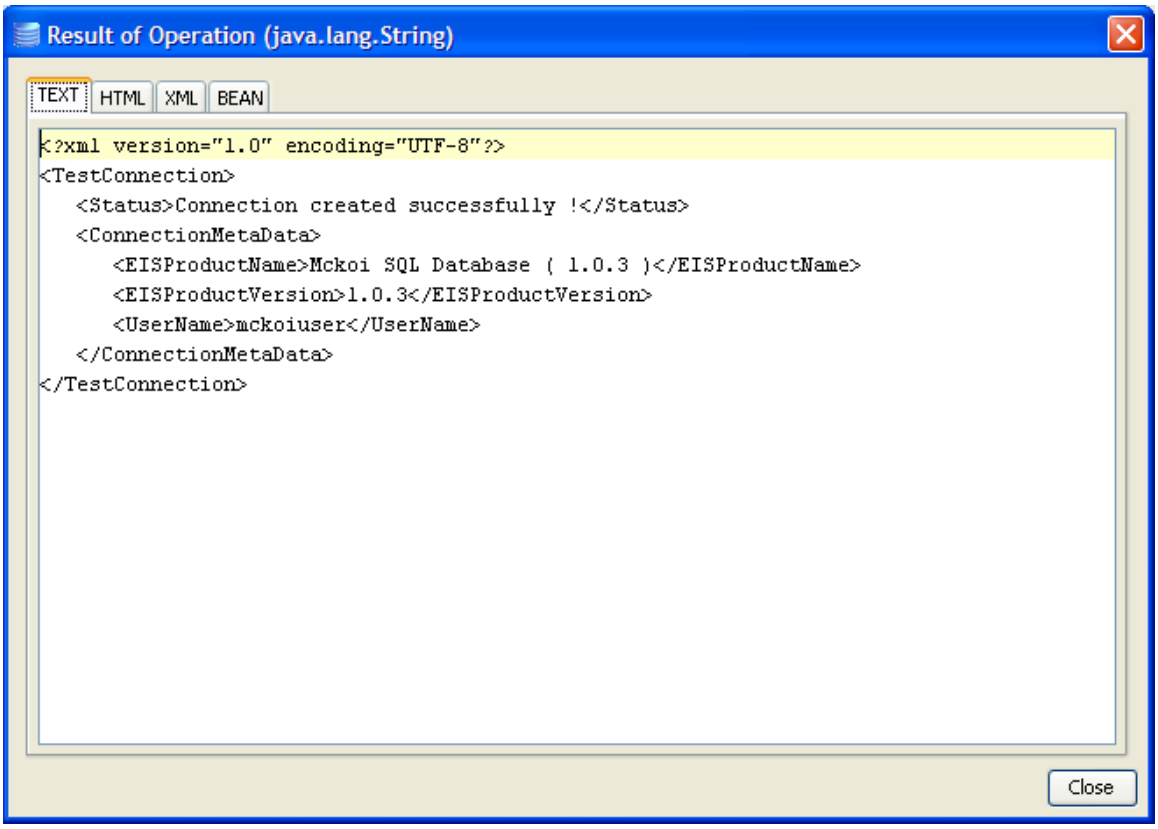


Figure 3.7.2: The result of test on MCF. The connection to the database is successfully established.

A trace of error/exception occurred while the component is trying to connect to MCF (Figure 3.7.3)

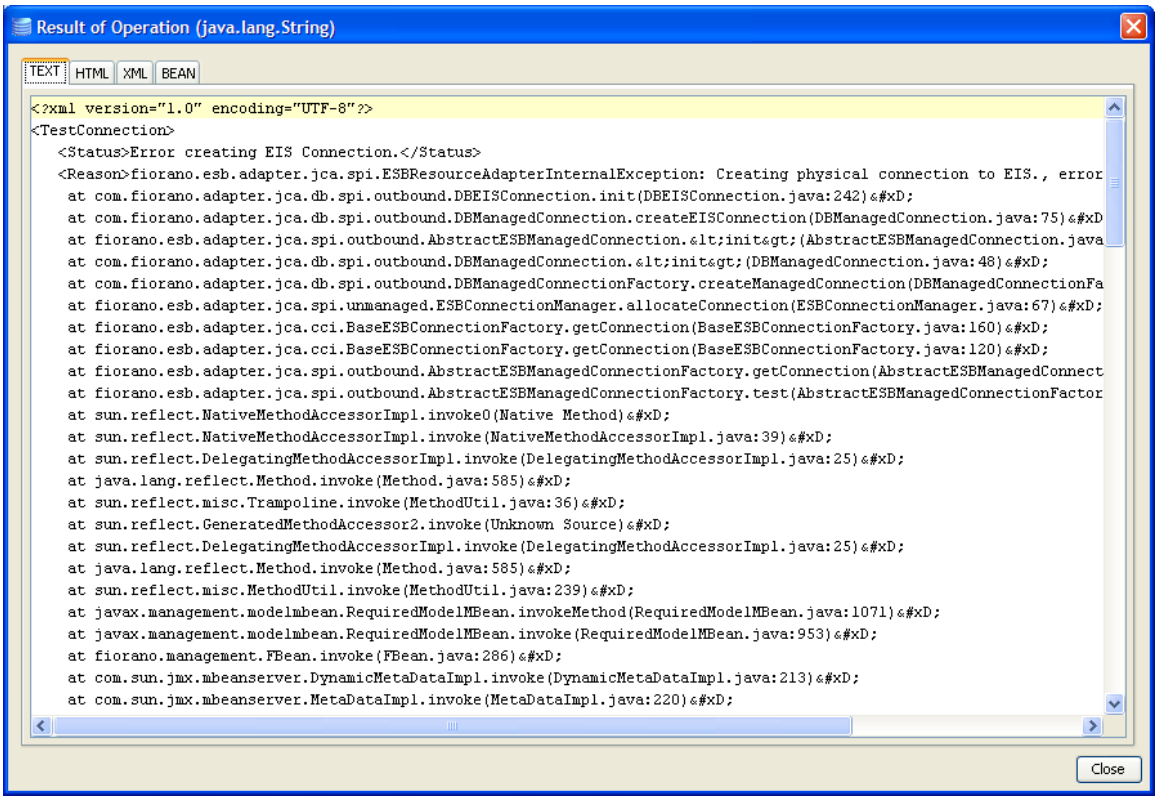


Figure 3.7.3: Exception when the test in MCF panel fails

3.7.1.1.2 Testing the business logic

The second step in the Configuration Property Sheet, called Interaction Configurations step, contains the details defining the business logic the component executes. The business logic of the component can be tested from the test button in the interaction configurations panel.

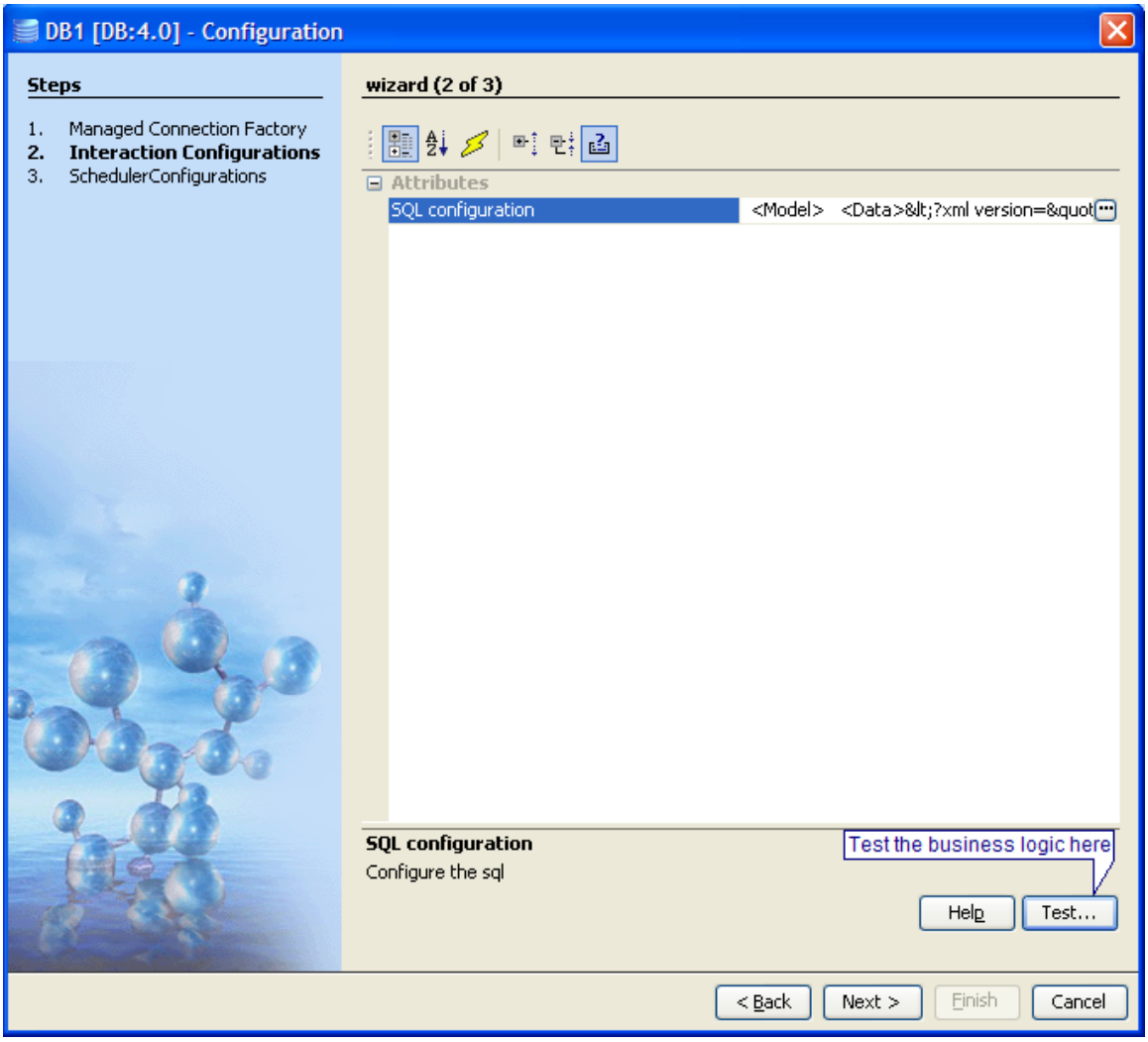


Figure 3.7.4: Interaction Configurations panel showing the test button

The test dialog allows the user to invoke the component with sample input, which would be pushed onto the input port of the component during runtime. The component processes the input message and displays the output message, which comes out onto the output port of the component during the runtime. Figure 3.7.5 explains the test dialog.

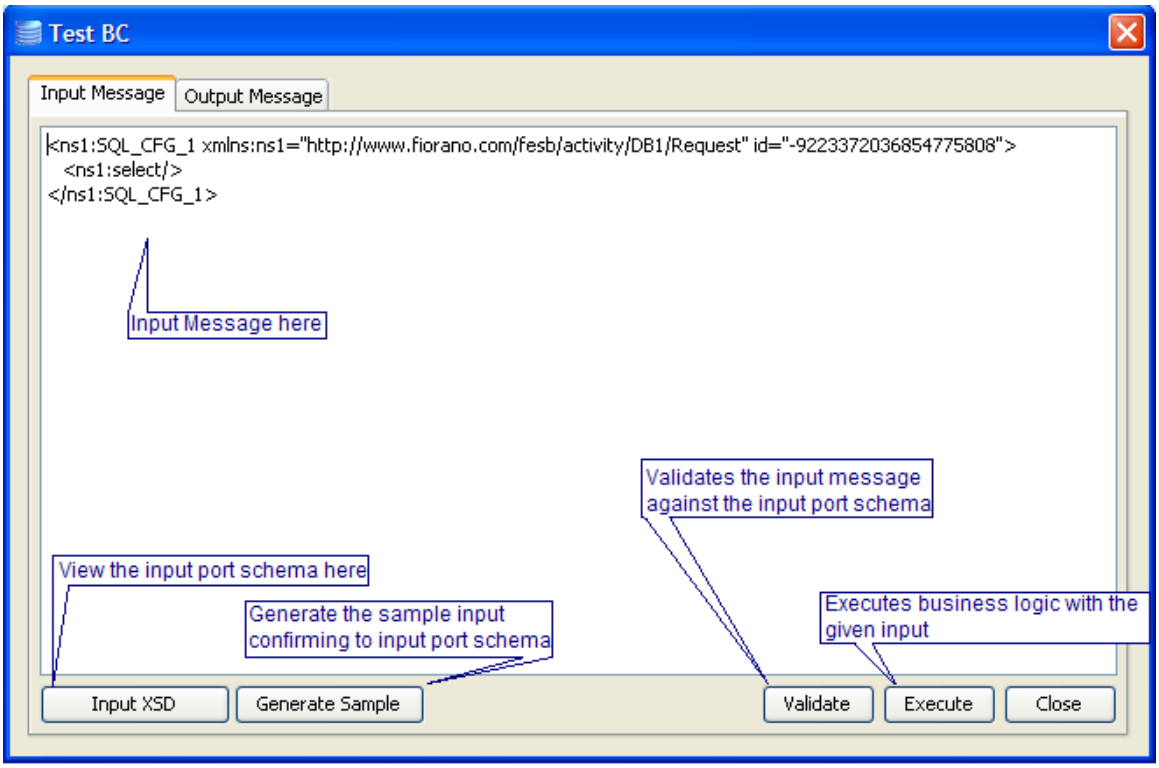


Figure 3.7.5: Test dialog to test the component's business logic, showing the input message details

The Input Message tab allows the user to specify the input message and shows the details of the input message. To view the input port schema click 'Input XSD' button. The dialog (Figures 3.7.6 to 3.7.8) that comes up shows the input port schema details.

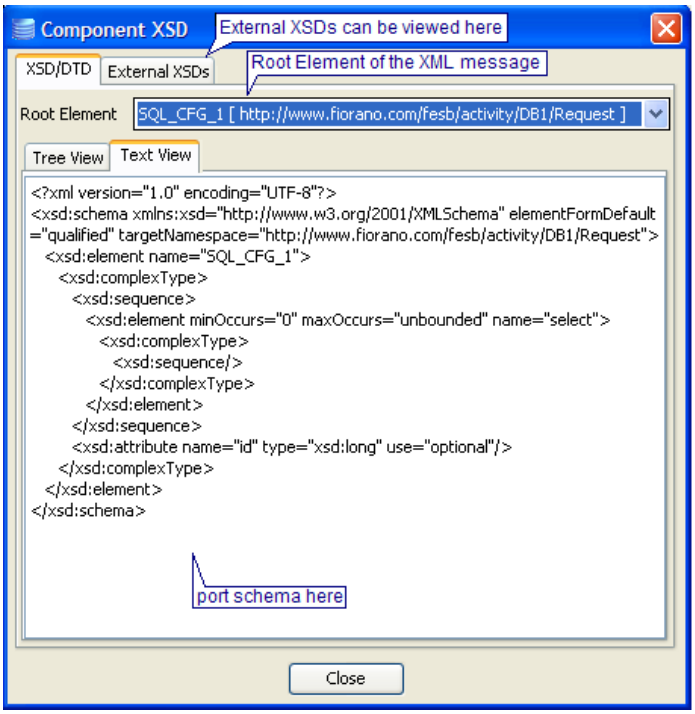


Figure 3.7.6: Port schema (text view)

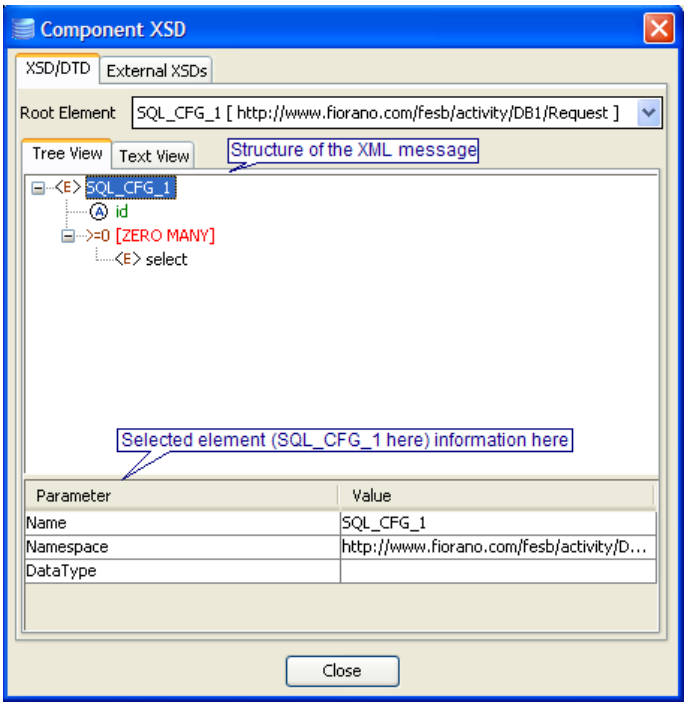


Figure 3.7.7: Port schema (tree view)

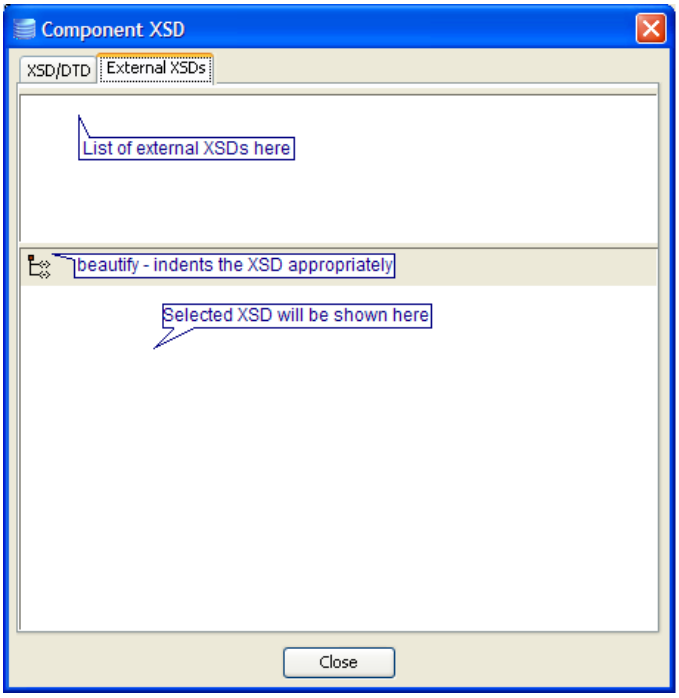


Figure 3.7.8: External XSDs for the port schema

To test the component, click execute in Figure 3.7.5. The output is shown in the output message tab (Figure 3.7.9). To view the output port schema click Output XSD. A dialog similar to the dialog of input port schema (Figures 3.7.6 to 3.7.8) comes up showing the output port schema details. To validate the output message against the output port schema click validate.

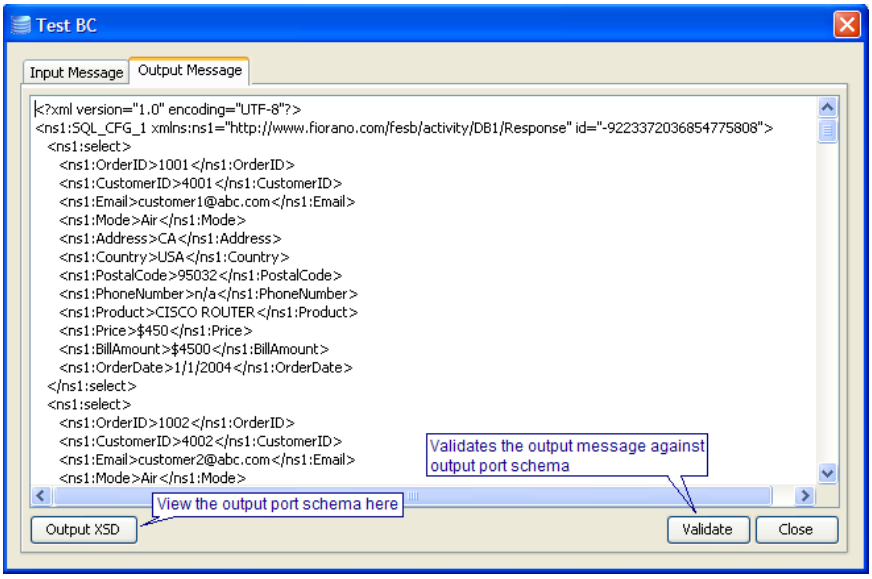


Figure 3.7.9: Result of the test operation

3.7.1.2 Testing using JUnit test cases

Fiorano provides a JUnit test framework to test the logic of the component created. A JUnit test case can perform both the test operations (from Managed Connection Factory and Interaction Configurations panels).

3.7.1.2.1 Configuring a JUnit test case

To write a JUnit test cases create the test hierarchy as shown in Figure 3.7.10.

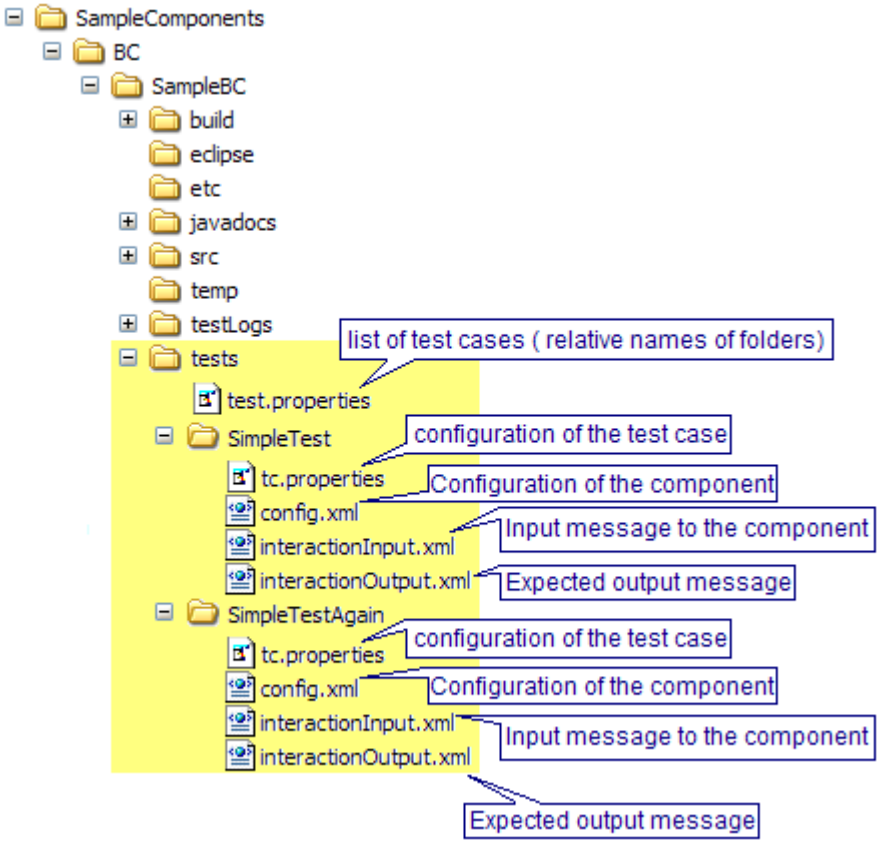


Figure 3.7.10: Structure for JUnit test cases for the component

The test.properties file is a properties file that contains the list of paths to test cases, relative to the tests directory. The entry in test.properties for structure showed in Figure 3.7.11 looks as follows:

```
test.ids=/SimpleTest,/SimpleTestAgain
```

Figure 3.7.11: Entry in test.properties listing all the test cases

Each test should contain a file tc.properties that contains the configuration of the test case. The configuration of a test case includes configuration of the component, input message text that has to be passed to the component and expected output the component should return among other properties. The list of properties that can be specified is shown in the following table.

Property	**Description**	**Default**
bcConfig	Configuration XML of the component saved using configureBC.bat utility	bcConfig.xml
configuredUsingStudio	Indicates whether the configuration is given as mcf and jca files or as configuration xml • true – configuration is specified as mcf and jca files • false – configuration is specified as a configuration xml from configureBC.bat utility	true
compareInteractionOutput	Specifies whether output of the component should be compared with jca.output or not • true – compare • false – do not compare	true
connectionLess	Indication to test framework whether a test for connection has to be performed or not. • true – component is connection based and connection test has to be performed • false – component does not create connection or the test for connection need not be performed **Note**: if this is false then mcf.output property should be specified	false
inputContentType	The type of input message text. Values from • TEXT – plain text • XML – XML message • BYTES – Binary content	XML
inputPropsFile	Location pointing to a file containing message properties that should passed to the component. Example: COMPLETED=true in case of FileWriter	inputProps.properties
Jca	jca file containing the interaction configurations	bc.jca
jca.exception	File containing the expected exception for a negative test case	interactionException.xml

Property	Description	Default
jca.input	File containing the input message content that should be used to test the component	interactionInput.xml
jca.output	File containing the expected output from the component.	interactionOutput.xml
Mcf	mcf file containing the connection configurations	bc.mcf
mcf.output	File containing expected result of the connection test	mcfOutput.xml
mcf.exception	File containing the expected exception message for a negative test case	mcfException.xml
negativeTest	• true – if the test case is configured for a negative test (compares output with jca.exception and mcf.exception • false – if the test case is configured for a positive test (compares output with mcf.output and jca.output)	false
xmlDiffClass	Custom comparison class for comparing outputs of interaction. **Usage**: Implement a class extending org.custommonkey.xmlunit.Diff with a constructor to take two String arguments actualOutput and expectedOutput. Add it to the class path for tests by specifying it in tests.lib path like structure which can be over-ridden in components.	none

A typical tc.properties file is shown in Figure 3.7.12.

```
configuredUsingStudio=false
jca.input=interactionInput.xml
jca.output=interactionOutput.xml
mcf.output=mcfOutput.xml
bcConfig=config.xml
connectionLess=false
inputContentType=TEXT
```

Figure 3.7.12 Sample tc.properties

Note: For components and component configuration which might return multiple outputs for a single input. The expected output files should be suffixed with the sequence number before the extension.

Example: If the entry in the tc.properties file is output.xml then the actual expected outputs should be specified in files output1.xml, output2.xml and output3.xml for configuration returning 3 output messages

3.7.1.2.2 Executing a JUnit test case

To run / execute the JUnit test case, execute ant runtests from the component's folder. A folder named testLogs is automatically created in the component folder. Logs of the individual test cases is present in tests folder which is under testLogs. The directory structure of testLogs folder is shown in the Figure 3.7.13.

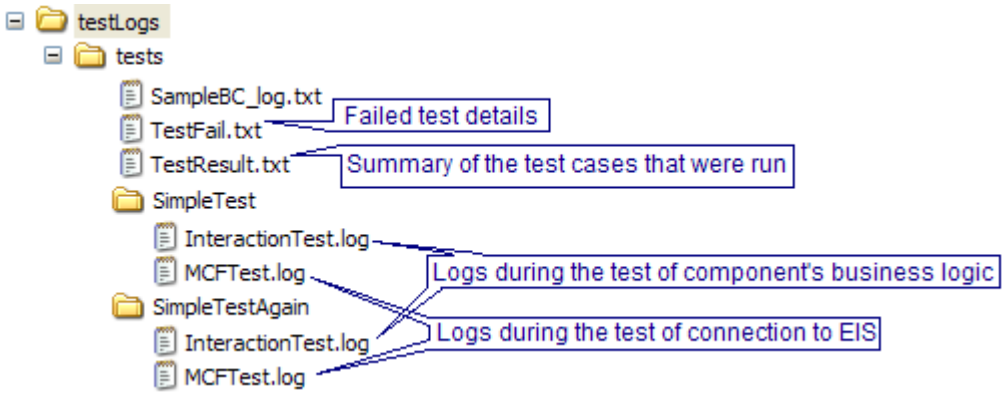


Figure 3.7.13: Directory structure of the testLogs containing the results of tests

Sometimes it is required to add classes specific to the component to classpath of the tests execution, such classes can be added in the tests.lib path in the build.xml of the component as shown in Figure 3.7.14.

To perform some tasks before the tests start add a target testspreprocess to the component's build.xml (shown in Figure 3.7.14)

Similarly to perform tasks after the tests end add a target runtests to the component's build.xml (shown in Figure 3.7.14)

Note: both testspreprocess and runtests target should depend on the respective targets in the common.xml.

A sample build file containing necessary entries to start the FMQ server before the test cases begin and to shutdown the FMQ server after the test cases end is shown in Figure 3.7.14

```
<target name="testspreprocess" depends="fbc-comp-common.testspreprocess">
                <exec dir="${deploy.dir}/fmq/bin" executable="runContainer.bat" resolveexecutable="true" spawn="true"/>
</target>

<target name="runtests" depends="fbc-comp-common.runtests">
        <exec dir="${deploy.dir}/fmq/bin" executable="shutdown.bat" resolveexecutable="true"/>
</target>

<path id="tests.lib">
        <pathelement location="${deploy.dir}/fmq/lib/client/all/fmq-client.jar"/>
</path>
```

Figure 3.7.14: sample build.xml containing necessary entries for tests.lib, testspreprocess and runtests

3.7.2 Testing Asynchronous Components

Unlike synchronous components the asynchronous components cannot be tested during the configuration time, unless the required functionality is specifically added. However, Fiorano provides a testing framework for testing asynchronous components as well.

JUnit testing of asynchronous components is similar to JUnit testing of synchronous component

3.7.2.1 Configuring a JUnit test case

To write a JUnit test cases create the test hierarchy as shown in Figure 3.7.15

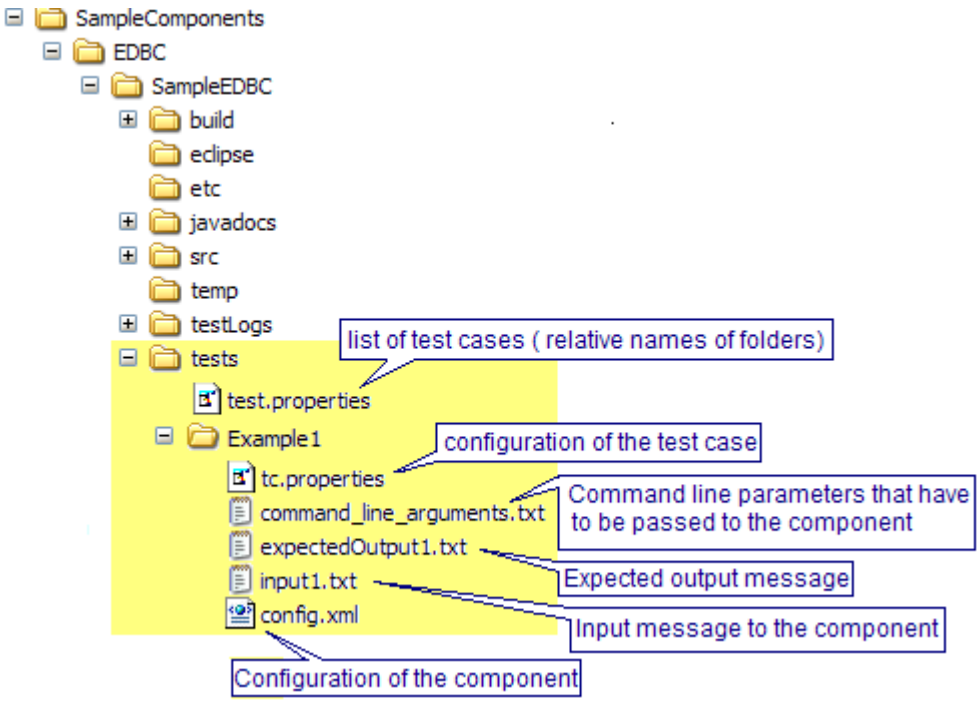


Figure 3.7.15: Structure for JUnit test cases for the component

The test.properties file is a properties file that contains the list of paths to test cases, relative to the tests directory. The entry in test.properties for structure shown in Figure 3.7.16 looks as follows:

```
test.ids=/SimpleTest,/SimpleTestAgain
```

Figure 3.7.16: entry in test.properties listing all the test cases

Each test should contain following files

config.xml

This file contains the configuration of the component. The configuration can be obtained as follows

- Create an event process
- Drag and drop the asynchronous component to be tested
- Configure the component
- Save the event process

Extract the configuration from application's xml file and copy it in config.xml and save the config.xml file

To extract the configuration from the application's file

- Open the application's xml file located at %FIORANO_HOME%\esb\fes\repository\applications\<APPLICATION_NAME>
- In the xml file under the /Application/ServiceInstances node look for ServiceInstance node of the required component
- Under the ServiceInstance node from ./RuntimeArguments/ UserDefinedPropertySheet node copy CDATA section and save it into config.xml file
    - command_line_arguments.txt

This file consists of all the command line arguments that have to be passed to the component during the launch of the component. A typical command_line_arguments.txt is shown in Figure 3.7.17.

```
java
-classpath
C:\PROGRA~1\Fiorano\FioranoSOA9.3.0\esb\tools\templates\testEDBC\build\dist
\testEDBC.jar:C:\PROGRA~1\Fiorano\FioranoSOA9.3.0\esb\server\repository\components
\EDBCEngine\4.0\EDBCEngine.jar
com.fiorano.edbc.testedbc.TestEDBC
numConnections
1
-username
anonymous
numSessions
1
-serviceversion
1.0
numConsumers
1
-password
anonymous
-transportprotocol
TCP
-url
http://localhost:1867
-serviceGUID
testEDBC
-instancename
testEDBC1
-eventprocessname
EVENT_PROCESS1
-componentrepopath
C:\PROGRA~1\Fiorano\FioranoSOA9.3.0\esb\server\repository\components
-eventstopic
FES_USER_EVENTS_TOPIC
-connfactory
EVENT_PROCESS1__TESTEDBC1
-backupurl
http://localhost:1956
```

Figure 3.7.17: command_line_arguments.txt for the asynchronous component

tc.properties that contains the configuration of the test case. The configuration of a test case includes input ports and output ports that have to be created and location of the files containing input messages to be passed and output messages to be compared. The list of properties that can be specified is shown in the following table.

Property	Description
inputPortNames	Comma separated values of the input port names the component has for this configuration
outputPortNames	Comma separated values of the output port names the component has for this configuration
otherTopicNames	Any other JMS destinations that have to be created for the component's execution
inputFiles	A list of files in which each contains a single message to be sent on the input port. Specify comma separated list of values to be sent on to each of input port specified in the inputPortNames list respectively. Example If component has to input ports and message in input1.txt is to be sent on to the IN_PORT1 and message in input2.txt is to be sent on to the IN_PORT2. Then entries should look like this. inputPortNames=IN_PORT1, IN_PORT2 inputFiles=input1.txt,input2.txt If multiple inputs have to be sent to the input port then specify colon separated list in each of the comma separated value. Example: In the example given above if messages from input11.txt and input 12.txt have to be sent on to IN_PORT1 and messages from output21.txt, output22.txt and output23.txt have to be sent on to IN_PORT2, then specify as follows inputPortNames=IN_PORT1, IN_PORT2 inputFiles=input11.txt:input12.txt,input21.txt: input22.txt: input23.txt
outputFiles	A list of files which have to be populated with the output messages when the test is executed. This should be a comma separated list each mapping to a corresponding output port in outputPortsNames list. These file can be empty or non-existent

Property	Description
expectedOutputFiles	A list of files in which each contains an expected output message that should be received on the output ports of the component. This should be a comma separated list each mapping to a corresponding output port in outputPortsNames list.

A typical tc.properties file is shown in Figure 3.7.18

```
inputPortNames=IN_PORT
outputPortNames=OUT_PORT
otherTopicNames=FPS_USER_EVENTS_TOPIC
inputFiles=input1.txt
outputFiles=output1.txt
expectedOutputFiles=expectedOutput1.txt
```

Figure 3.7.18: Sample tc.properties

3.7.2.2 Executing a JUnit test case

Before executing the test case, an EventProcess with the newly created custom component should be launched and the same details have to be given in *command_line_arguments.txt* file as explained above. To run / execute the JUnit test case, execute ant runtests from the component's folder. A folder named testLogs is automatically created in the component folder. The directory structure of testLogs folder is shown in Figure 3.7.19.

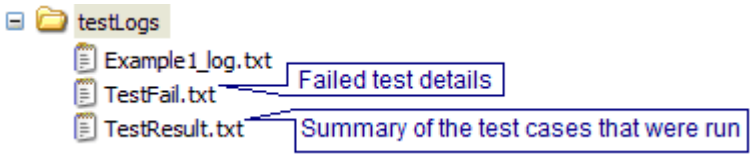


Figure 3.7.19: Directory structure of the testLogs containing the results of tests

Sometimes it is required to add classes specific to the component to classpath of the tests execution, such classes can be added in the tests.lib path in the build.xml of the component as shown in Figure 3.7.20.

To perform some tasks before the tests start add a target testspreprocess to the component's build.xml (shown in Figure 3.7.20)

Similarly to perform tasks after the tests end add a target runtests to the component's build.xml (shown in Figure 3.7.20)

Note: both testspreprocess and runtests target should depend on the respective targets in the common.xml.

A sample build file containing necessary entries to start the FMQ server before the test cases begin and to shutdown the FMQ server after the test cases end is shown in Figure 3.7.20.

```
<target name="testspreprocess" depends="fbc-comp-common.testspreprocess">
                <exec dir="${deploy.dir}/fmq/bin" executable="runContainer.bat" resolveexecutable="true" spawn="true"/>
</target>

<target name="runtests" depends="fbc-comp-common.runtests">
        <exec dir="${deploy.dir}/fmq/bin" executable="shutdown.bat" resolveexecutable="true"/>
</target>

<path id="tests.lib">
        <pathelement location="${deploy.dir}/fmq/lib/client/all/fmq-client.jar"/>
</path>
```

Figure 3.7.20: sample build.xml containing necessary entries for tests.lib, testspreprocess and runtests

3.8 Component Generation - SimpleJMS, MultiThreaded and POJO

3.8.1 EDBC Templates

Fiorano SOA provides EDBC templates which aid in component development. Three types of templates are supported for EDBC components.

- Simple JMS
- Multi Threaded
- POJO (Plain Old Java Object)

3.8.1.1 Scripts

Windows: %FIORANO_HOME%\esb\tools\templates\edbctemplates.bat

Linux: %FIORANO_HOME%/esb/tools/templates/edbctemplates.sh

3.8.1.2 Simple JMS

3.8.1.2.1 Generating template

- Execute %FIORANO_HOME%\esb\tools\templates\templates-console.bat [sh]
- From the command prompt launch the wizard using %FIORANO_HOME%/esb/tools/templates/edbctemplates.bat [sh]

Syntax: edbctemplates.bat [sh] –dest <dir>

<dir> - target directory where component code is generated.

Example: edbctemplates.bat –dest C:\SampleComponents\EDBC\SimpleJMS

- Provide service details in "Service Information" panel and click "Next".

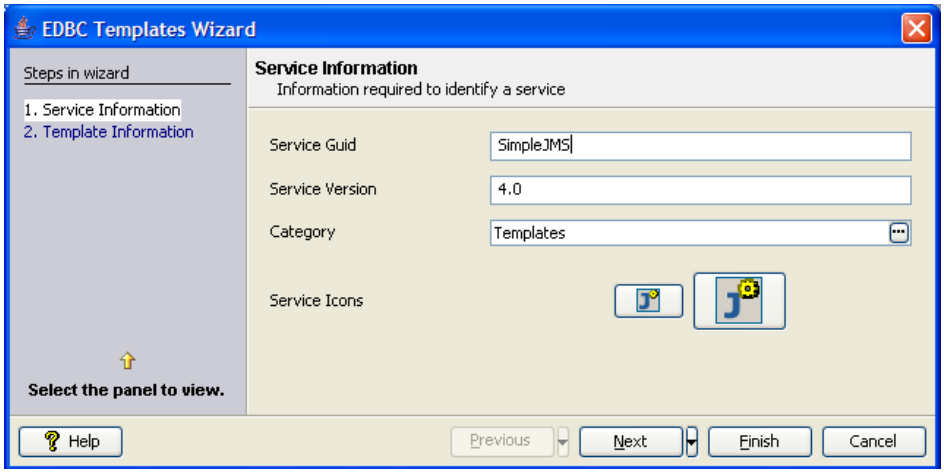


Figure 3.8.1: Details of service

- In **Runtime Arguments** panel add required runtime arguments.
- In **Template Information** panel Select **Simple JMS**.

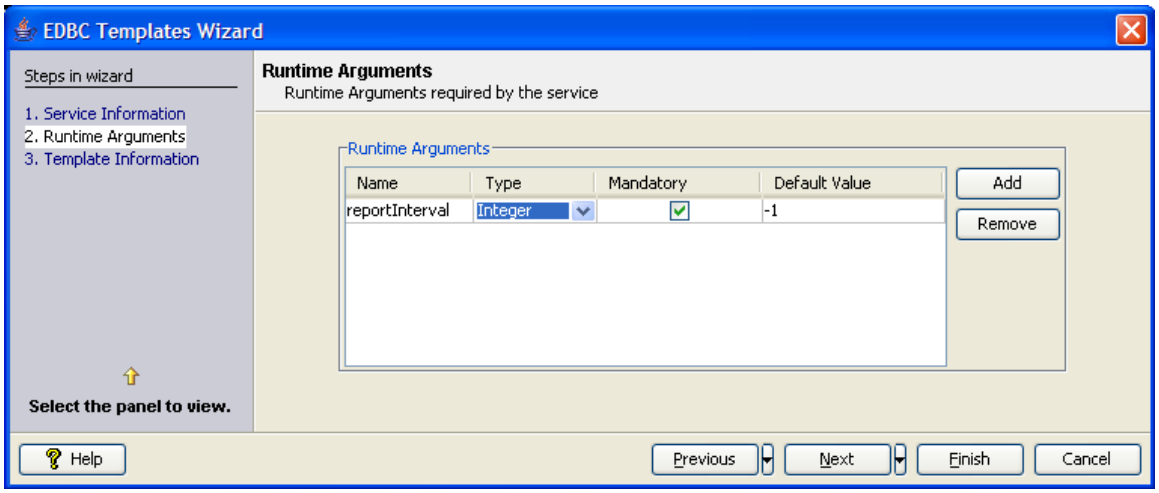


Figure 3.8.2: Adding runtime arguments

- Specify required number of input and output ports and click **Finish**.

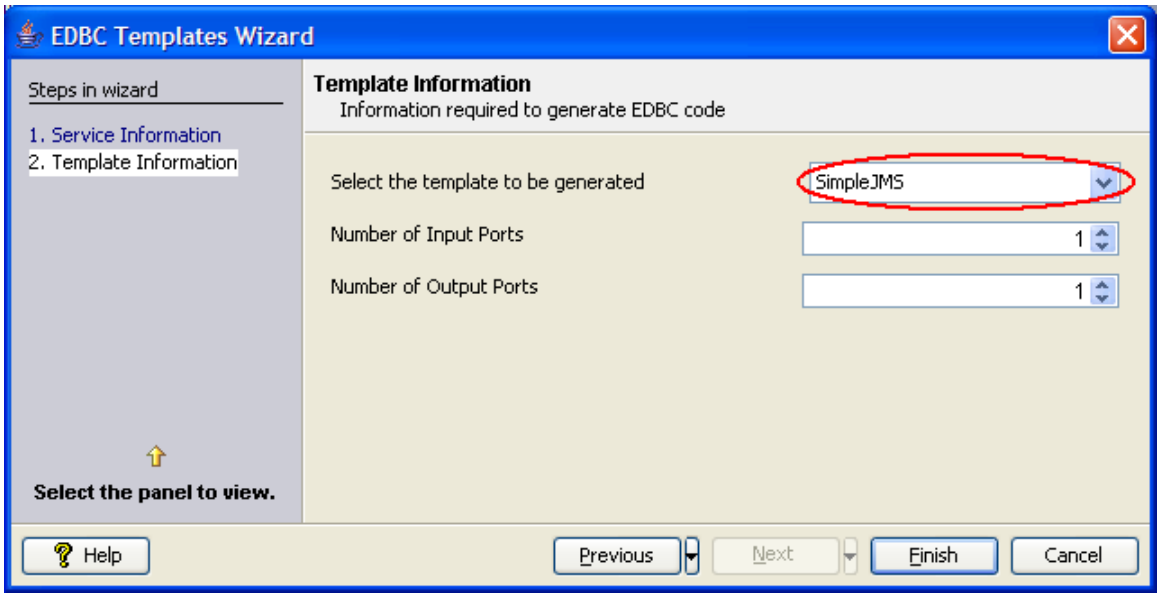


Figure 3.8.3: Specifying template to use

3.8.1.2.2 Component development

Adding business logic

Business logic should be added in onMessage(Message msg) method of ${ServiceGUID}MessageListener (in the above example SimpleJMSMessageListener) class

```
/**
 * Listens for messages on input port.
 * @param msg message received on input port
 */
public void onMessage(Message msg) {
    /**
     * todo: add the business logic here.
     */
}
```

Figure 3.8.4: ${ServiceGUID}MessageListener showing method where business logic should be added

3.8.1.2.3 Accessing Runtime Arguments

Runtime arguments passed to the service can be accessed using runtimeArguments.getParameter("<runtime argument>")

```
/**
 * Listens for messages on input port.
 * @param msg message received on input port
 */
public void onMessage(Message msg) {
    String value = runtimeArguments.getParameter("reportInterval");

    /**
     * todo: add the business logic here.
     */
}
```

Figure 3.8.5: Accessing runtime arguments

3.8.1.2.4 Sending Messages on output port

Messages can be sent on required output port using producer.send((Destination) outputDestinations.get("<output port name>"), <message>);

```
/**
 * Listens for messages on input port.
 * @param msg message received on input port
 */
public void onMessage(Message msg) {

    /**
     * todo: add the business logic here.
     */
    try {
        producer.send((Destination) outputDestinations.get("OUT_PORT_1"), msg);
    } catch (JMSException e) {
        e.printStackTrace();
    }
}
```

Figure 3.8.6: Sending message on output port "OUT_PORT_1"

3.8.1.3 Multi threaded

3.8.1.3.1 Generating template

- Execute %FIORANO_HOME%\esb\tools\templates\templates-console.bat [sh].
- From the command prompt launch the wizard using %FIORANO_HOME%\esb\tools\templates\edbctemplates.bat [sh].

Syntax: edbctemplates.bat [sh] –dest <dir>

<dir> - target directory where component code is generated

Example: edbctemplates.bat –dest C:\SampleComponents\EDBC\MultiThreaded

- Provide service details in **Service Information** panel and click **Next**.

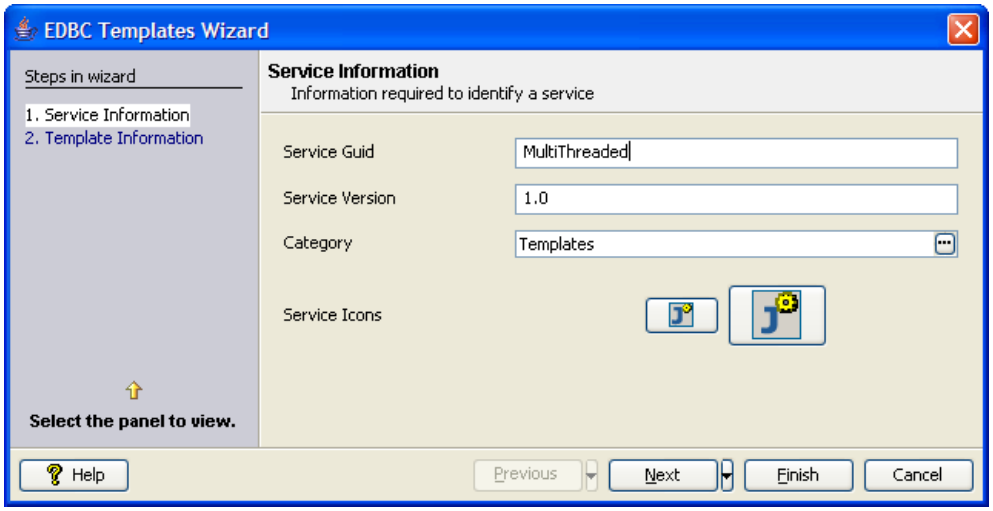


Figure 3.8.7: Details of service

- In **Runtime Arguments** panel, add required runtime arguments.

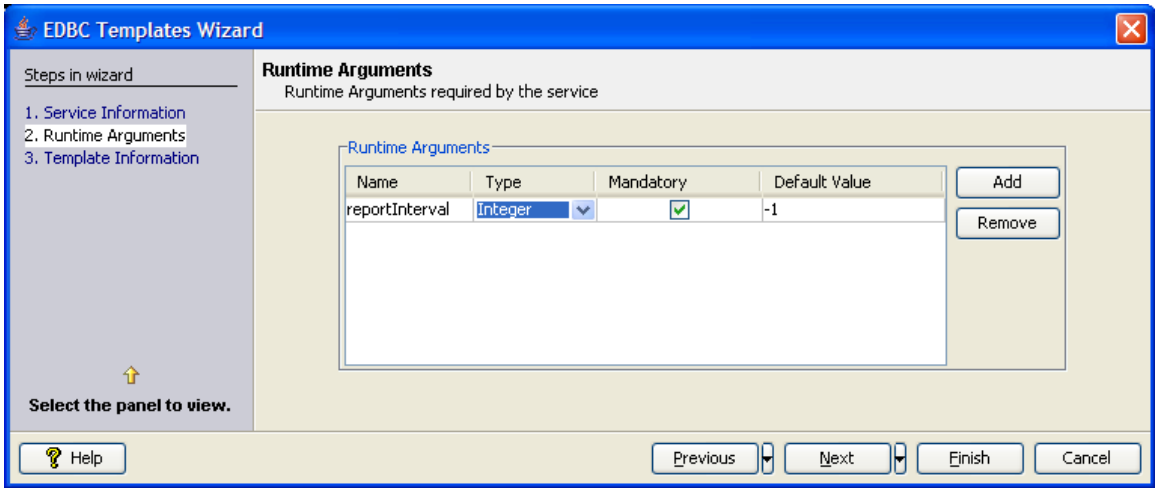


Figure 3.8.8: Adding runtime arguments

- In **Template Information** panel, select **MultiThreaded**.
- Specify required number of input and output ports and click **Finish**.

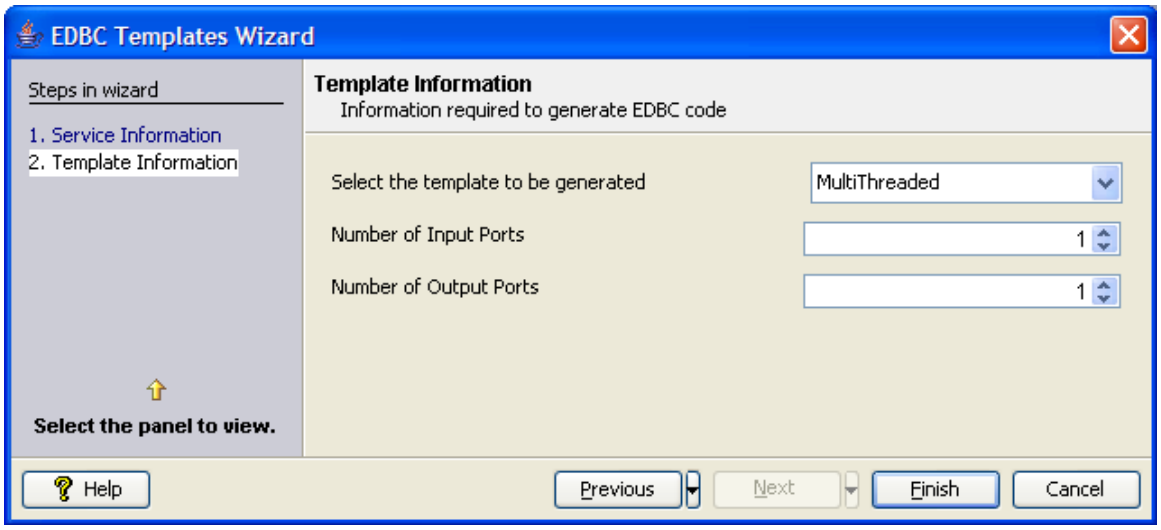


Figure 3.8.9: Specifying template to use

3.8.1.3.2 Component development

Adding business logic

Business logic should be added in execute() method of ${ServiceGUID}Job (in the above example MultiThreadedJob) class

```
/**
 * This method contains business. To send message onto output port a call to {@link #send(javax.jms.Message)}
 * should be made from this method.
 *
 * @throws TifosiException
 */
public void execute() throws TifosiException {

    //todo: add the business logic here
    //input message is available as "request" object
    //output message with all properties copied from input message can be fetched using getResponseDoc()

    super.execute();
}
```

Figure 3.8.10: ${ServiceGUID}job showing method where business logic should be added

3.8.1.3.3 Accessing Runtime Arguments

Runtime arguments passed to the service can be accessed using runtimeArguments.getParameter("<runtime argument>")

```
/**
 * This method contains business. To send message onto output port a call to {@link #send(javax.jms.Message)}
 * should be made from this method.
 *
 * @throws TifosiException
 */
public void execute() throws TifosiException {

    runtimeArguments.getParameter("reportInterval");
    //todo: add the business logic here
    //input message is available as "request" object
    //output message with all properties copied from input message can be fetched using getResponseDoc()
    //send(Message) must be called here

    super.execute();
}
```

Figure 3.8.11:Accessing runtime arguments

3.8.1.3.4 Sending Messages on output port

Messages can be sent on required output port using sender.send((<message>,"<output port name>");

```
/**
 * Sends message onto output port
 *
 * @param message <code>javax.jms.Message<code> that has to be sent on output port
 * @throws JMSException exception thrown when sending fails
 */
private void send(Message message) throws JMSException {
    synchronized(sender) {
        //todo add the send logic here
        sender.send(message, "OUT_PORT_1");
        sender.transactionComplete();
    }
}
```

Figure 3.8.12: Sending message on output port "OUT_PORT_1"

3.8.1.4 POJO

3.8.1.4.1 Generating template

1. Execute %FIORANO_HOME%\esb\tools\templates\templates-console.bat [sh].
2. From the command prompt launch the wizard using

   FIORANO_HOME%\esb\tools\templates\edbctemplates.bat [sh]

   Syntax: edbctemplates.bat [sh] –dest <dir>

   <dir> - target directory where component code is generated

   Example: edbctemplates.bat –dest C:\SampleComponents\EDBC\POJO
3. Provide service details in **Service Information** panel and click **Next.**

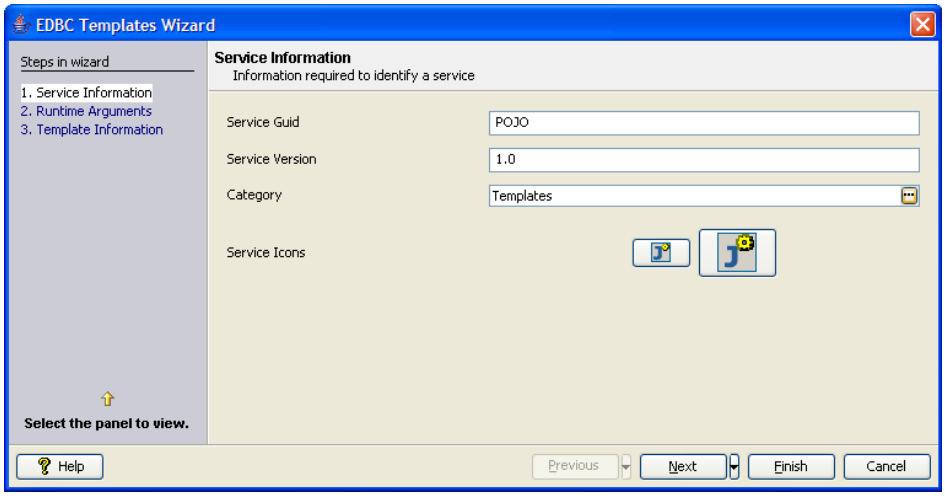


Figure 3.8.13: Details of service

4. In **Runtime Arguments** panel add required runtime arguments.
5. In **Template Information** panel, select **POJO**.

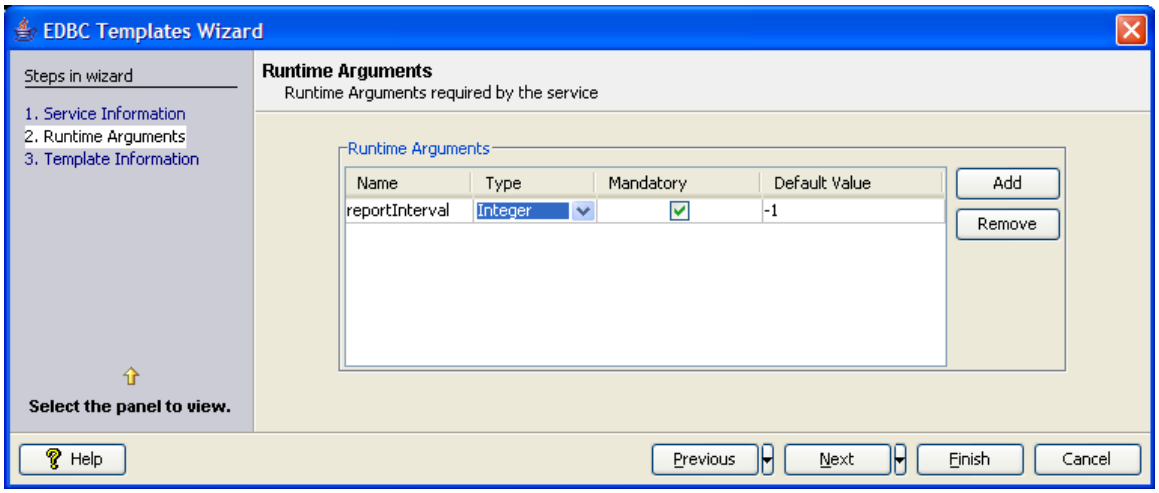


Figure 3.8.14: Adding runtime arguments

6. Select jar file containing the required POJO class.
7. Select the required method from list of methods shown.

   Only methods with following signatures is shown:

   public static void <method_name>(String)

   public static void <method_name>(Message)

   public static Message <method_name>(Message)

   public static Message <method_name>(String)

   public static String <method_name>(Message)

   public static String <method_name>(String)

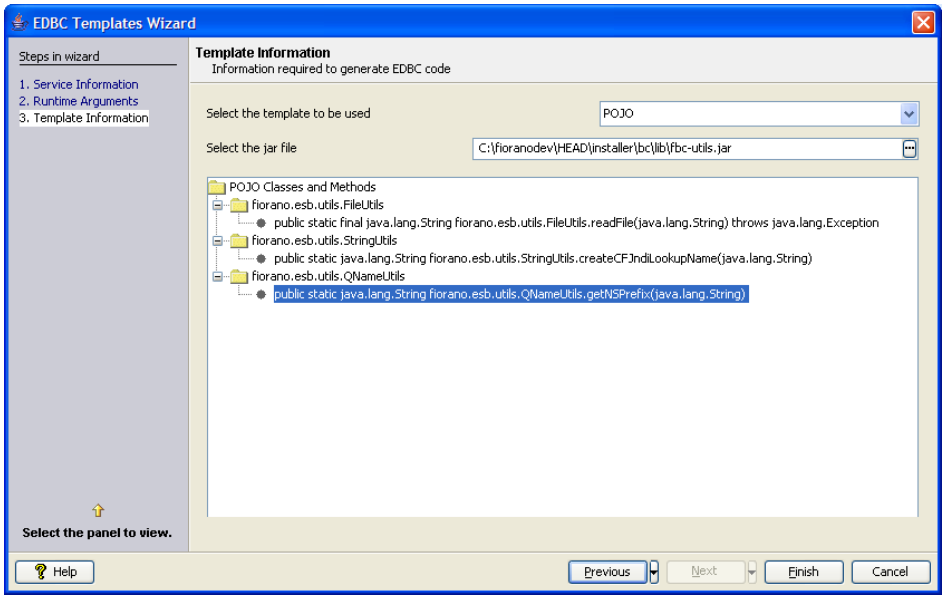


Figure 3.8.15: Specifying template to use and selecting required method

8. Component generated for a POJO will contain one input output pair for each method selected. However, if the return type is void output port will not be generated for that method.
9. When a message arrives on an input port associated method is invoked and response is sent to respective output port if present.

3.8.1.4.2 Component development

Business logic is generated into component and can be used as it is.

3.9 Eclipse IDE Support

Fiorano template engine also creates eclipse project which can be imported in eclipse IDE. The component can be modified, compiled and deployed to Fiorano Enterprise Server from eclipse. This section describes the process in detail.

3.9.1 Importing the Project into Eclipse

To import a project into eclipse:

1. Click **Import...** menu action from **File** menu as shown in the Figure 3.9.1

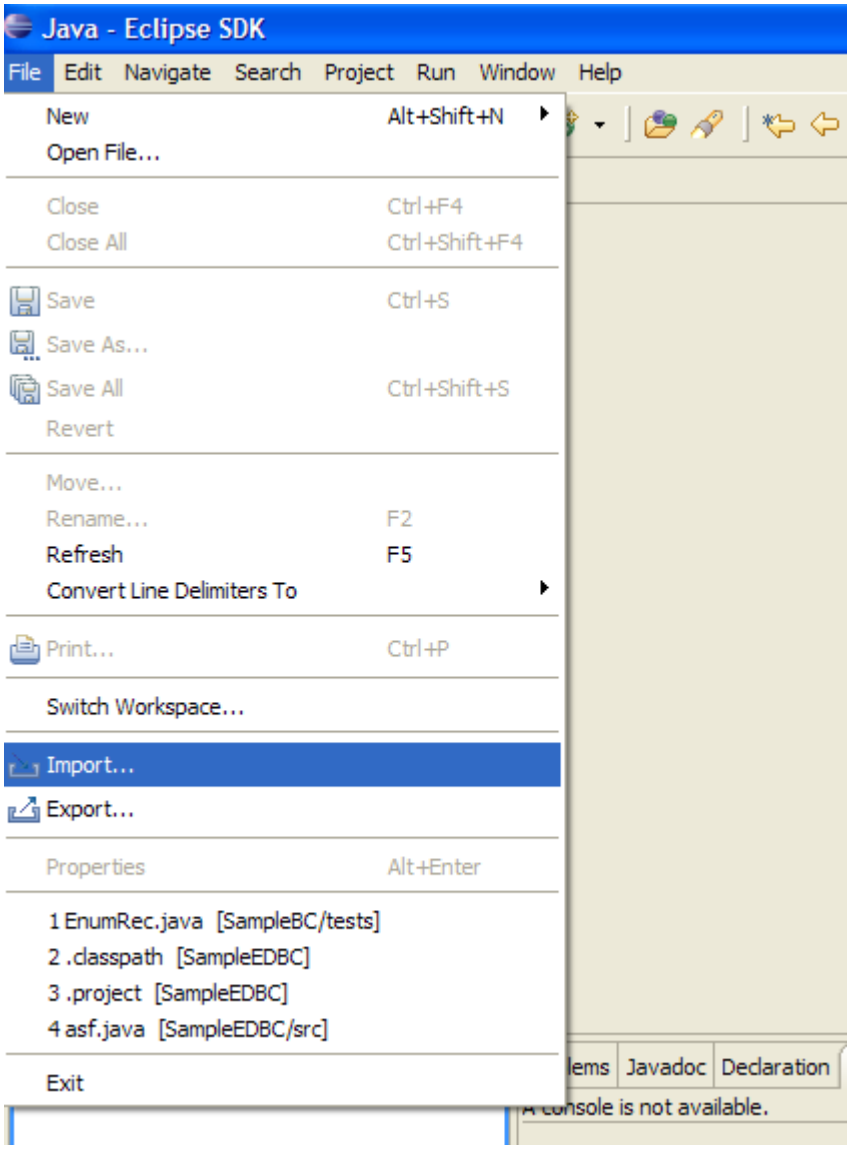


Figure 3.9.1: File menu showing import action

2. A wizard which guides through steps for importing a project is shown. Choose import source as **Existing Projects into Workspace** and click the **Next** button. See Figure 3.9.2.

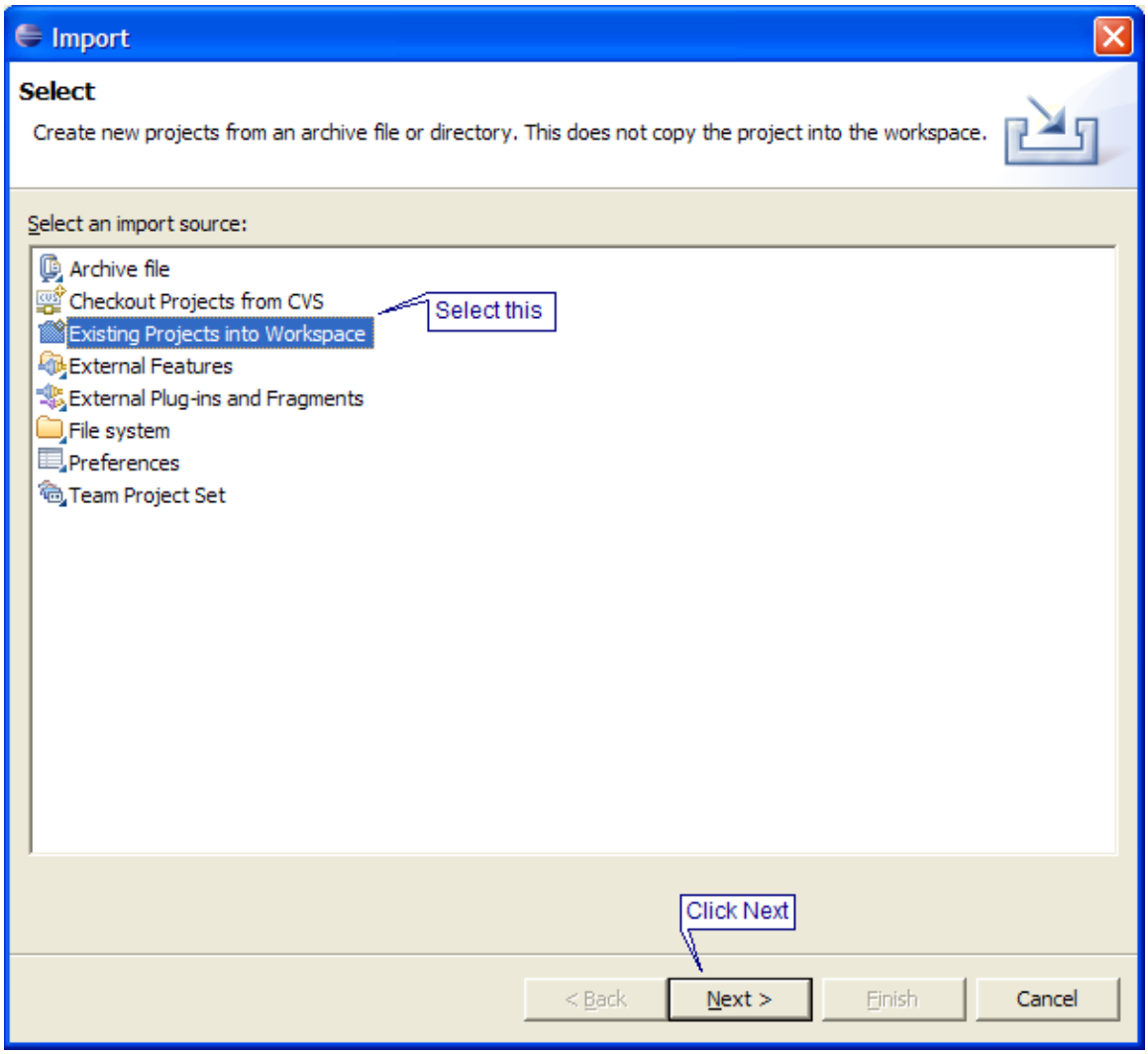


Figure 3.9.2: Selecting import source

3. Select **Select root directory** and provide the component's directory shown in Figure 3.9.3.

4. Click **Finish**.

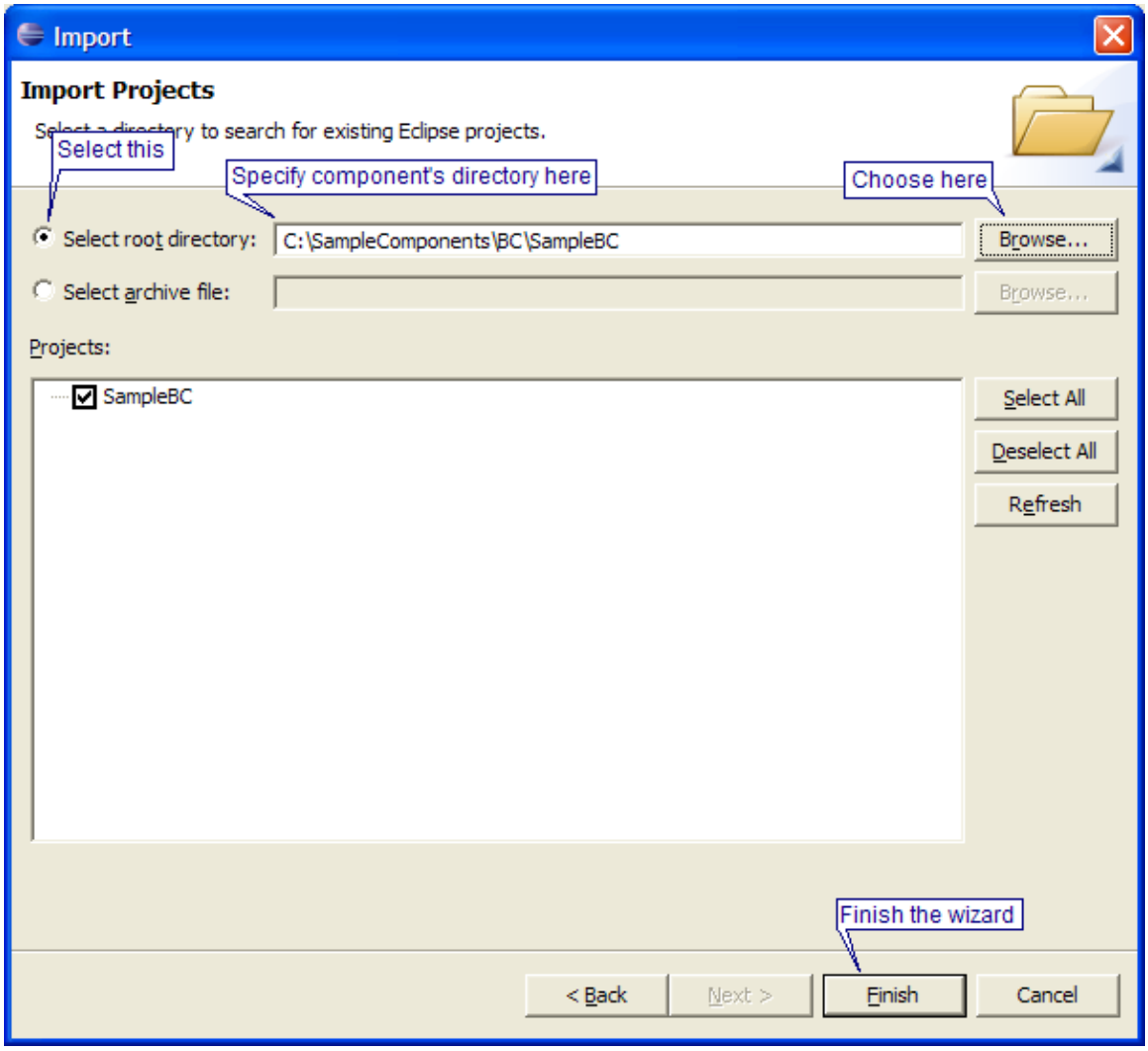


Figure 3.9.3: Specifying root directory

The project's directory structure is shown in **Navigator** pane and **Problems** pane shows errors and warnings present in the project (Figure 3.9.4)

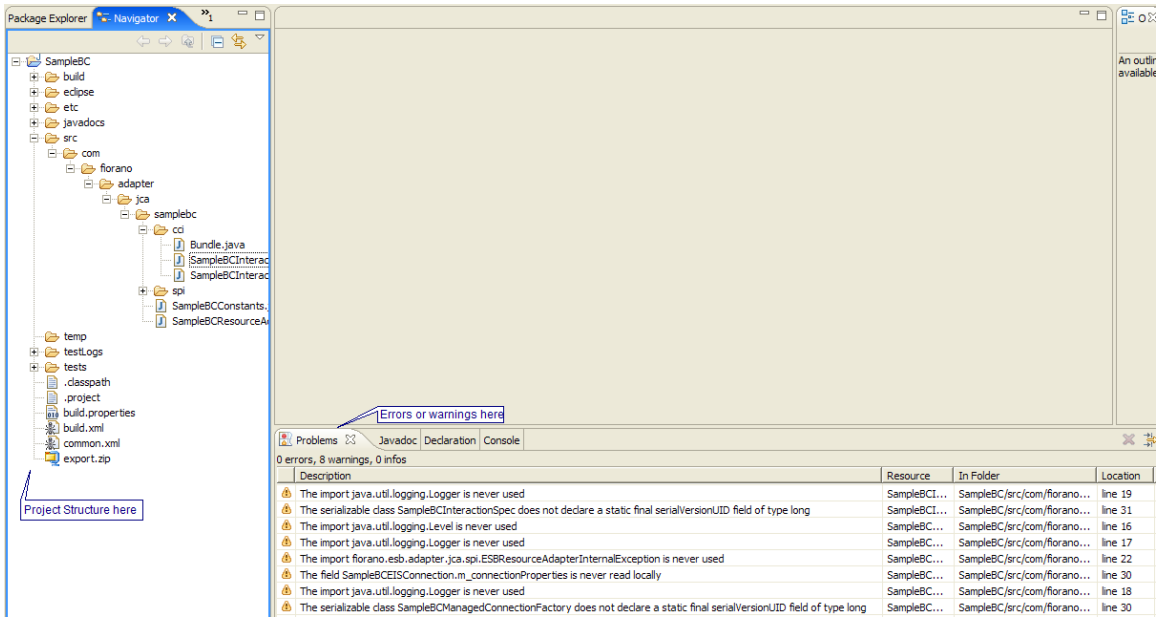


Figure 3.9.4: loaded project structure in Navigator window and warnings/errors in Problems window

5. Project import is completed. Sources can be modified as desired to include business logic.

3.9.2 Defining Variables

FIORANO_HOME classpath variable should be added to project's classpath. Steps to add any classpath variable are given below

1. Right-click on the project root (SampleBC)
2. Click on **Properties** menu item in the popup menu (shown in Figure 3.9.5)

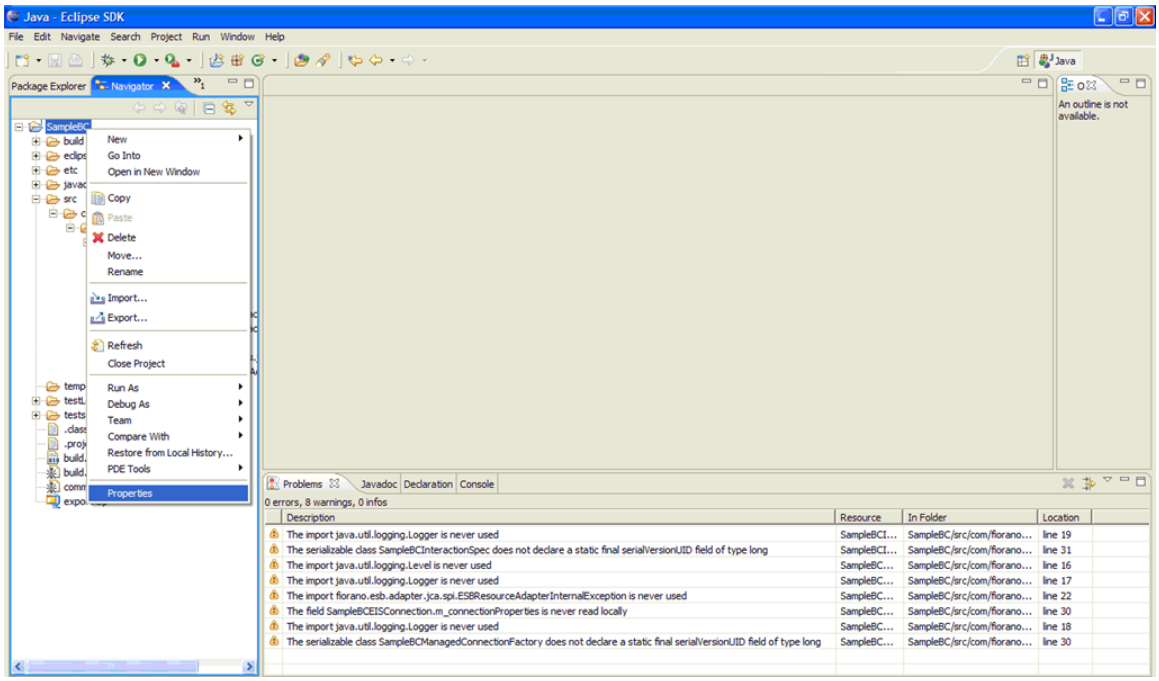


Figure 3.9.5: Opening properties of the project

3. The properties window for the project comes up
4. Select **Java Build Path** in the tree on the left side.

5. Switch to **Libraries** tab on the right side, see Figure 3.9.6.

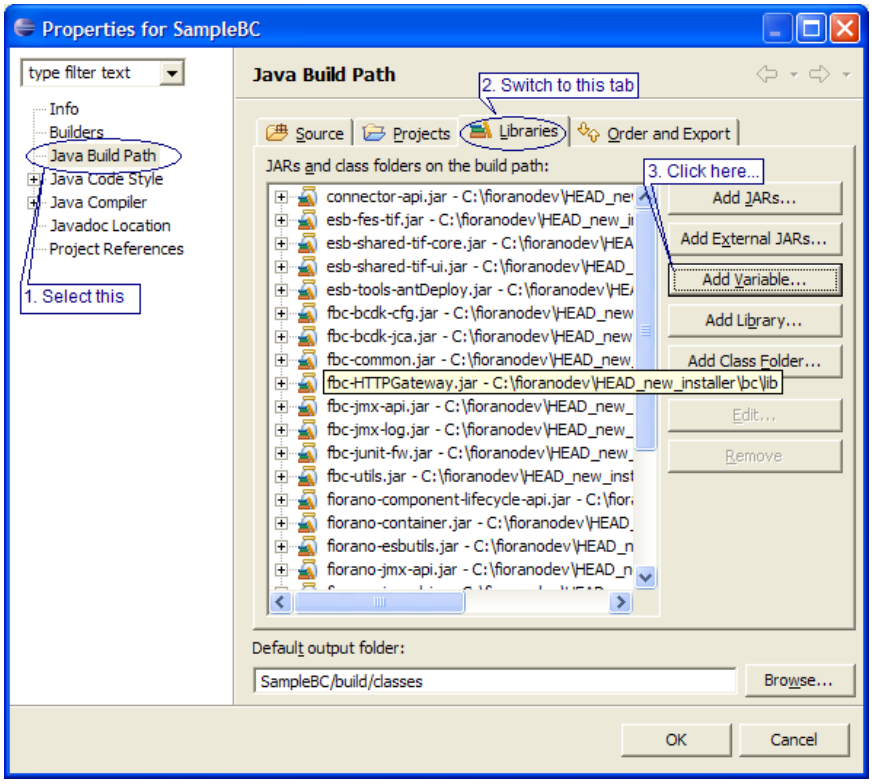


Figure 3.9.6: Properties window of project. Location to add variables shown

6. Click the **Add Variable ...** button
7. A dialog containing defined variables which can be added to build path is shown. Click on the **Configure Variables...** button. To add new variables or edit the existing variables, see Figure 3.9.7.

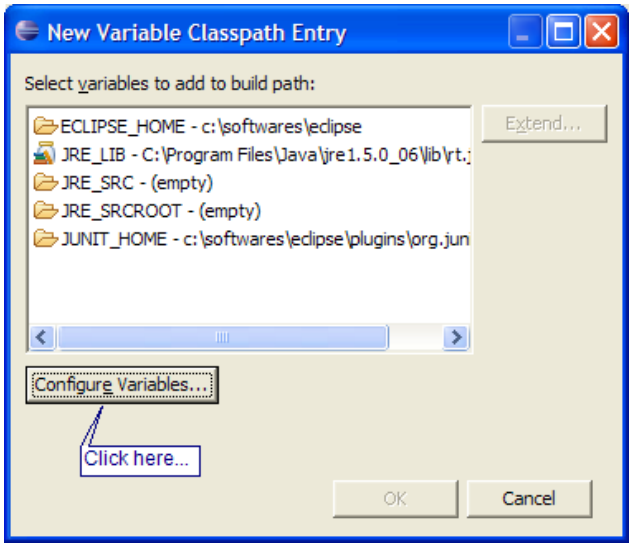


Figure 3.9.7: Dialog showing defined variables

8. A dialog with a list of variables is shown. To add new variables click on the **New** button, see Figure 3.9.8.

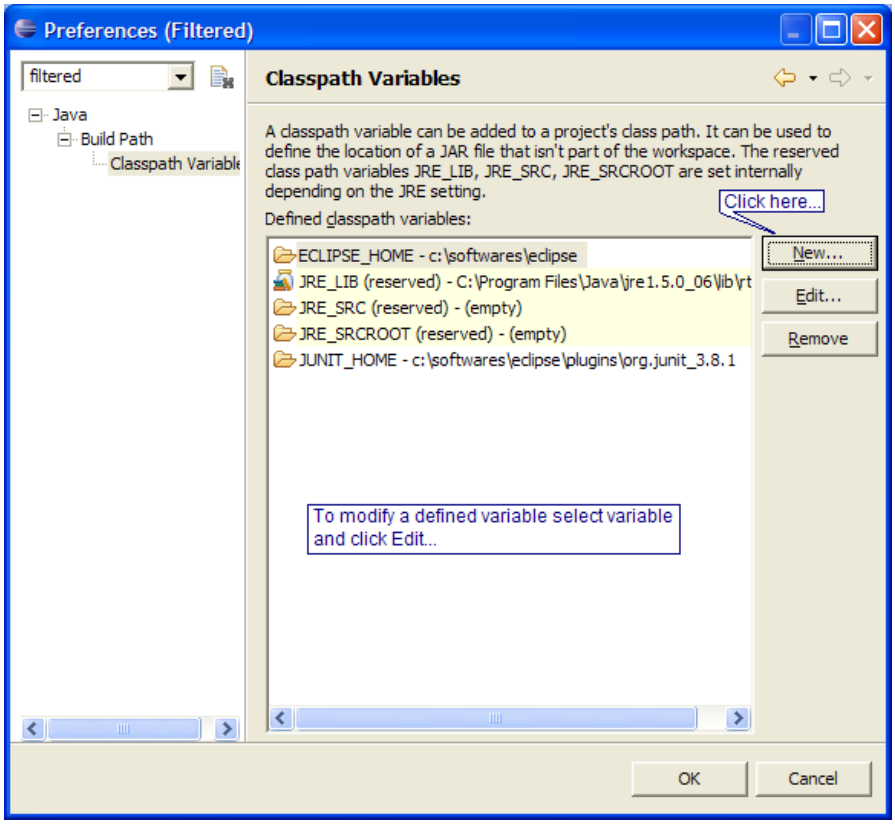


Figure 3.9.8: Add new variables or edit defined variables

9. Specify the variable name (FIORANO_HOME) and path (point to %FIORANO_HOME%) in the dialog that comes up. See Figure 3.9.9.

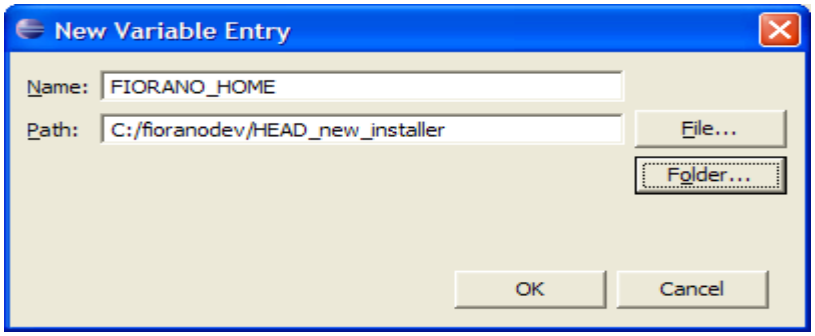


Figure 3.9.9: define the variable

10. Close all the windows. Make sure the new variable is selected when closing the window.

3.9.3 Defining ANT_HOME

To build and deploy the component the ANT_HOME should point to ant shipped in the installer. This can be done as follows

1. Click **Preferences...** on the Window menu as shown in Figure 3.9.10.

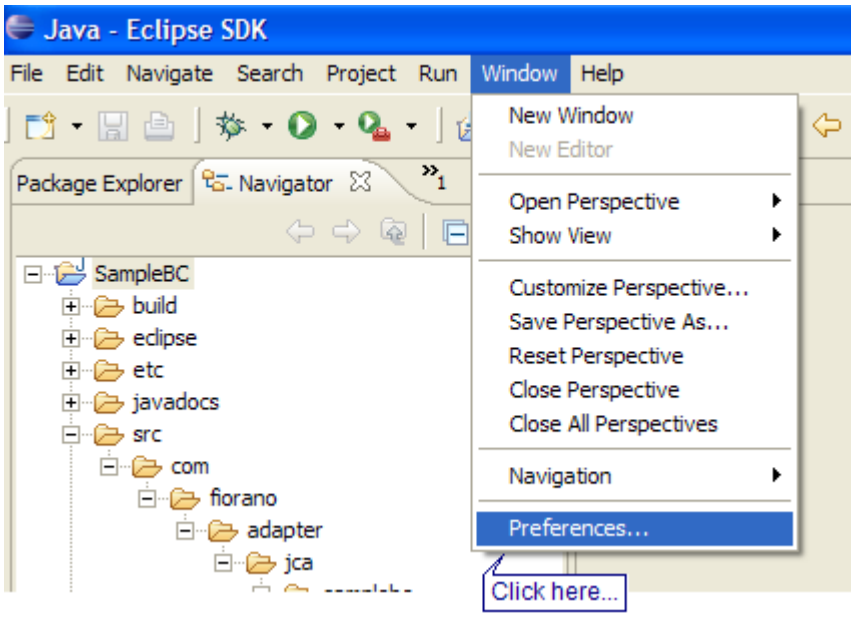


Figure 3.9.10: Opening preferences

2. Select **Runtime** node under **Ant** in the tree on left side.
3. Select **Classpath** tab on the right side as shown in Figure 3.9.11.

4. Click on the **Ant Home** button. This opens a window where the directory or file can be chose.

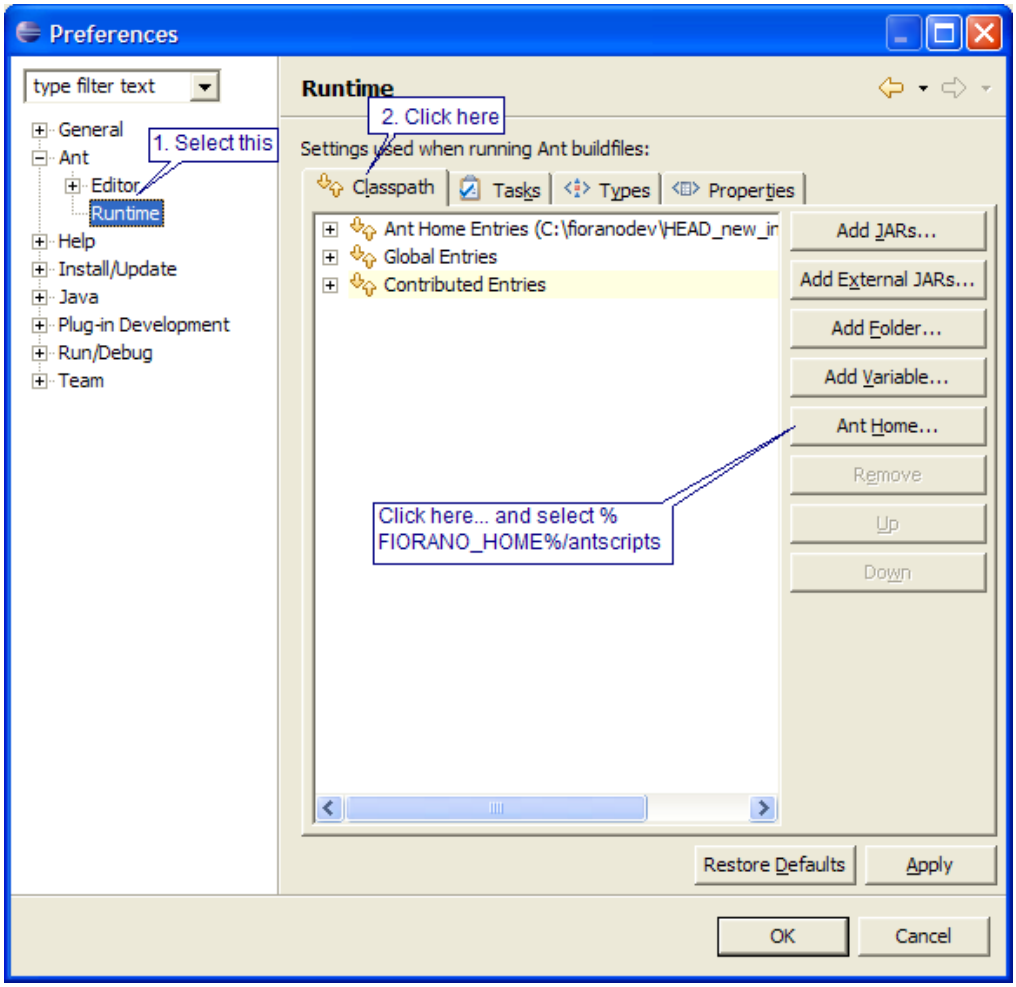


Figure 3.9.11: Configuring ant home in preferences

5. Select %FIORANO_HOME%\antscripts as shown in Figure 3.9.12

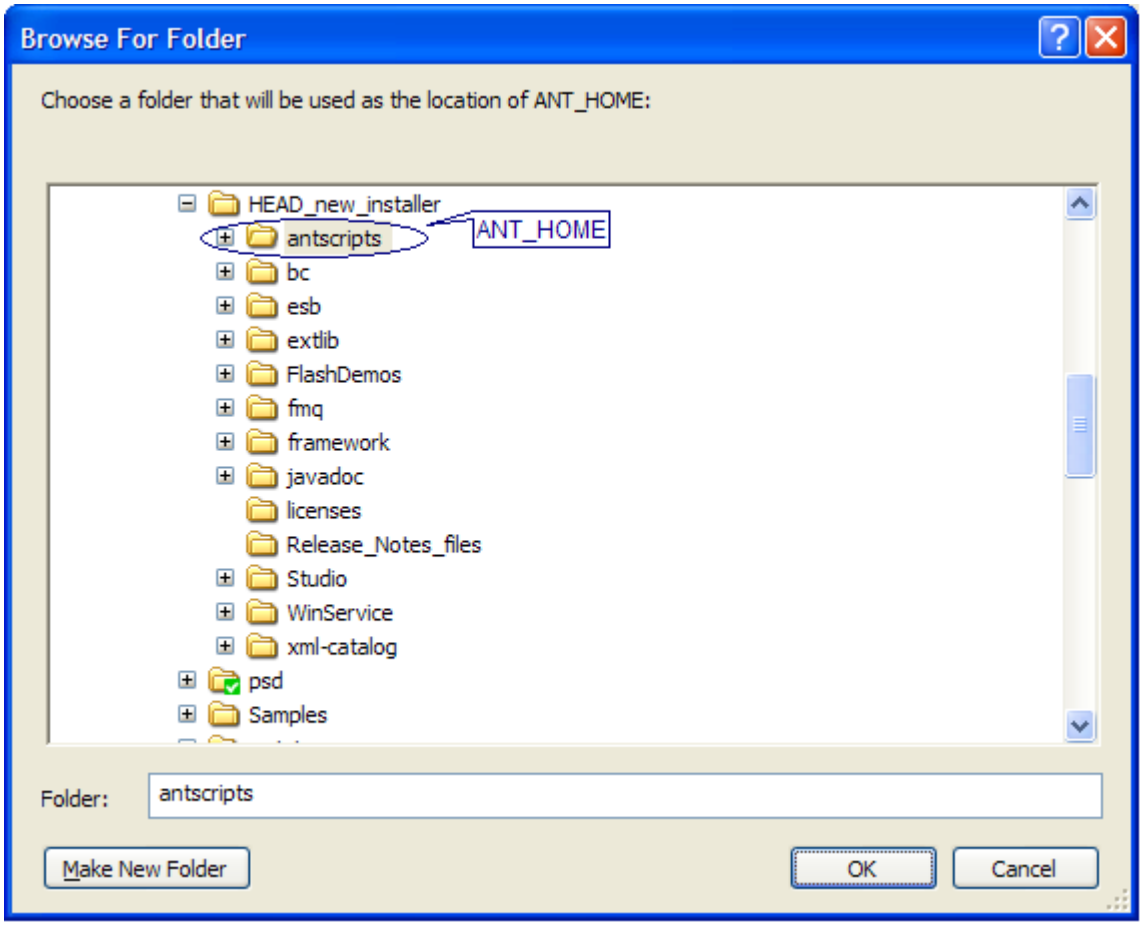


Figure 3.9.12: Selecting ant home

6. Close all the windows.

3.9.4 Defining JDK

To be able to compile the sources, JDK should be used and NOT JRE. To make the installed JRE entry point to JDK do the following

1. Set the installed JRE which is used to JDK.
2. To set installed JRE click **Preferences** from **Window** menu as shown in Figure 3.9.10.
3. In the preferences window, select **Installed JREs** under Java node as shown in Figure 3.9.13.
4. Click the **Add** button to add a new entry point which points to JDK or click the **Edit** button to modify the existing to point to JDK.

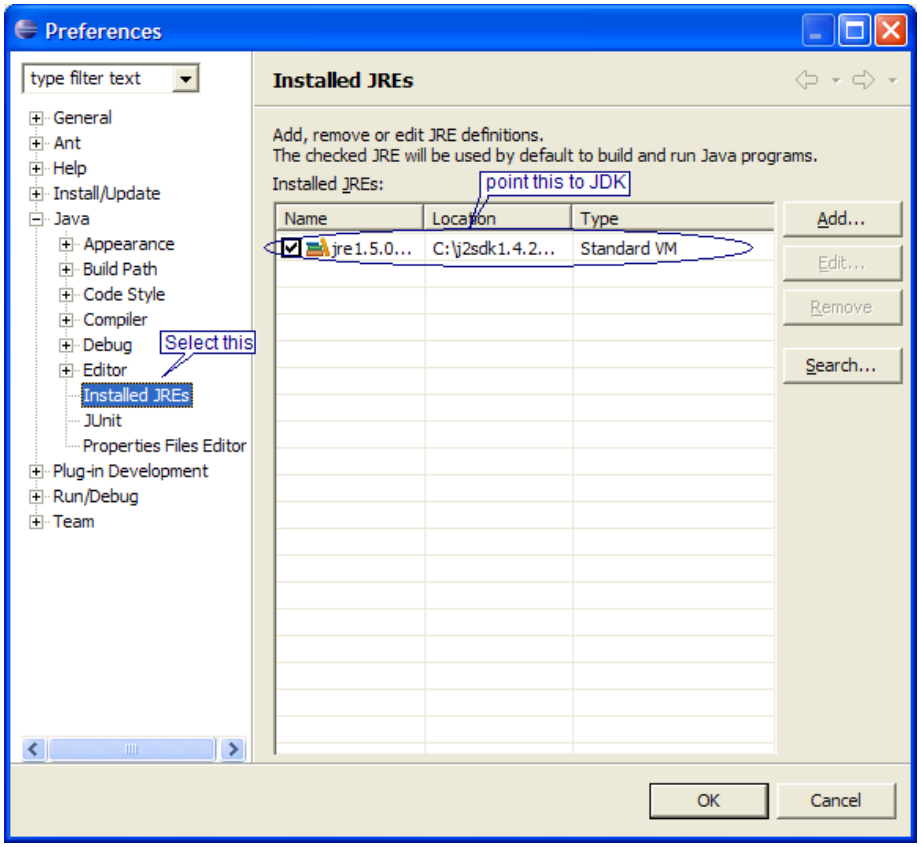


Figure 3.9.13: Adding JDK

3.9.5 Compiling Deploying and Registering the Component

To compile, deploy and register the component follow the below steps.

1. In the **Navigator** pane right-click on the build.xml file
2. From the popup menu select **Run As** → **External Tools** as shown in Figure 3.9.14.

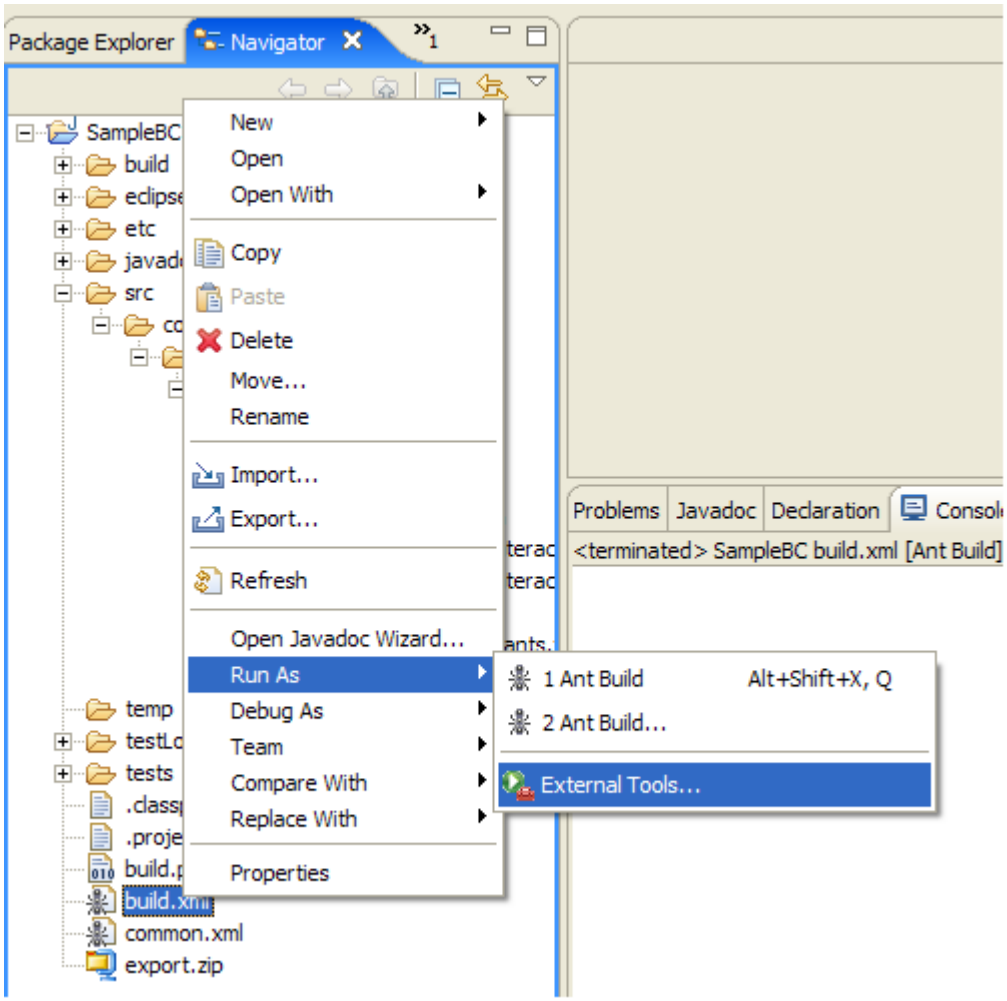


Figure 3.9.14: running ant using external tool

3. An **External Tools** window appears. Select the component's build file under **Ant Build** node in the tree shown in left pane.

4. Select **JRE** tab and choose the JDK as shown in Figure 3.9.15.

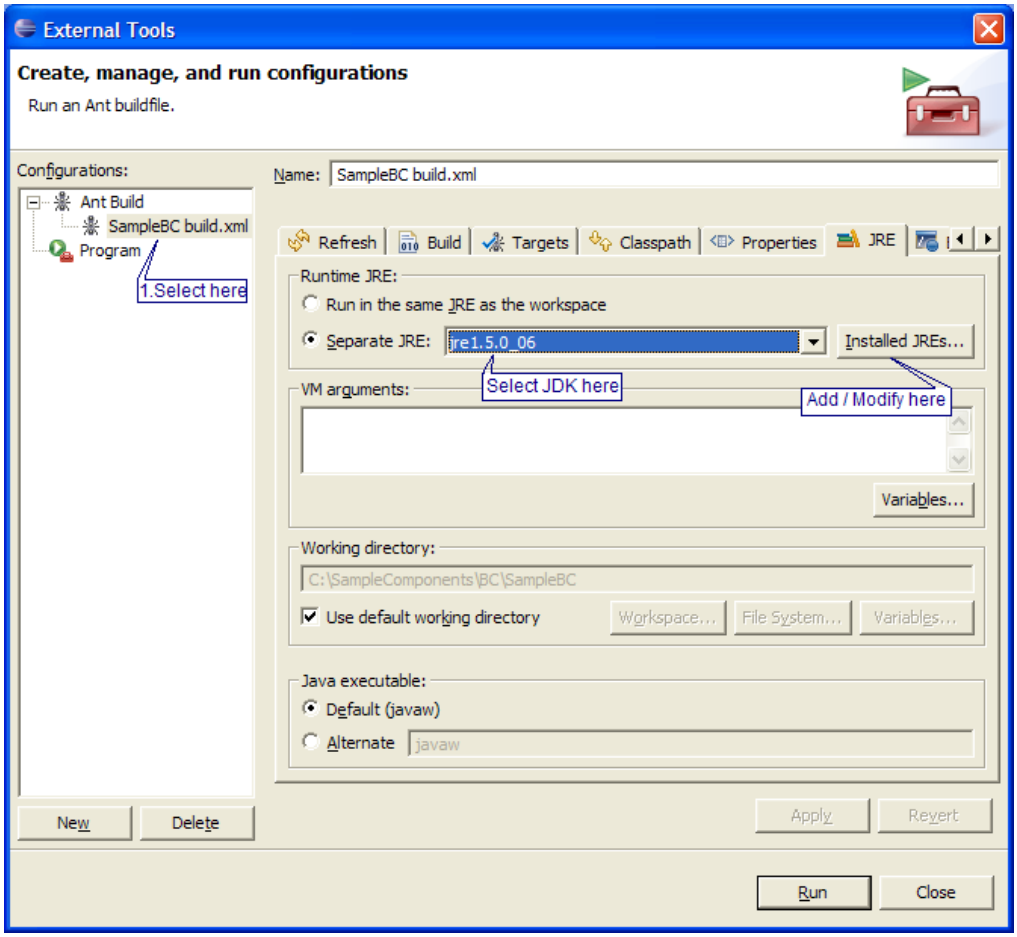


Figure 3.9.15: using JDK

5. Select **Targets** tab to select the target from the ant file which has to be executed. See figure 3.9.16
6. Select appropriate target from among the targets listed.
   a. **deploy** – compile and copy the jars to installer
   b. **register** – deploy and register the component with FES
   c. **reregister** – deploy and register the component with FES overriding existing component
   d. **unregister** – unregister the component from FES. It will no longer be available for use in flows

e. **cleanup** – removes all the class files, so that during the next build all the classes is compiled again

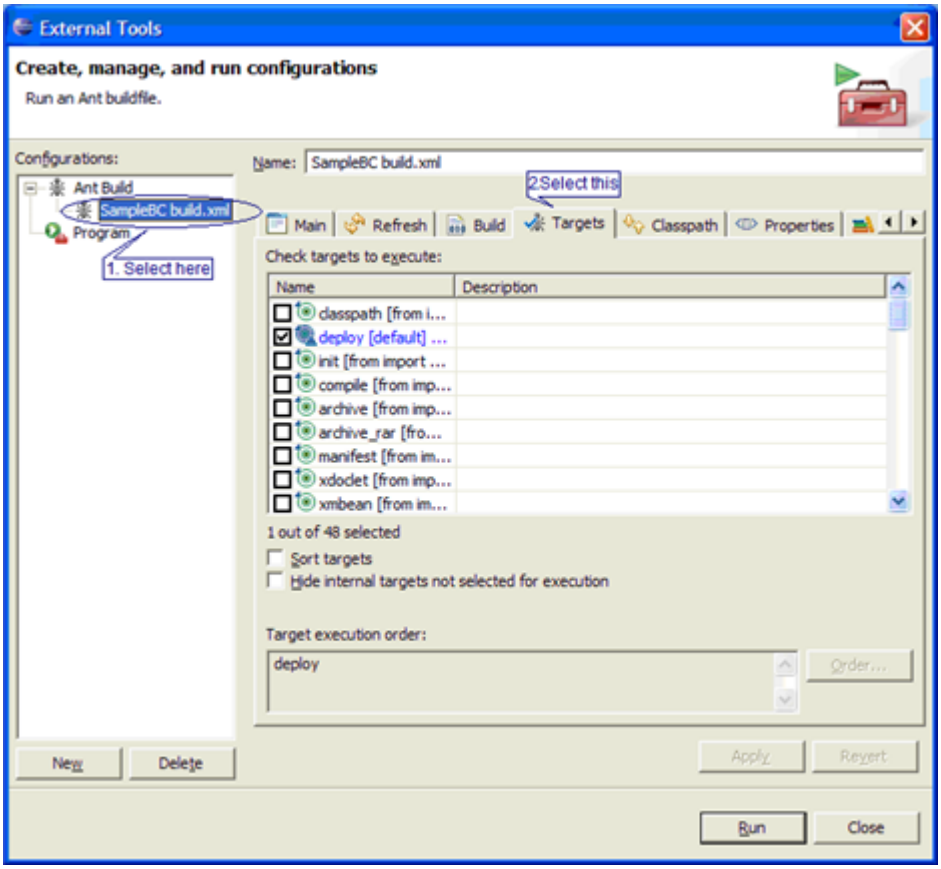


Figure 3.9.16: ant targets which can be executed are shown

Result of the task is shown in the console of eclipse as shown in Figure 3.9.17.

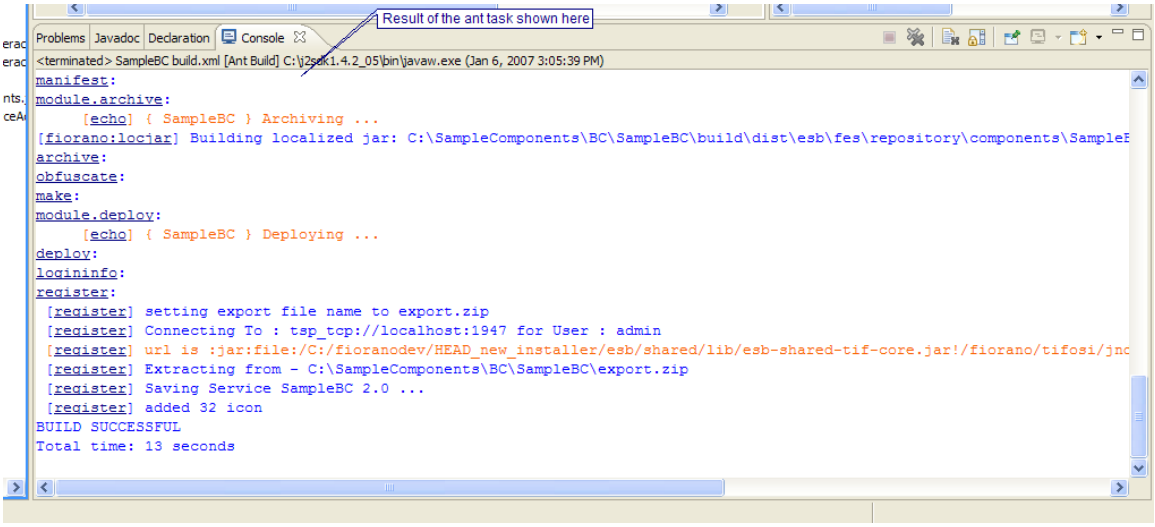


Figure 3.9.17: Result of Ant Task

3.10 Text Schema Editor

This chapter describes the Fiorano Text Schema Editor tool which is used to design XML schemas as **.tfl Text Format Layout files**. These TFL files are used by the XML2Text and Text2XML prebuilt components to fascilitate data conversion of non-XML data from and to its corresponding XML format respectively.

In case you require your composite component flow to read or write data from your data repository which exists as text or flat-files, you can use the FileReader component to read this flat-file and transform flat-file data into its corresponding XML using the Text2XML component. The opposite can be done using a combination of the XML2Text component and the FileWriter component. But before you can transform data from flat-file format into its corresponding XML or vice versa, you require defining a File Schema which can aid the transformation. This File Schema may be understood as the format meta-data that is required in both the above mentioned instances.

The Text Schema Editor (TSE) is a tool which assists you to visually define the format and hierarchy of the non-XML data graphically. The format structure created by this editor is called the File schema in which the structure of the non-XML data is defined in terms of records and fields. This format is stored using XML grammar in **tfl** (Text Format Layout) files.

Note: The Schema defines the rules used to convert non-XML text to XML text and vice versa.

Once this schema format is defined, it can be used by

- Text2XML transforms flat-file data to its corresponding XML
- XML2Text transforms XML data into its corresponding flat-file format

The following diagram shows how the FileReader component uses the transformation components to read XML and non-XML data.

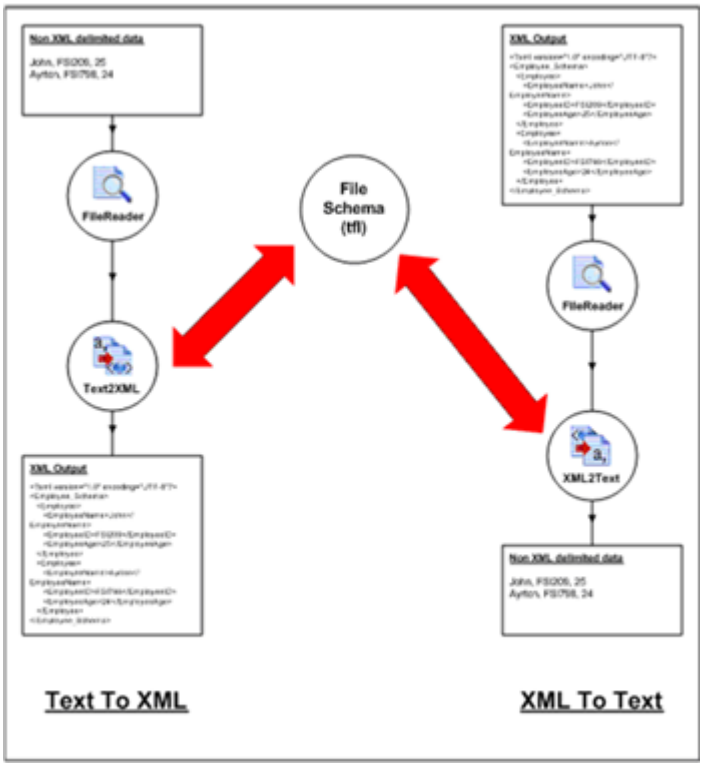


Figure 3.10.1: Using FileReader and Transformation components

The non-XML data mentioned above can be delimited, positional or both. TSE also provides the test functionality in which the user can verify and test the schema formats created. In the test functionality, the user can generate sample data and can also transform sample non-XML data to XML and vice versa.

3.10.1 Text Format Layout Concepts

A *tfl* (text format layout) document is a specialized XML grammar which is used to describe the structure of non-XML structured (delimited, positional) data. In the *tfl* document, the structure of the data is defined as a hierarchical tree of records and fields in a given order.

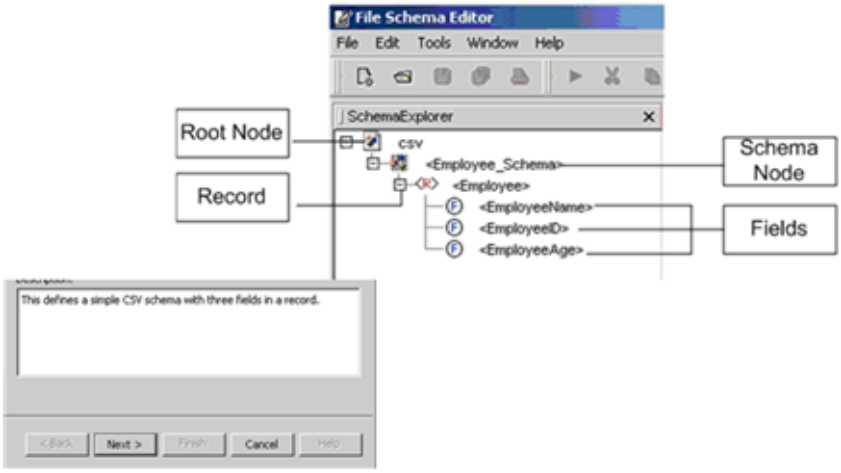


Figure 3.10.2: Structure of a tfl document

The schema of the structured data is added as a child node to this Root Node. This node is called the *Schema Node*.

When you create a new schema in Fiorano Schema Editor, the Root Node and the Schema Node are created automatically.

Schema Node: In the schema structure, each opened schema file is shown as Schema Node and is the child of the Root Node. The Schema Node corresponds to the Root tag of the output XML which is generated from the structured non-XML text or input XML which is to be converted to the structured non-XML text. Schema Node can also be renamed. The properties of the Schema Node represent the default properties which can be used during data transformation. In a Schema Node, you can add multiple record nodes which represent the structure of input/output data. Adding fields to the Schema Node is not allowed.

Record: Record represents a collection of information. It can contain a set of fields and/or other records.

Field: Field represents items of information that are simple in nature, such as strings and numbers.

3.10.2 Launch Fiorano Text Schema Editor

The FileReader and FileWriter components facilitate reading and writing of both XML and non-XML files. So as to enable your component flow to utilize the accessed data, you may need to use Text2XML or XML2Text components to transform flat-file data into XML data and XML data into flat-file data respectively.

These transformation components require a **tfl** file which is a schema of the data that is to be accessed. To create this schema you may need to use the Text Schema Editor.

The following steps may be used to create a new schema:

1. Launch Fiorano Studio from the Start menu, click **Start** > **Programs** > **Fiorano** > **Fiorano SOA Platform** > **Fiorano Tools** > **Fiorano Studio**.
2. From the toolbar, click on **File** > **New File.** The Choose File Type dialog box appears:

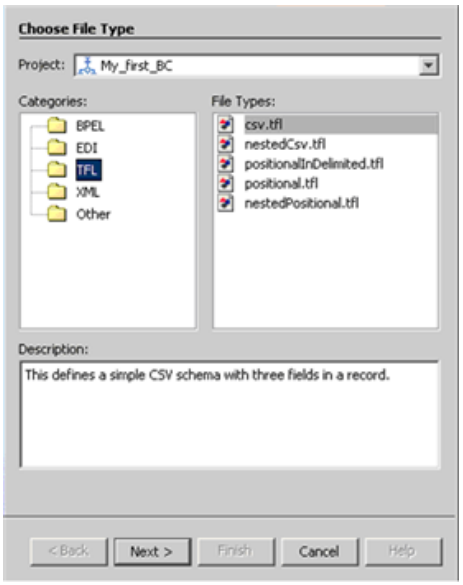


Figure 3.10.3: Creating a New File: Text Schema Editor

3. Click on TFL in the Categories easel.
4. Click on the type of schema you intend to use in the File Types easel.
5. Click on **Next**. The Name and Location dialog box is displayed:

Figure 3.10.4: Creating a New File: Text Schema Editor - New CSV File

6. The File Name field may be populated with the name you intend to use for the schema file (tfl).
7. The path where you intend to store this file should be supplied in the Folder field.
8. Click **Finish** to save you configuration settings and create a new tfl schema file.

This **.tfl** file can be invoked by the transformation components to transform non-XML data into its corresponding XML and vice versa.

3.10.2.1 Defining Text File Schemas

This tool is shipped with five samples that represent various schema types. These are broadly classified under two categories, namely Delimited File Schema samples and Positional File Schema samples. The prebuilt schema samples are given below:

- Delimited File Schema samples
  - CSV File Schema
  - Nested CSV File Schema
- Positional in Delimited File Schema
  - Positional File Schema samples
  - Positional File Schema
  - Nested Positional File Schema

As mentioned earlier, the schema of a file is represented by a tree structure. The tree format is shown in the Figure 3.10.5.

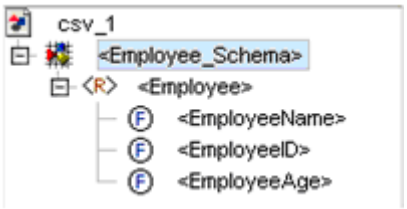


Figure 3.10.5: Schematic representation of a file schema

3.10.2.1.1 Schema Node Properties

The Schema Nodes of all the file schemas represent the same set of properties. These properties act as global properties of the file schema which are available to all the descendent records and fields.

Note: If you change the Name value on the Properties panel, the name of the Root Node in the specification tree automatically changes to match it and vice versa. The name of the node should be a valid XML name.

The following table lists all properties associated with the Schema Node:

Property	Value
Comment Start Identifier	An identifier which indicates the start of a comment in the source file.
Comment End Identifier	An identifier which indicates the end of a comment in the source file. The data between the 'Comment Start' and 'Comment End' identifier is ignored. **Note:** Comment Start and Comment End Identifiers must not be identical.
Name	This is the name of the Root Node.
Description	This is the description of the specification.
Delimiter Value	Type or select a value for the delimiter. To specify a delimiter value, you must first set the Delimiter Type to Custom Delimiter. The delimiter can be multi-character.
Escape Character	Specifies the default value of the escape character for this Schema Instance. Type or select a character value for the escape character. To specify an escape character value, you must first set the Escape Type to Character.
Delimiter Type	Select one of the following options to choose a delimiter for the records/fields directly below the current record. • **Default Field Delimiter** Indicates that the delimiter is the value of the Default Field Delimiter property, which is defined for the schema instance. • **Custom Delimiter** Allows the user to designate a field delimiter value for the record. If you select **Custom Delimiter**, you must specify a delimiter value.

Property	Value
Escape Character Type	You can choose the escape character type from the following values: Default Escape Character - Indicates that the escape character is the value of the Default Escape Character property which is defined for the schema instance. **Character -** Allows the user to designate an escape character value. If you select **Character**, you must also specify an escape value. An escape character is useful if you have a character in your field data that is also used as the delimiter character in the field's parent record. For example, if your field data is the following and you have chosen a comma as the delimiter value of the record that contains the field, TSE interprets the comma after "Fiorano" to be a delimiter, even if you intend for it to be part of the field data: Fiorano,Software,USA Solution for this is to place an escape character directly preceding the delimiter character that you want to include in the field data. For example, if your escape character is specified as a backslash, you can place a backslash directly preceding a delimiter character, as in the following example: Fiorano\,Software,USA TSE interprets the comma after the backslash as field data rather than a delimiter character.
Escape Character	This is the escape character which is to be used as the field delimiter.
Delimiter Value	Type or select a value for the delimiter. To specify a delimiter value, you must first set the Delimiter Type to Custom Delimiter. The delimiter can be multi-character.

Property	Value
Escape Character Type	You can choose the escape character type from the following values: Default Escape CharacterIndicates that the escape character is the value of the Default Escape Character property which is defined for the schema instance. **Character**Allows the user to designate an escape character value. If you select **Character**, you must also specify an escape value. An escape character is useful if you have a character in your field data that is also used as the delimiter character in the field's parent record. For example, if your field data is the following and you have chosen a comma as the delimiter value of the record that contains the field, TSE interprets the comma after "Fiorano" to be a delimiter, even if you intend for it to be part of the field data: Fiorano,Software,USA Solution for this is to place an escape character directly preceding the delimiter character that you want to include in the field data. For example, if your escape character is specified as a backslash, you can place a backslash directly preceding a delimiter character, as in the following example: Fiorano\,Software,USA TSE interprets the comma after the backslash as field data rather than a delimiter character.
Delimiter Type	This is the field delimiter of this file schema. The delimiter can be multiple characters.

3.10.2.1.2 Record Node Properties

Every file schema is a unique entity, with a unique set of records and fields. You can create a new schema by modifying an existing schema. To modify an existing schema, you need to add and/or remove records. After adding records, you must specify the properties associated with it. If you remove a record, its properties are also removed, along with all child records and fields. In addition to adding and removing records, you can also rename them. You can edit the name of an existing record and its properties by selecting the record and editing it. The tool is also designed to handle duplicate records. If you paste a record into a schema, which already has a record by a similar name, the new record is added and a number is appended to the end of the name of the record you are pasting. For example, when you paste a record named Department to a schema, which already has a record by that name, the new record is added to the schema with the name Department_1.

Following are some basic rules pertaining to records.

- Every new record, which you create, is inserted as a descendant of the record that you selected.
- The name of a record or field needs to be unique. The tool will display an exception if you specify a name that has already been assigned to an existing record or field.
- When you delete a record, all child records and fields are also deleted.

The following table lists all the properties associated with the record node:

Property	Value
XML Type	The target XML type for the field. Depending on this value, the tag in the resultant XML is generated. Its value can either be Element (default), Attribute or None. If 'None' is selected, then the field is NOT mapped to the resultant XML.
Minimum Occurrences	The minimum number of occurrences specified for a particular record. If the record does not occurs the specified number of times, then an exception is thrown.
Maximum Occurrences	The maximum number of occurrences allowed for a specified record. After these many occurrences, the parser will not attempt to match the record and an exception is thrown.
Parsing Type	Specifies whether the data input is to be considered as Positional or Delimited.
Record Identifier Type	Type of the Identifier to be used for identifying a record. You can choose the Record Identifier from the following values: • **Field Value** Choose this option if you want to identify the record based on the value of some child field. In this case you need to select the field value in the Record Identifier Value property. • **Child Count** The record is identified based on the number of child counts. While parsing, if the child count in the record data does not match the number of children defined in the file schema, then parsing error is thrown. • **None Record** data is parsed against the record definition irrespective of the fact that the data satisfies the complete record definition or not.
Name	The name of the field. The name of the node should be a valid XML name. You cannot provide an existing field the same name as an existing record. Sibling fields cannot have the same name.

Property	Value
Description	The description of the field.
Escape Character Type	You can choose the escape character type from the following values: Default Escape Character - Indicates that the escape character is the value of the Default Escape Character property which is defined for the schema instance. **Character -** Allows the user to designate an escape character value. If you select **Character**, you must also specify an escape value. An escape character is useful if you have a character in your field data that is also used as the delimiter character in the field's parent record. For example, if your field data is the following and you have chosen a comma as the delimiter value of the record that contains the field, TSE interprets the comma after "Fiorano" to be a delimiter, even if you intend for it to be part of the field data: Fiorano,Software,USA Solution for this is to place an escape character directly preceding the delimiter character that you want to include in the field data. For example, if your escape character is specified as a backslash, you can place a backslash directly preceding a delimiter character, as in the following example: Fiorano\,Software,USA TSE interprets the comma after the backslash as field data rather than a delimiter character.
Delimiter Value	Type or select a value for the delimiter. To specify a delimiter value, you must first set the Delimiter Type to Custom Delimiter. The delimiter can be multi-character.
Delimiter Type	This is the field delimiter of this file schema. The delimiter can be multiple characters.
Escape Character	This is the escape character which is to be used as the field delimiter.

3.10.2.1.3 Field Node Properties

Depending on the type of file schema you are defining, you might need to add and/or remove fields. After adding fields to any schema, you must specify their properties. If you remove a field, its properties are also removed. You can't add records or fields under a field.

When you add a field, you can immediately rename the field. You can edit the name of an existing field and its properties by selecting the field and editing it.

1. If you click **Add > Field** from the pop up menu that appears after right clicking the mouse, the new field is inserted as a descendant of the record that you selected.
2. You cannot give an existing field the same name as an existing record.
3. You cannot provide a new field instance the same name as an existing sibling field or record.
4. Sibling fields cannot have the same name.

When a field is selected, the 'General' and 'Parsing' sets are enabled for a field. These properties define the format and structure of the field.

Any changes to the visible properties in the table are set for the currently selected node of the schema tree, which can be a record or field or the root node.

The following parameters are associated with the field node:

Property	Value
XML Type	The type for the record. This value can either be Element (default) or None. If None is selected, then the record is NOT mapped to the resultant XML.
Data Type	Represents the data type for the field data. This property can be set if you want to validate the field data against the supported data types. Data types supported by it include String (default), Integer, Numeric, Date, Byte, Data Format. This can be defined if the data type for the field is either Numeric or Date. For Numeric data type, data format can be defined based on the syntax rules of java.text.DecimalFormat. For Date data type, data format can be defined based on the syntax rules of java.text.SimpleDateFormat.
Minimum Length	The minimum number of characters that the field can contain.
Maximum Length	The maximum number of characters that the field can contain.
Default value	The default value for a field. The field is matched only if it's value is the same as the default value. Can be used to set Headers and Column Names.

Property	Value
Map If Null	Whether or not the field should be defined in the output XML if the value for the field in the source file is null/blank. • This property is redundant if the **XML Type** for the field is set to **None** in the **General properties set**. • If the value for the field in the source data is null/blank but **Default Value** is defined for the field, then the default value is set in the output XML. • This property field displays only if the structure of the parent record is delimited.
Wrap Character	Character used to enclose field data. This property is useful if you have a character in your field data that is also used as the delimiter value for the field's parent node. For example, if your field data is the following and you have chosen a comma as the delimiter value of the node that contains the field, TSE Parser interprets the comma after "Fiorano" to be a delimiter, even if you intend to include it as a part of the field data: Fiorano,Software,USA A solution for this is to define a value for the wrap character property and then enclose the field data in the wrap character. For example, you can set the wrap character property to double quotation marks for the first field and then type your field data, as in the following example: Fiorano, Software, USA The comma between the double quotation marks is interpreted by TSE Parser to be field data rather than a delimiter value. • This property field displays only if the structure of the parent record is delimited. • If you have a field that uses a wrap character, there cannot be any data between the wrap character and any delimiter leading or following a wrap character. • If your field data includes characters that are also used as the wrap character, you must enclose those characters in another set of wrap characters.
Padding character	This functionality is for the File Writer. If a certain field is smaller than the required size (either minimum length for delimited records or field length for positional records) then the FileWriter will pad the field with the padding character. Fields are always padded to the right of the field.

Property	Value
Valid Characters	The value for this property represents the set of valid characters for the field value. If this value is set and the field data contains any character which doesnot belong to this list, then parsing error is thrown.
Invalid Characters	The value for this property represents the set of invalid characters for the field value. If this value is set and the field data contains any character which belongs to this list, then parsing error is thrown.
Trim Spaces	Whether to trim the spaces from the source field data before setting in the output XML. You can opt for trimming the spaces from the following positions: • Both (Leading and Trailing) • Leading Trailing • None
Name	The name of the record. The name of the node should be a valid XML name.
Description	The description of the record.

3.10.2.2 Using the Text Schema Editor

The Text Schema Editor is an easy to use tool for defining various schemas. Typically, you can use this tool to create, import and test a schema. In addition, you can modify an existing schema by moving, copying and pasting elements in the schema file. The tasks that you can perform are:

1. Create and save a file schema
2. Open and modify an existing schema
3. Test a schema

3.10.2.2.1 Creating a desired File Schema

The following steps may be used to create a File Schema that you intend to use.

1. From the main toolbar, click on **File** > **New File.** The following pop-up is displayed:

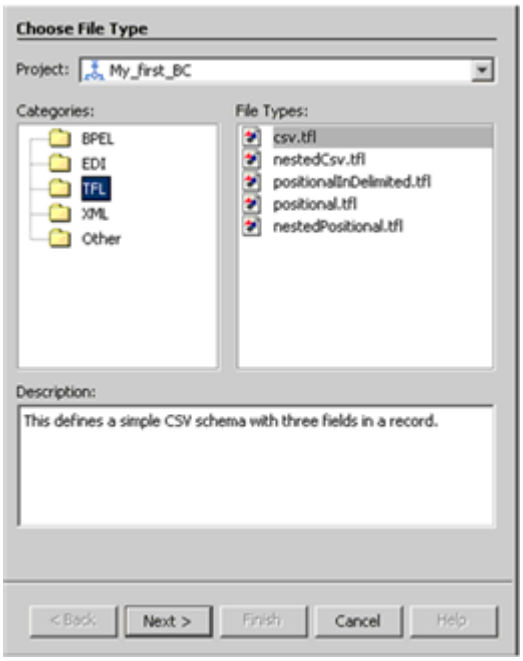


Figure 3.10.6: Creating a File Schema: New File

2. Click on TSE in the Categories easel.
3. Click on the type of schema you intend to use in the File Types easel. The templates of the following types of File Schemas are available:
   - Delimited File Schemas
   - CSV File Schema
     a. Nested CSV File Schema
     b. Positional in Delimited File Schema
   - Positional File Schemas
     a. Positional File Schema
     b. Nested Positional File Schema

   The desired Text Schema can be specified on the **New File** pop-up before you click **Next**.
4. Click on **Next**. The Name and Location dialog box appears:

Figure 3.10.7: Creating a New File Schema: Text Schema Editor - New CSV File

5. The File Name field may be populated with the name you intend to use for the scema file (tfl).
6. The path where you intend to store this file should be supplied in the Folder field.
7. Click **Finish** to save you configuration settings and create a new **tfl** schema file. This **.tfl** file can be invoked by the transformation components to transform non-XML data into its corressponding XML and vice versa.

8. Enter the relevant details of the schema in the appropriate fields and click the **Finish** button. The schema file is saved in the specified location with an extension .tfl and the corressponding XML is visible in the Source easel when the source button is enabled as shown in the following snapshot:

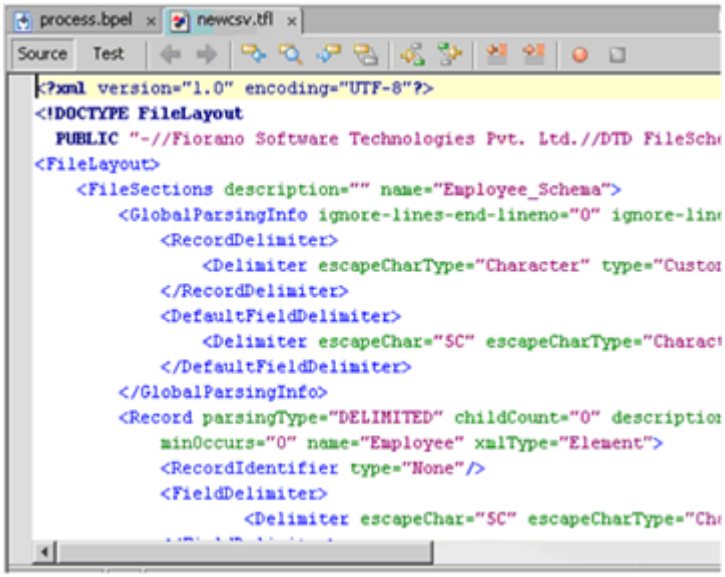


Figure 3.10.8: A newly created TFL File displayed in the Display easel

9. Finally, you can test this recently created schema by clicking the **Test** button as shown in the following snapshot:

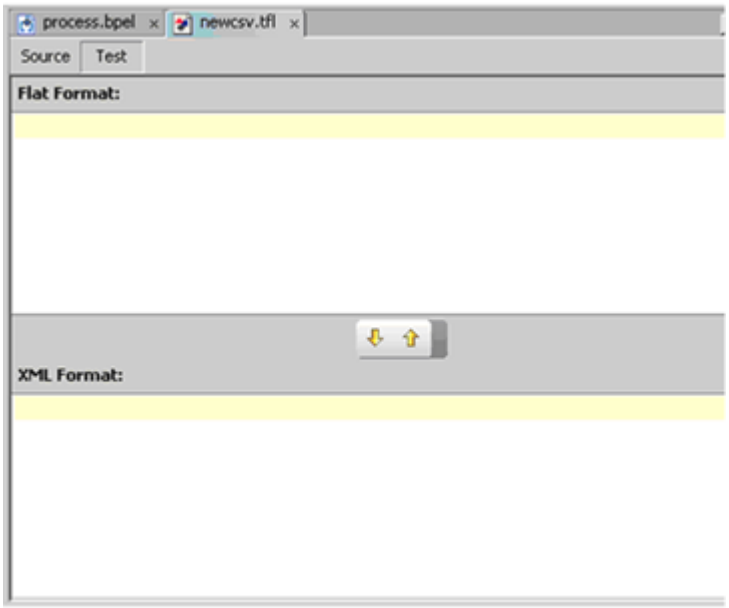


Figure 3.10.9: Testing a Schema

10. Click on the **Generate Sample** button to generate a sample script.
11. The up arrow key can be used to test XMl to Text format transformation while the down arrow key may be used to test Text to XML transformation.

3.10.2.2.2 Modify an Existing Schema

An existing schema can be modified by adding, deleting or moving records and fields. In addition, when working with multiple files, you can copy an entire record from one file to another.

Adding Records and Fields

Consider a scenario where you need to add a new record named Invoice_No to a schema file named Invoice.tfl.

1. To open the file, click **File > Open File**. The **Open** dialog box is displayed as shown in the Figure 3.10.10.

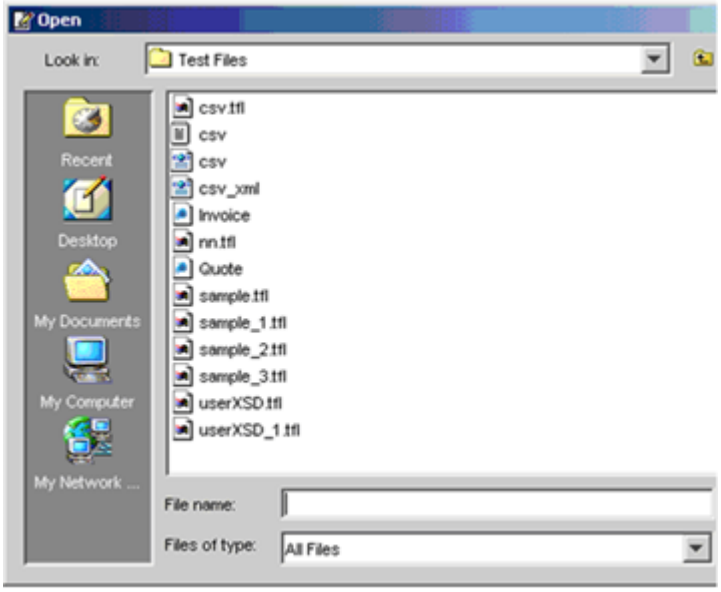


Figure 3.10.10: Open dialog box

2. Using the Look in drop down list, navigate to the folder that contains the file.
3. Type the name of the .tfl file in the File name field and click the Open button. The schema is displayed.

4. To add a record, right-click the record node and select the **Add > Records** option from the shortcut menu as shown in the Figure 3.10.11.

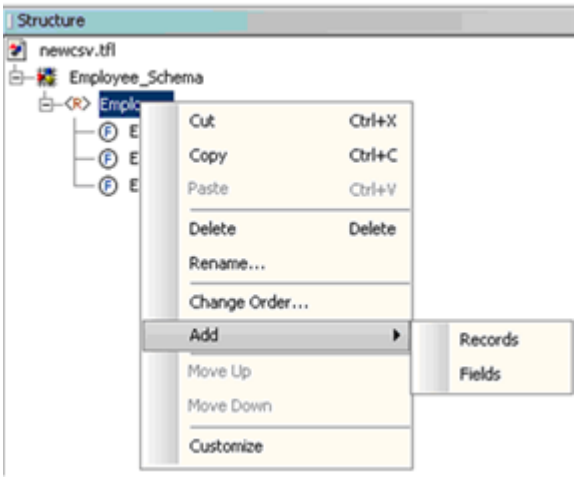


Figure 3.10.11: Add a New Record shortcut menu

5. The **Input** dialog box is displayed as shown in the Figure 3.10.12 after you click the Fields option in the earlier step.

Figure 3.10.12: Input dialog box

6. Type a desired name in the Add field and click the **OK** button. A new record is added to your existing record.Simlarly a field may be added to an exisiting record. You can move a record or field up or down hierarchially in the structure easel by using the Move Up and Move Down options in the right click menu on the Structure Easel.

Deleting Records and Fields

If you intend to delete a record or a field from your existing File Schema, you may do so by using the following steps:

1. Open the *tfl* file of the existing File Schema.
2. In the Structure easel, right-click on the record or field you intend to delete.
3. From the right-click menu choose the **Delete** option.

3.10.2.3 Warnings

A warning, which is normally associated with positional records, is the Out of Sequence Error. This occurs if the start position of a field is not equal to the end position of the previous field + 1. This is only a warning and parsing on such a record can be carried out successfully.

3.10.2.4 Limitations

As of now the tool is logically stand-alone and so you have to create a file schema, save it and then import it in any of the components. The following are some limitations of TSE.

1. All escape characters can be used to escape only the delimiter of the layout/record that they belong to. For example, if the record delimiter for the whole file is "\r\n" (CR-LF) and the corresponding escape character is "\", while the field delimiter for a record in the file is "," and the corresponding escape character is ":", then, in order to escape "\r\n", only "\" can be used.
2. In order to escape reserved characters other than the delimiters, the wrap character functionality has to be used. (A field has to be enclosed within wrap characters).

3.11 Public Key, Cryptography Keystore, And Truststore

This section describes the concepts as they are implemented in Fiorano SOA Platform.

3.11.1 Using Public Key Cryptography for Authentication

Authentication is the process of verifying the identity of an entity to ensure that one entity verifies the identity of another entity. In the following example, user A and user B uses public key cryptography to verify user B's identity. The following notation indicates that an item has been encrypted or decrypted using key cryptography.

{something}key

where something is a description of the item that has been encrypted or decrypted, and key is the key that is used to encrypt or decrypt that item.

In the following example, user A wants to authenticate user B. User B has a pair of keys, one public and one private. User B discloses the public key to user A. User A generates a random message and sends it to User B as follows:

A->B random_message

User B uses the private key to encrypt the random message and returns the encrypted version to user A:

B->A {random_message}User_B's_private_key

User A receives this message and decrypts it by using the public key that B previously published. User A compares the decrypted message with the message that user A originally sent to user B; if the messages match, user A knows that the later message came from user B, because an imposter presumably would not know user B's private key and so would not be able to properly encrypt the random message to send to user A.

Web server certificates are used to authenticate the identity of a web server to the clients accessing that web server. When a client wants to send confidential information to a web server, the client accesses the server’s digital certificate. The certificate, which contains the web server’s public key is used by the client to authenticate the identity of the web server and encrypt the information sent to the server using the Secure Socket Layer (SSL) technology. Since the web server is the only entity with access to its public key, only the server can decrypt the information. This is how the information remains secure during transit across the Internet.

3.11.2 Keystore and Truststore

Secure Sockets Layer (SSL) is a protocol designed to enable secure communications on an insecure network such as the Internet. SSL provides encryption and integrity of communications along with strong authentication using digital certificates. SSL allows a secure connection between a client and a web server.

SSL uses public and private keys to encrypt and decrypt information.

Public key encryption is a technique that uses a pair of asymmetric keys for encryption and decryption. Each pair of keys consists of a public key and a private key. The public key is made public when it is distributed widely. The private key is never distributed; it is always kept secret.

Data that is encrypted with the public key can only be decrypted with the private key. Conversely, data that is encrypted with the private key can be decrypted only with the public key. This asymmetry is the property that makes public key cryptography so useful.

As diagramatically illustrated below, the public key is shared between the client and the server. The private key is however kept private by the client as well as the server.

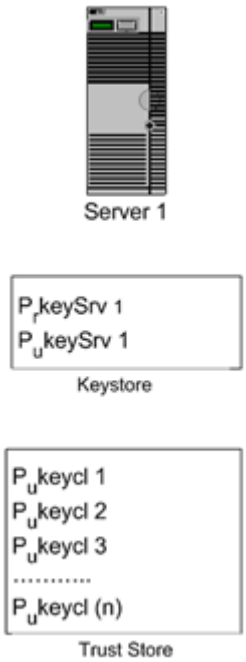


Figure 3.11.13: Public key shared between the client and the server

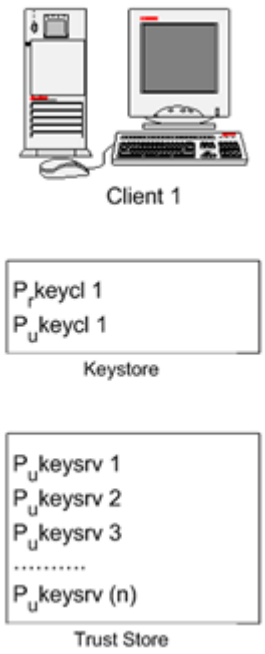


Figure 3.11.14: How Public and Private keys are used

As illustrated in the diagram above, the exchange of message between the client and the server is ensured in the following way.

- The client has a keystore where it stores its public key/private key pair. Likewise the server also has its keystore containing its public key/private key pair.
- The client publishes its public key. The server stores the public key of the client in its trust store.
- Likewise the server publishes its public key. The client stores it in its trust store. Truststore is a file where digital certificates of trusted sites are stored and retrieved for authentication during an SSL connection.

When the client wants to send confidential information to the web server, the client accesses the server's digital certificate. The certificate, which contains the web server's public key is used by the client to authenticate the identity of the web server and encrypt the information sent to the server using the Secure Socket Layer (SSL) technology. Since the web server is the only entity with access to its public key, only the server can decrypt the information. This is how the information remains secure during transit across the Internet.

The steps to be followed in using SSL over the internet are as follows:

- Generating a client keystore
- Getting the Digital Certificate of Server
- Creating the Client Truststore
- Using the Keystore and Truststore in an SSL Application

3.11.2.1 Generating a Client Keystore

A keystore is a file that holds the public and private key pairs and certificates for SSL specifications. The Keystore is a database of public and private keys. Java keytool is used to generate the public/private key pairs. Java keytool utility is present at %JAVA_HOME%/bin.

To generate a keystore, open the command prompt and type in the following command line and press the Enter key:

<Your directory>%JAVA_HOME%\bin\keytool -genkey -alias [alias name] -keystore [keystoreName] -keyalg [algorithm] -validity [days in integer] -storepass [store password] -keypass [key password]

Here is a brief description of the options used in the keytool command:

Parameters	Description
-genkey	Requests keytool to generate a key pair
-alias	Identifies the new key pair within the keystore
-storetype	Declares the type of the keystore. JKS is the default type
-keyalg	Declares the algorithm to be used; we're using the RSA public key algorithm, which is the default
-storepass	Specifies the password for the entire keystore
-keypass	Specifies the password for the new key pair
-Validity	It is the validity of the key pair in days
-keystore	File that holds the public and private key pairs

For example, you want to generate the keystore in the directory d:\WorkStudio\keystore, and then the command would look something like:

D:\WorkStudio\keystore>%JAVA_HOME%\bin\keytool -genkey -alias client1 -keystore client1.keystore -keyalg RSA -validity 365 -storepass cl1storepass -keypass cl1keypass

Press the **Enter** key. The result of the command is as shown in the Figure 3.11.15.

```
C:\WINNT\System32\cmd.exe

D:\WorkStudio\keystore>%JAVA_HOME%\bin\k
yalg DSA -validity 365 -storepass cl1sto
What is your first and last name?
  [Unknown]:  Ayrton
What is the name of your organizational
  [Unknown]:  Programming
What is the name of your organization?
  [Unknown]:  Fiorano
What is the name of your City or Localit
  [Unknown]:  10th Avenue Street
What is the name of your State or Provin
  [Unknown]:  California
What is the two-letter country code for
  [Unknown]:  US
Is <CN=Ayrton, OU=Programming, O=Fiorano
  [no]:  y

D:\WorkStudio\keystore>_
```

Figure 3.11.15: Running the keytool command

Note: You will have to type in the answers to the questions that appear as shown above.

The keystore file is generated in the specified directory. The next step is to create a truststore and add the server certificate in it.

3.11.2.2 Getting the Digital Certificate of Server

Digital certificate contains the public key and are stored in a Truststore. The Truststore is a file where certificates of trusted sites can be retreived for authentication during an SSL connection.

To generate a truststore, you will have to first export and save the public key of the server you are going to access using SSL. To do this, perform the following steps:

1. Type in the address of the secure website on the address bar of your internet browser. As an example, you may type in https://adwords.google.com. The following dialog is displayed.

Figure 3.11.16: Security Alert dialog

2. Click the **View Certificate** button. The **Certificate** dialog is displayed.

Figure 3.11.17: Certificate dialog

3. Click the **Details** tab and highlight the public key as shown in the Figure 3.11.18.

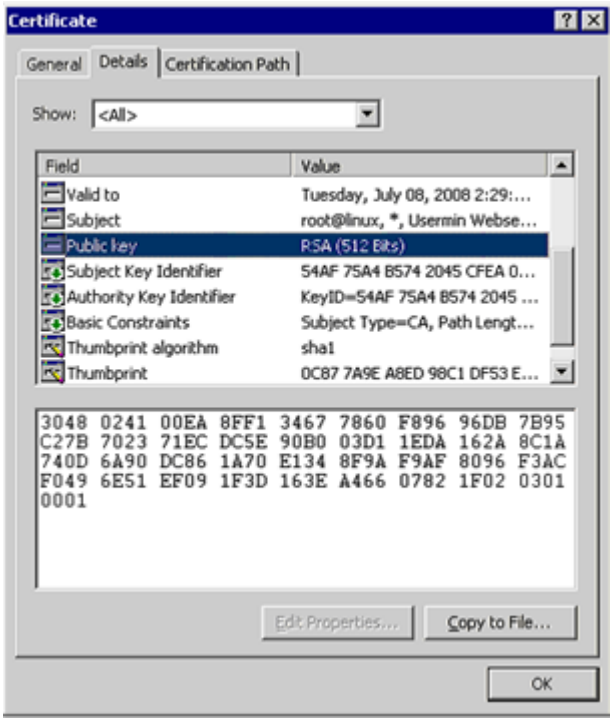


Figure 3.11.18: Public Key

4. Next, click the **Copy to File** button and save the certificate file in the directory where the keystore has been generated as illustrated in the Figure 3.11.19.

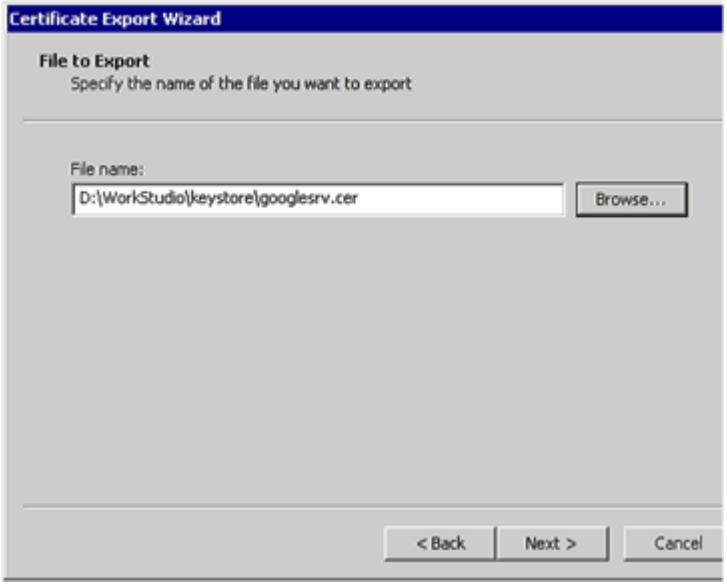


Figure 3.11.19: Certificate Export wizard

Note: The process of copying and saving the digital certificate may differ from browser to browser. The concept is however the same. In the guide we have illustrated the process on Microsoft's Internet Explorer.

5. Once you have saved the digital certificate, you are now ready to create the truststore.

3.11.2.3 Creating the Client Truststore

Perform the following procedure to create a trust store:

1. Open the command prompt and type in the following command and press Enter:

   Your directory>%JAVA_HOME%\bin\keytool -import -alias [alias name] -file [file name.cer] -keystore [truststorename.keystore] -storepass [storepassname]

   If you want to generate the truststore in the directory D:\WorkStudio\keystore, then the command would look something like:

   D:\WorkStudio\keystore>%JAVA_HOME%\bin\keytool -import -alias mailserver -file googlesrv.cer -keystore truststore.keystore -storepass trustpass

2. Next press the Enter key. The result of the command is as shown in the Figure 3.11.20.

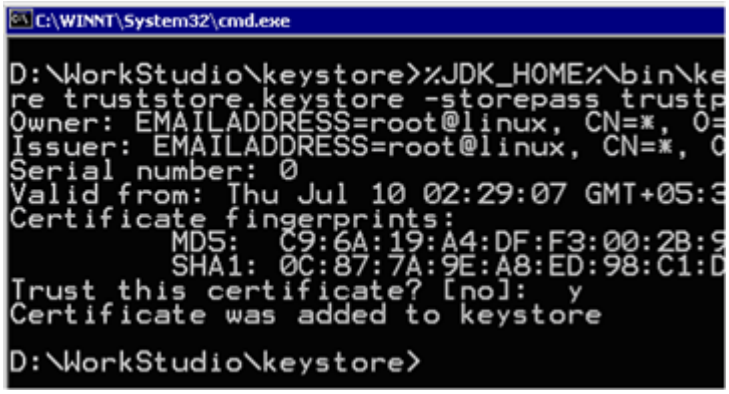


Figure 3.11.20: Command Prompt

Once the keystore and the truststore have been created, you are now ready to use them in the configuration of SSL in the webcomponents.

3.11.2.4 Using the Keystore and Truststore in an SSL Application

Now we will use the HTTPAdapter to access a secure HTTP connection. We use the keystore and the trustore that we created in the previous sections of this chapter.

The section demostrates how HTTPAdapters component can be configured for **HTTP Get** operation to access a secured HTTP connection:

- Start the **Studio**. Next, drag and drop the **HTTPAdapters** component from the **Service Pallete** as shown in Figure 3.11.21.

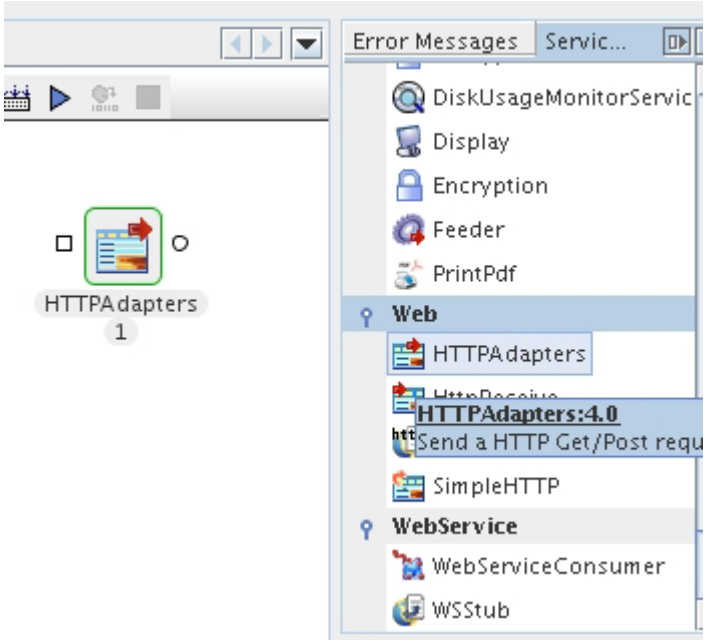


Figure 3.11.21: Service Palette

- Next, double-click on the **HTTPAdaptes** component. The HTTPAdapters custom properties wizard is displayed.
- Enter the value for the host field, for example, https:// adwords.google.com. Enter the port number. Https hosts by default use a port number of 443. If the host uses a different port number then it has to be provided.
- Provide SSL Security configurations in the CPS. By default the property **SSL Security** is not visible. Click on the 'expert properties' icon to make the property visible.

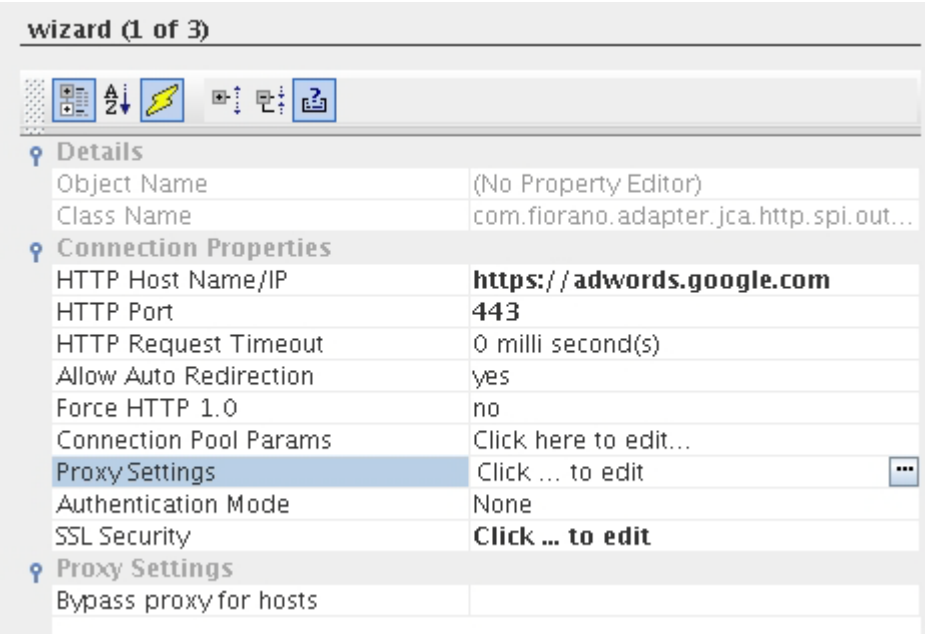


Figure 3.11.22: HTTPGet Custom Properties

- Open SSL Security by clicking on the ellipsis and select the **Enable SSL** check-box. The SSL related fields gets enabled as shown in the Figure 3.11.23. Provide the details for the following fields.

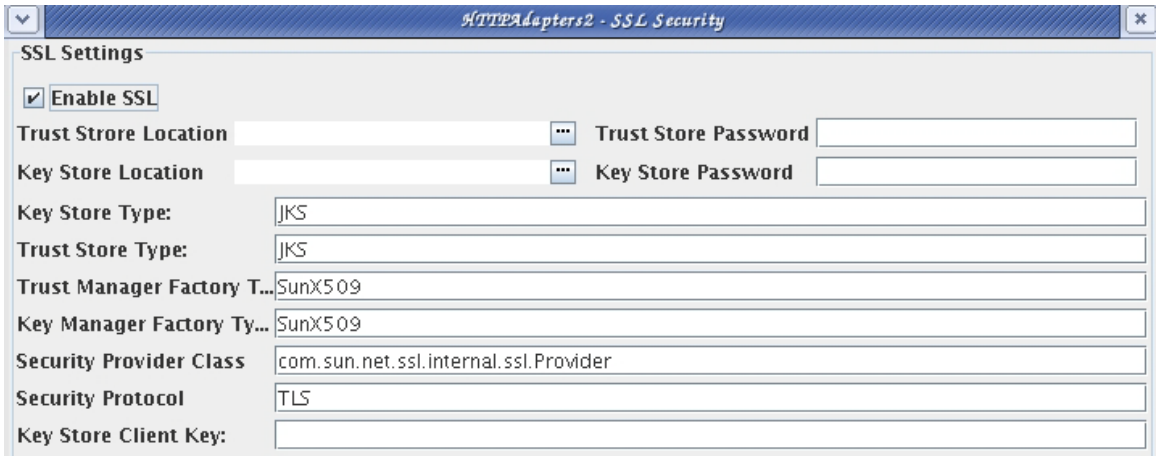


Figure 3.11.23: SSL Security parameters

- **Truststore Location** - Location and filename of the truststore file. For providing the Truststore, select the truststore.keystore file in \workstudio\keystore that we created in the previous section.
- **Truststore Password** - Password of the specified truststore. Specify trustpass as the truststore password.
- **Keystore Location** - Location and filename of the keystore. Select the client1.keystore file in \workstudio\keystore directory.
- **Keystore password** - Password for the keystore. Specify cl1storepass as the store password.
- **Keystore Client Key** - Password for the client key. Specify cl1keypass as the Keystore Client key.

Note: The values we are providing for keystore and truststore are the values created in the previous sections. These are not the default values.

- Accept the default values for the fields whose values are already provided.
- Next, activate the **HTTPRequestProperty** parameters panel in the Configuration Property Sheet. The **HTTPRequestProperty** parameters panel is as shown in the Figure 3.11.24.

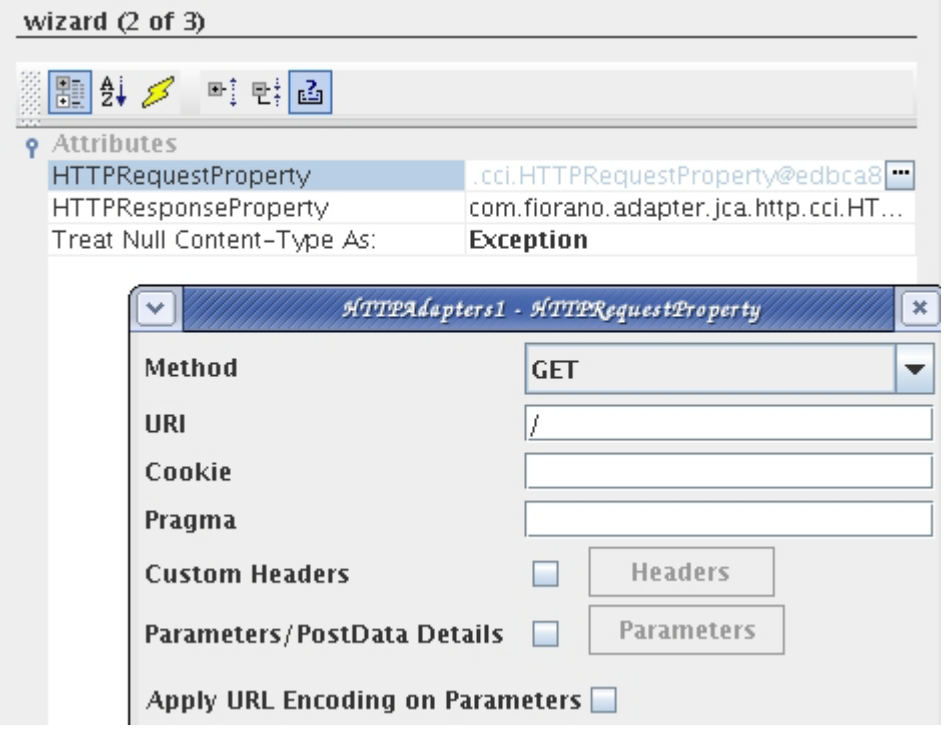


Figure 3.11.24: HTTPGet Custom Properties

- Enter a **/** (slash) for the URI field.
- Finish the configuration by navigating through the remaining panels and accepting the default values for each panel.
- Connect a feeder and display component with the HTTPAdapters component using routes as shown in the Figure 3.11.25.

Figure 3.11.25: P2P Connection

- Next, double-click the feeder component to display its **Configuration Property Sheet** (CPS). The **CPS** is displayed. Now, click the **Fetch from Connected Service** icon. The **Output DTD/XSD** is displayed as shown in the Figure 3.11.26.

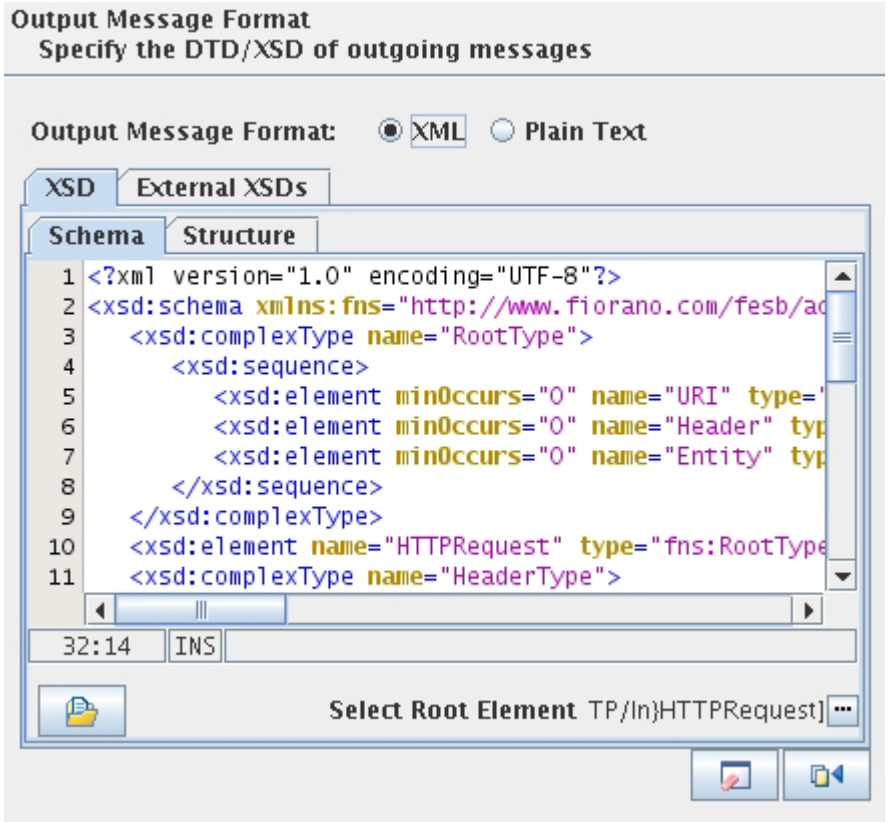


Figure 3.11.26: Feeder Custom Properties

- Click the **Next** button to navigate to the **Options** panel. Click the **Generate Sample XML** button. The **Generate Sample XML** dialog is displayed as shown in the Figure 3.11.27.

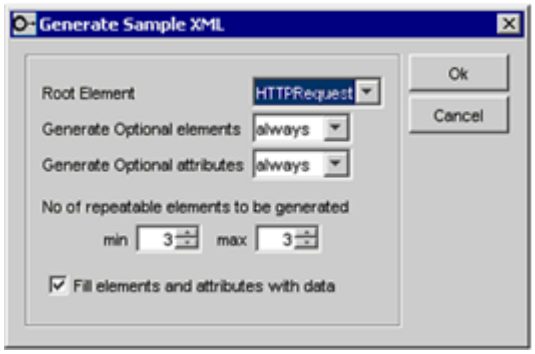


Figure 3.11.27: Generating Sample XML

- Accept the default values and click the **OK** button. From the sample XML that is generated, delete all the XML tags and leave only the <HTTPRequest> opening and closing tags as shown in the Figure 3.11.28.

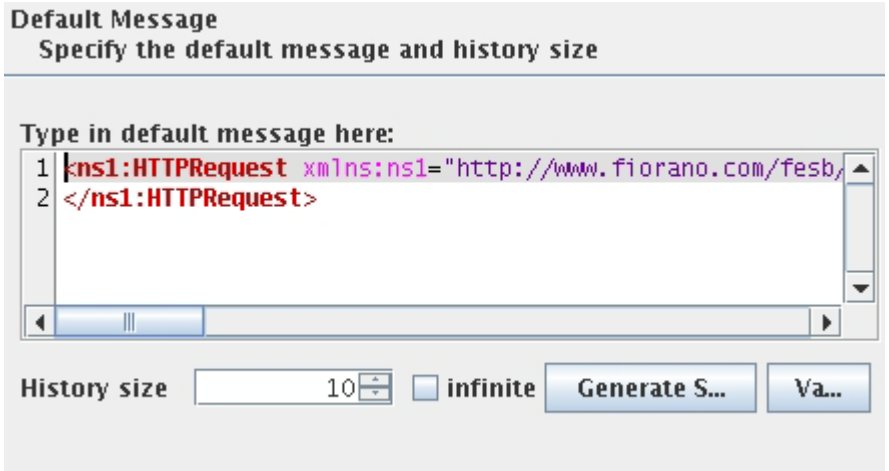


Figure 3.11.27: Feeder Custom Properties

- Click the **Finish** button to finish the configuration of the feeder.
- Next, do the **Component Resource Check** and run the event process. Wait until the components turn green in the **Event Process** easel. When the event process successfully runs, the feeder window is displayed as shown in the Figure 3.11.29.

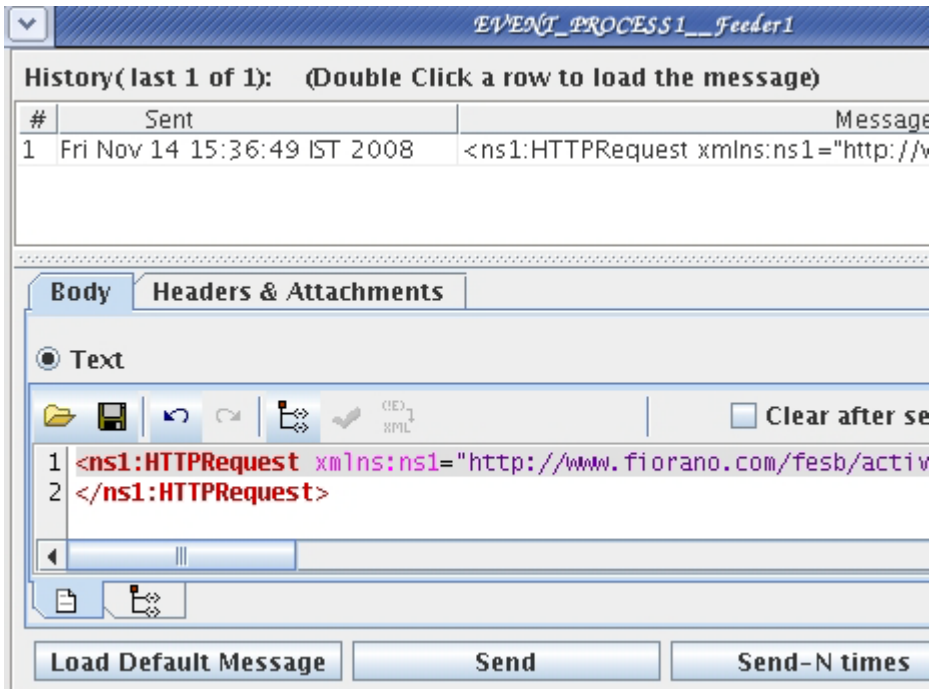


Figure 3.11.29: Feeder Business Service

- Click the **Send** button to send the HTTP request from the feeder to the HTTPAdapters component. The result of the request is displayed in the display business component as shown in the Figure 3.11.30.

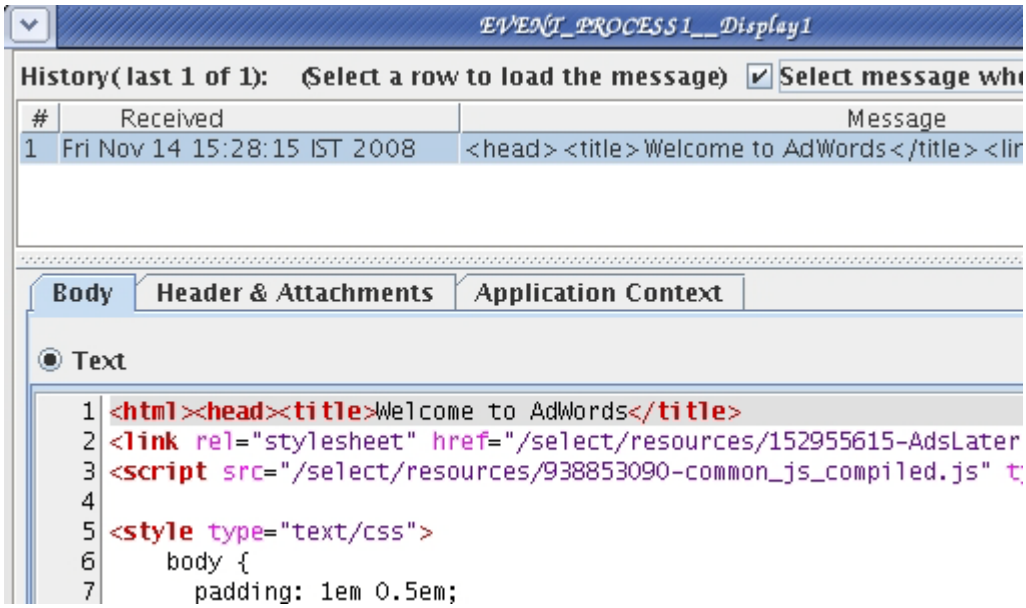


Figure 3.11.30: Display component displaying Result of HTTPAdapters Request

3.11.3 Generating self-signed X509 Certificates

If the certificate need not be signed by a Certificate Authority (CA), for example when used for just testing purposes, a certificate can be self-signed. This can be done in several ways.

1. Generate a Certificate Signing Request (CSR) and sign the request using openssl.
2. Generate self-signed certificate without generating CSR using openssl.
3. Generate a CSR using keytool and sign the request using openssl.

3.11.3.1 Generate a Certificate Signing Request (CSR) and sign the request using openssl

Generate a private key using openssl

openssl genrsa -des3 -out cakey.pem 2048

Generate a CSR

openssl req -new -keyout cakey.pem -out careq.pem

Sign the CSR

openssl x509 -signkey cakey.pem -req -days 365 -in careq.pem -out cacert.pem -extfile /etc/ssl/openssl.cnf -extensions v3_ca

cacert.pem can be used as a self-siged certificate.

3.11.3.2 Generate self-signed certificate without generating CSR using openssl

1. Generate a private key using openssl as explained earlier.
2. openssl genrsa -des3 -out cakey.pem 2048, and then generate a self-signed certificate
3. openssl req -new -x509 -key cakey.pem -out cacert.pem -days 1095
4. cacert.pem can be used as a self-siged certificate.

3.11.3.3 Generate a CSR using keytool and sign the request using openssl

1. Use keytool to generate a certificate request
2. Generate a client keystore as mentioned in Section 3.11.2.1 above. Use this keystore to generate a certificate request to be sent for signing by a recognised CA.
3. keytool -certreq -alias fiorano -keystore fiorano.ks -file fiorano.csr
4. This certificate request can also be self-signed instead of getting it signed by a CA.
5. Generate self-signed X509 certificate
6. openssl req -new -keyout cakey.pem -out careq.pem
7. openssl x509 -signkey cakey.pem -req -days 365 -in careq.pem -out cacert.pem -extfile /etc/ssl/openssl.cnf -extensions v3_ca

8. openssl x509 -CA cacert.pem -CAkey cakey.pem -CAserial serial.txt -req -in fiorano.csr -out fiorano.cer -days 365
9. Import self-signed X509 certificate into keystore
10. keytool -import -alias testCA -file cacert.pem -keystore fiorano.ks
11. keytool -import -alias fiorano -file fiorano.cer -keystore fiorano.ks

For more information on openssl commands, please refer to http://www.openssl.org/docs/

3.12 Component Control Protocol (CCP)

The Component Control Protocol is a protocol designed for exchange of control and state information over bidirectional channel between Peer Server and components which run on the Peer Server.

3.12.1 Communication Channels

Communication between the Peer Server and components happen over two JMS topics:

- CCP_COMPONENT_TO_PEER_TRANSPORT – Topic on which components send data, status and acknowledgement events to the Peer Server
- CCP_PEER_TO_COMPONENT_TRANSPORT – Topic on which the Peer Server sends control commands and data requests to components.

Message selector which is a function of the GUID of application to which component belongs and the instance name of component is used to identify requests sent to and from components on the control topics.

3.12.2 Events

Javadocs for different types of events that are exchanged between component and the Peer Server can be found at %FIORANO_HOME%/javadoc/esb/ccp.

As of this release, only INITIATE_SHUTDOWN Command event and Status events are supported.

3.12.3 Component Lifecycle and CCP

A component's life cycle typically consists of following stages:

- Component startup – This stage starts as soon as the component process is created and lasts until component is ready to process requests. The component looks up all required configurations and initializes all required objects
- Transport startup – This stage is a part of component startup where all necessary transport objects like JMS connection, sessions, consumers, producers, message listeners and other related objects are created
- Component runtime – This stage starts as soon as all necessary initialization is done and component can accept requests and lasts until a shutdown is initiated.

- Transport shutdown – This stage is a part of component shutdown. After this stage all processing is stopped and only cleanup of any other objects remain.
- Component shutdown – This stage starts with a shutdown request and lasts until the component terminates. The component cleans up all objects created in an orderly fashion during this stage.

Without CCP, when a component is launched in separate process it has no way to control the how its state is reflected as it cannot provide feedback to the PeerServer on its state. This causes following problems:

A component is considered to be started as soon as a JMS connection used for transport is created and its state is reflected as such in tools.

When a component is stopped there is no way to communicate to component to initiate an orderly cleanup and shutdown. The component process is forcefully destroyed instead.

CCP transfers the control over component states to the component. The Peer Server only sends necessary commands to the component and waits for component to provide feedback on its state.

For startup, the Peer Server wait for COMPONENT_STARTED status event to indicate that component is started instead of relying on JMS connection creation.

For shutdown, a component which has CCP feature enabled would initiate orderly shutdown of the component as soon as a INITIATE_SHUTDOWN command is received from the Peer Server and terminates itself once the cleanup is done. A component can send intermediate status events to indicate the progress of shutdown process which are logged by the server. Peer Server waits for component process to terminate for a configured interval of time. If the component process is not terminated within the configured time, the Peer Server will forcefully terminate the component process. This can be configured in FES profile against ComponentStopWaitTime property on ApplicationController node at path FES → Fiorano → Esb → Application → Controller → ApplicationController

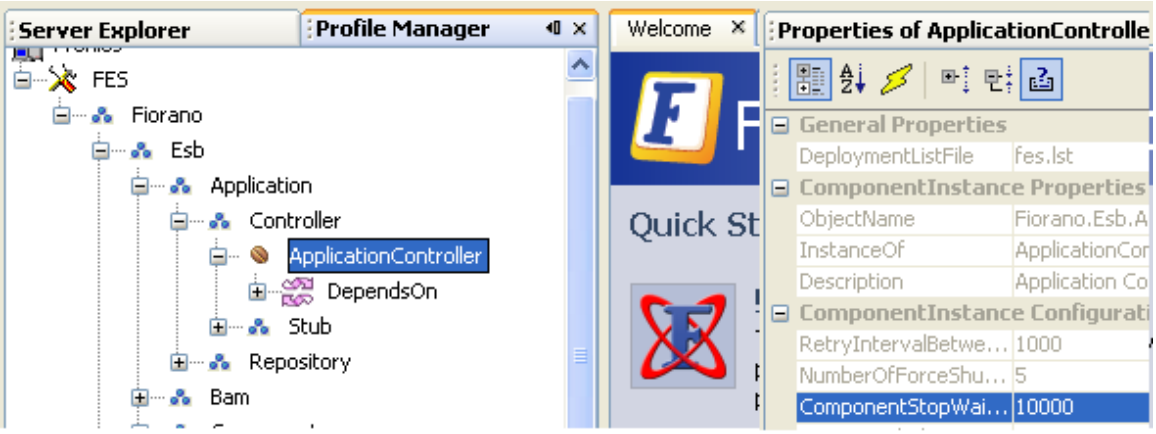


Figure 3.12.1: ComponentStopWaitTime property in FES profile

3.12.4 Sample component

A sample component which implements CCP can be found at %FIORANO_HOME%/esb/tools/templates/samples/CCPComponent.

Chapter 4: Pre-built Components

This chapter lists all the component categories and the pre-built components that are available in the Fiorano SOA Platform. This section also describes how to develop components in various languages which can be deployed into.

Fiorano SOA Platform provides with a wide range of pre-built components, each performing a simple business task. These components are the building blocks for designing various automated business processes each meeting a definite business need. The components are loosely coupled to each other and interact through asynchronous invocations using messages over JMS.

List of Pre-built components

The pre-built components which are available in Fiorano SOA Platform as listed below:

Category	**Components**
Bridges	EJBAdapter, FTPGet, FTPPut, IWay, POP3, SAPR3, SMSBridge, SMTP, SocketAdapter, SalesForce, SAPB1, SapR3Monitor, SAPStub, HL7Receiver, and HL7Receiver
Collaboration	chat, csChat, vbChat, vcChat
DB	DB, DBProc, DBQuery, DBQueryOnInput
Error	ExceptionListener
File	FileReader, FileWriter, FileTransmitter, and FileReceiver
Flow	Aggregator, CBR, Cache, DistributionService, Join, Sleep, Timer, WorkList, WorkListManager, XMLSplitter, XMLVerification, and Cache
MOMs	JMSIn 4.0 and 5.0, JMSOut 4.0 and 5.0, JMSReplier, JMsRequestor 4.0 and 5.0, MQSeriesIn, MQSeriesOut, MSMQReceiver, MSMQSender, TibcoRVIn, TibcoRVOut
Performance	Receiver, Sender
Samples	BinaryFileReader, CRM, CompositeBC, LDAPLookup, LDAPAuthenticator, MarketPricesGui, Prices, RfqManager, TradeBus, erp
Script	BeanShell, GroovyScript, JavaScript, PerlScript, Python Script
Transformation	EDI2XML, HL7Reader, HL7Writer, Text2XML, XML2EDI, XML2PDF, XML2Text, Xslt
Util	Compression, Decompression, Decryption, DiskUsageMonitorService, Display, Encryption, Feeder, and PrintPDF

Web	HTTP, HttpReceive, HttpStub, SimpleHTTP, RESTStub, and RESTConsumer
WebService	Stub, WebServiceConsumer 4.0 and 5.0
Finance	FixAcceptor and FixInitiator

4.1 Bridges

Bridges category consists of components like EJBAdapter, FTPGet, FTPPut, IWay, POP3, SAPR3, SMSBridge, SMTP, SapR3Monitor, HL7Sender, and HL7Receiver. The following section describes each component.

4.1.1 EJBAdapter

The EJB component can be used to access an Enterprise Java Bean hosted in any application server, like Weblogic, for the development and deployment of transactional, distributed object applications-based, server-side software components. The adapter uses EJB 1.1 semantics for accessing the EJB.

Points to note

2. The EJB client jar needs to be added as a resource to this adapter. Please refer to the application server (on which the EJB is deployed) documentation on how this can be done.
3. The Initial Context Factory class, specific to the application server being connected to, should also be added as a resource to this adapter. For example, weblogic.jar should be added as a resource if the EJB is deployed in Weblogic application server.
4. Only stateless session beans and entity beans are supported. Using this adapter with stateful session beans may not produce the desired results.
5. In case of JBOSS application server, log4j-1.2.9.jar present in '$FIORANO_HOME$/extlib/log4j/' has to be updated with the jar from JBOSS installation directory.

4.1.1.1 Configuration and Testing

4.1.1.1.1 Managed Connection Factory Panel

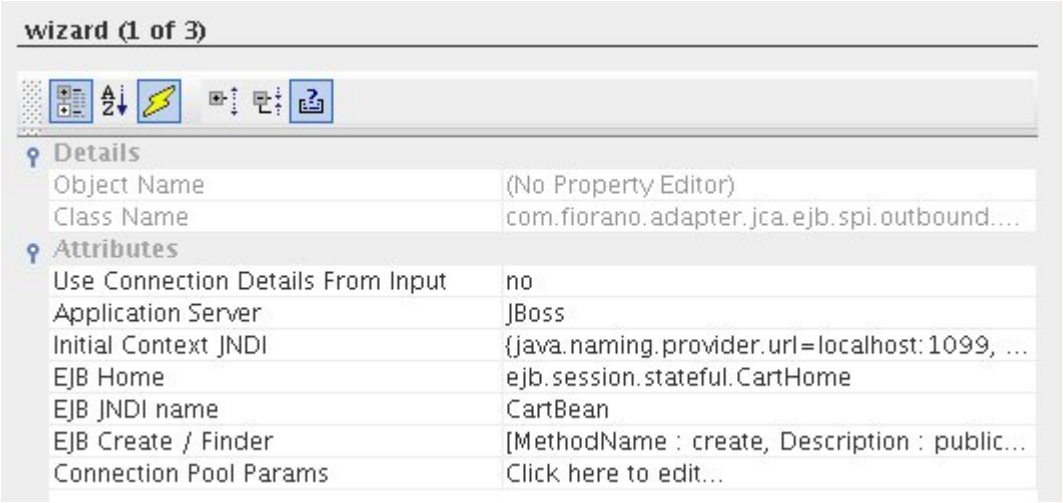


Figure 1: Managed Connection Factory Panel

Attributes

Use Connection Details From Input: Parameters to create the connection can be specified in the input message when this property is set to true. If this property is selected and the connection parameters are not given in the input request, then the values given in this panel for those parameters will be used.

Application Server: The Application Server on which the Enterprise Java Bean is deployed should be specified here.

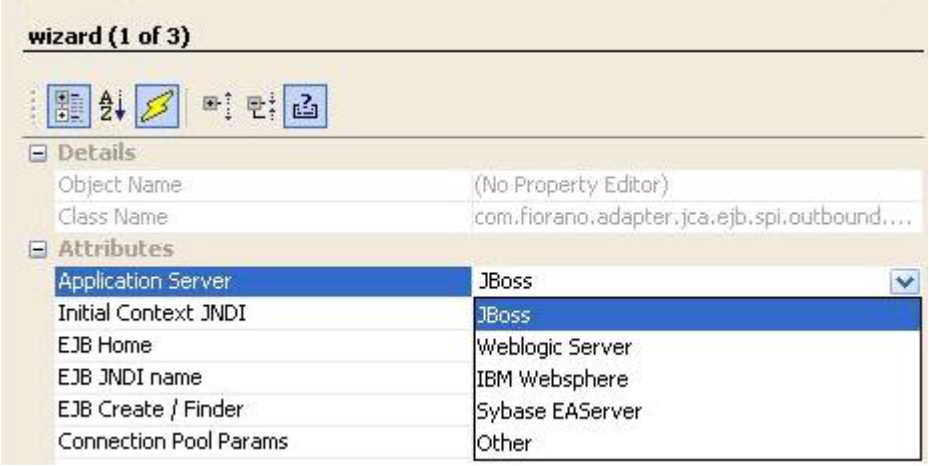


Figure 2: Choose an Application Server

Initial Context JNDI: Properties like InitialContextFactory, Provider URL etc. used to create Initial Context should be specified here by clicking on the elipsis and adding entries in the table as shown in figure 3.

For example, to connect to JBOSS server, the following properties should be added:

- **java.naming.factory.initial** – org.jnp.interfaces.NamingContextfactory
- **java.naming.provider.url** - <IP on which the server is running>:<Port number on which the server is running>

In addition to these, the user may provide other properties depending on the Application Server the component should connect to.

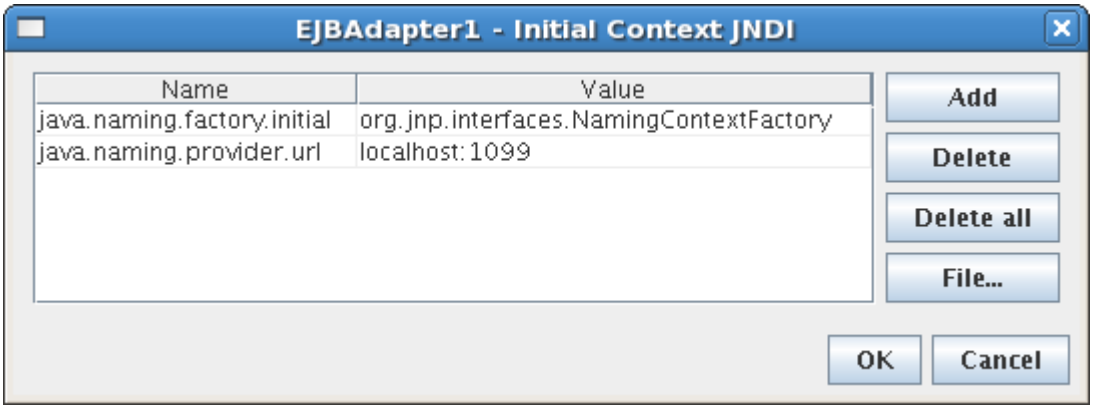


Figure 3: Initial Context JNDI

Note: Default InitialContextJNDI values vary depending on the **Application Server** selected.

- **EJB Home**: Fully qualified name of the class which implements the EJBHome interface.
- **EJB JNDI Name**: JNDI name with which the EJB is bound to the Application Server.
- **EJB Create / Finder**: After providing EJB Home and EJB JNDI Name properties, click on the ellipsis to select a method from the list of Create / Finder methods.

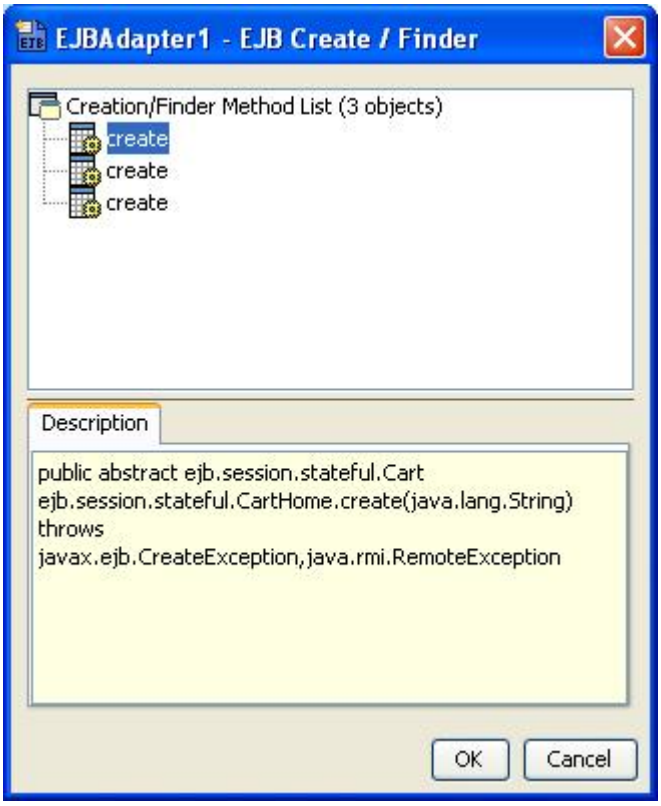


Figure 4: Select Create/Finder method

If the selected Create/Finder method has any parameters, double-click on the selected node to provide those parameters to the method.

Note: Only primitive types can be passed to the Create/Finder methods. Complex date types are not supported.

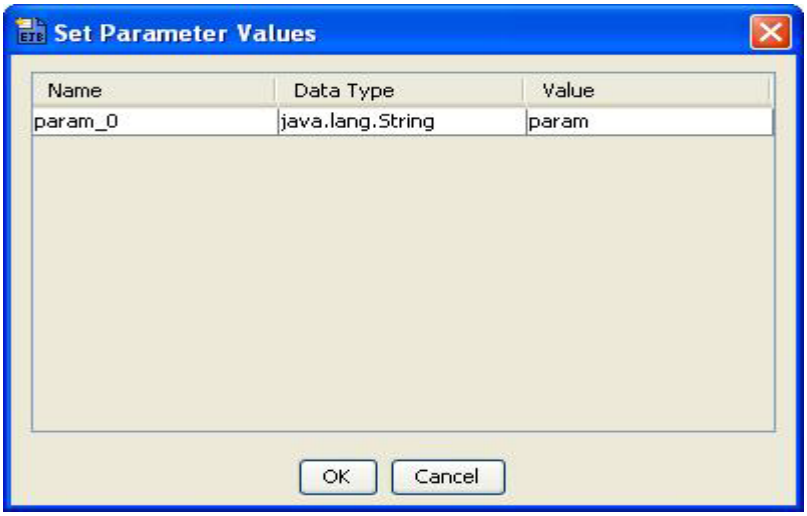


Figure 5: Pass parameters to the Create/Finder method

4.1.1.2 Interaction Configurations Panel

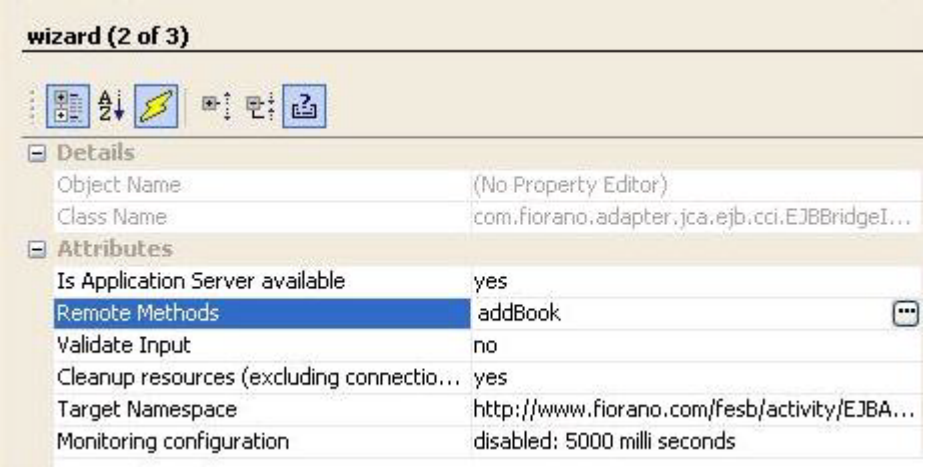


Figure 6: Interaction Configuration Panel

Attributes

Is Application Server Available: Specifies whether Application Server is available.

- **yes**: If a connection can be established with the Application Server at the time of CPS configuration. Remote Methods option is visible in this case.
- **no**: If a connection cannot be established with Application Server at the time of CPS configuration. Remote Methods option is not visible in this case. In this case, the details of the Remote Method to be executed should be provide manually, and so options Method Name, Number of Parameters, Parameter types, Return Type are visible.

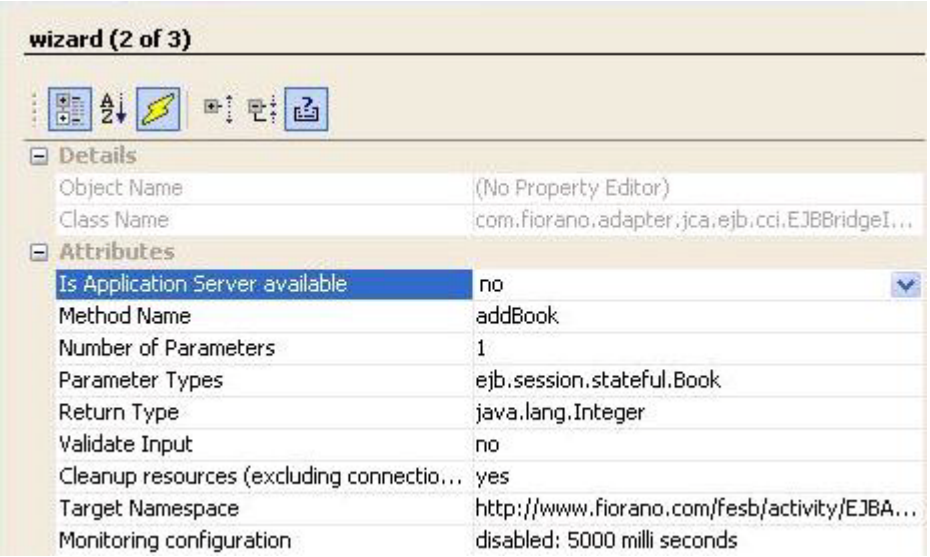


Figure 7: Interaction Configuration when Application Server is not available

Remote Methods: Click on the ellipsis to select the Remote method to be executed from the list of methods.

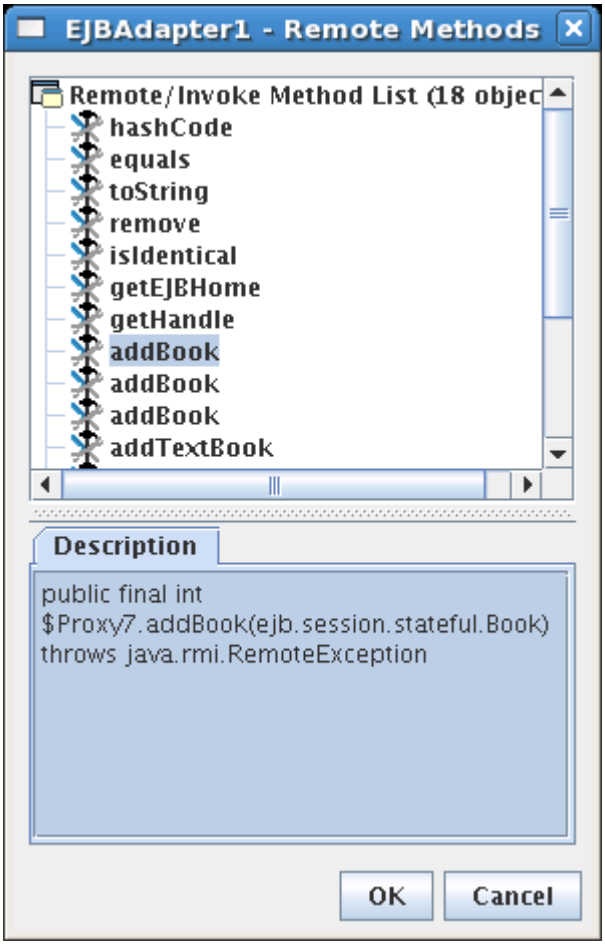


Figure 8: Select Remote Method

- **Method Name**: The name of the Remote Method of the EJB that is to be executed.
- **Number of Parameters**: The number of parameters that are to passed to the remote method.
- **Parameter Types**: Comma separated, fully qualified class names of the parameters that are to be passed to the remote method.
- **Return Type**: Fully qualified class name of the return type of the remote method.

Note: In case there are primitive types in Parameter Types and/or Return Type, corresponding wrapper classes should be used.

4.1.1.2.1 Using Named Configurations

Changes are done in eStudio w.r.t to Named Configurations. With the implementation of Named Configurations MCF panel is changed as shown in Figure below.

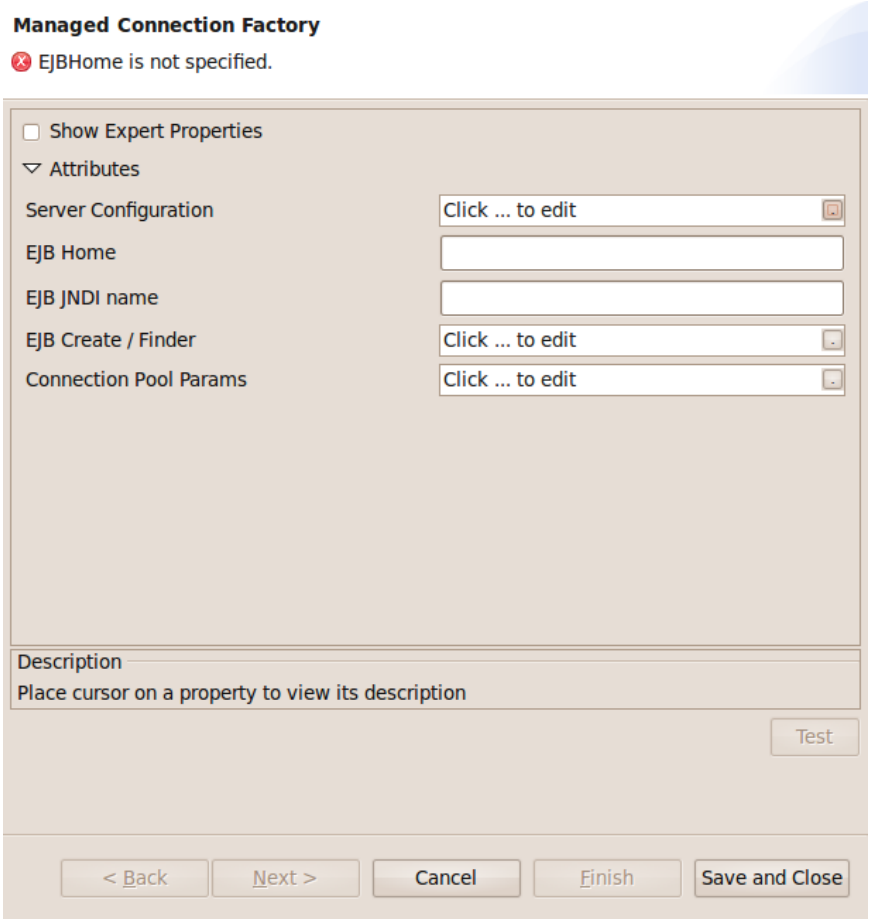


Figure: Modified Managed Connection Factory

Some properties are grouped as one property in the modified configuration panel. Application Server and Initial Context JNDI which were present previously in the MCF panel are grouped under the name Server Configuration.

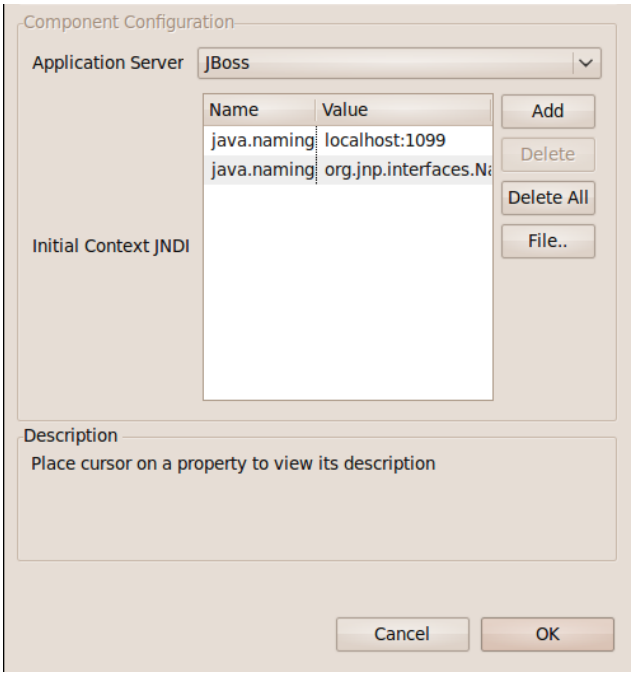


Figure: Server Configuration

4.1.1.3 Functional Demonstration

Configure the EJBAdapter as described in *Configuration and Testing* section and use feeder and display component to send sample input and check the response respectively.

Figure 9: Demonstrating with sample input and output

4.1.1.4 Input Message

<ns1:hello.Request xmlns:ns1="http://www.fiorano.com/fesb/activity/EJBAdapter1/In"/>

When the property Use Connection details from input is chosen, an additional element ConnectionFactorySettings is added to the input schema, as shown in the figure. Properties that are used to create the connection are present under this element.

XML Item	Value Type
<ns1:hello.Request>	
<ConnectionFactorySettings>?	
<ConnectionProperties>?	
<ApplicationServer>?	JBoss | Weblogic Server | IBM Websp...
<IntialContextJNDI>?	#string
<EJBHome>?	#string
<EJBJNDIName>?	#string

4.1.1.5 Output Message

```
<?xml version="1.0" encoding="UTF-8"?>
<hello.Response xmlns="http://www.fiorano.com/fesb/activity/EJBAdapter1/Out">
                          <returnVal xmlns="">Hello World!</returnVal>
</hello.Response>
```

4.1.2 FTPGet

The FTP Get component is used for downloading files from the FTP Server. It can be used for downloading a single file or all files in a directory to a desired location.

Using the FTP Get component, a file can be downloaded by any of the following methods:

6. By monitoring a remote directory for any modifications such as file addition or updation and download of the corresponding file to the desired location.
7. By downloading the file specified in the input message to the desired location.

The FTP Get component uses the FTP protocol for file transmission. The component ensures uninterrupted download by attempting to reconnect to the remote server in case the connection to the server is lost.

4.1.2.1 Managed Connection Factory Panel

The connection properties can be configured using the properties of Managed Connection Factory panel as shown in Figure 4.1.

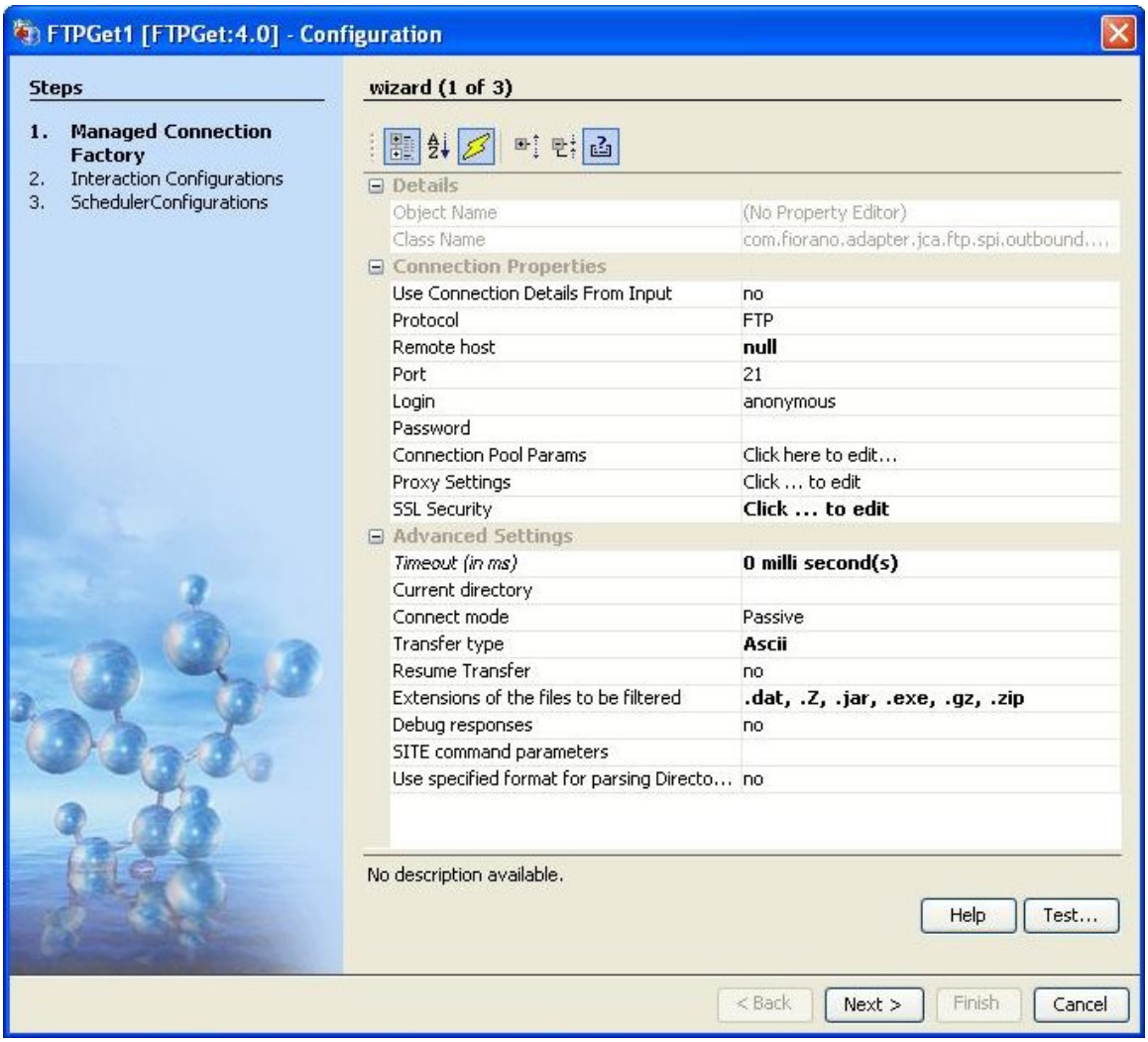


Figure 4.1: Managed Connection Factory panel

4.1.2.1.1 Connection Properties tab

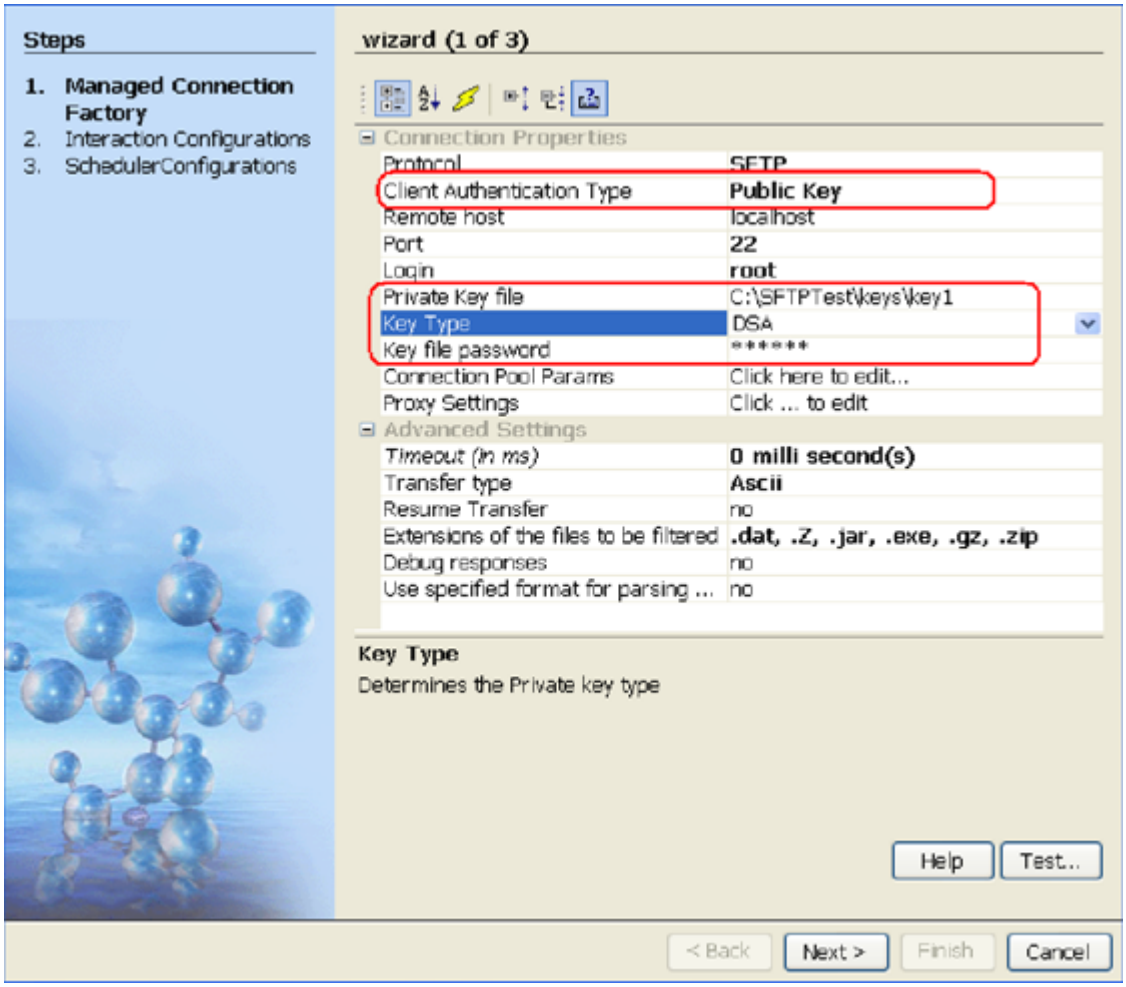
Use Connection Details From Input

Parameters to create the connection can be specified in the input message when this property is set to true. If this property is selected the validation errors in the managed connection factory panel of the CPS are treated as warnings. So user can bypass this step without giving valid configuration and complete the configuration of the component. If valid properties are not provided even in the input message exception will be thrown at runtime

Protocol

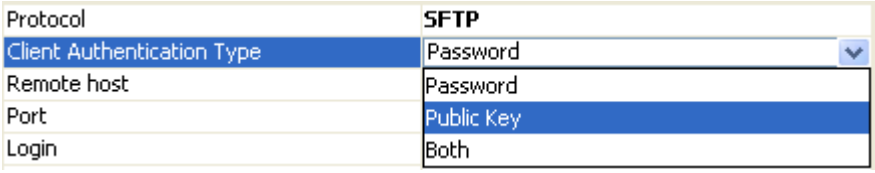
Select the Protocol either FTP or SFTP

- **FTP:** Select this option if you want the protocol to use as File Transfer Protocol.
- **SFTP:** Select this option if you want the protocol to use as Secured File Transfer Protocol. If we select this protocol, then we need to provide **Client Authentication Type**. If we select **Client Authentication Type** as Password, then client's user name and password are sufficient to login successfully. These details can be set by using the properties **Login** and **Password**. If we select **Client Authentication Type** as **Public Key** or **Both** we need to provide the details of **Private Key File**, **Key File Type** and **Key File Password**. For detail explanation of SFTP setting, please refer Scenario 3 under section Functional Demonstrations.



Client Authentication Type

This Property determines the authentication type for the client validation in case of SFTP Protocol. We can select one of the following validations



- **Password:** If this is selected client's username and password are used for client's authentication.
- **Public Key:** If this is selected client's **Private Key** and the **Key File Password** are used for client's authentication. (Key file password is different from client password)
- **Both:** This will authenticate using a private/public key-pair, followed by password authentication. If the authentication fails while using client's private key then it will try to authenticate using password authentication.

Remote Host: The host name/IP address of the machine where the FTP server is running.

Port: The port number on which the FTP server is running.

Login: User name of the FTP user.

Password: Password of the FTP user. This field will be disabled if user selects Protocol as **SFTP** and Client Authentication Type as **Public Key**.

Private Key file

The private key file path in the local machine used for client authentication in case of protocol SFTP. The path should include the file name also. The key file should be present on the machine where the peer server (on which peer the component is running) is running. This property is visible when the Protocol is selected as SFTP and “Client Authentication Type” as “Public Key” or “Both”.

Key Type

Determines the private key type either 'DSA' or 'RSA'. This property is visible when the Protocol is selected as SFTP and “Client Authentication Type” as “Public Key” or “Both”.

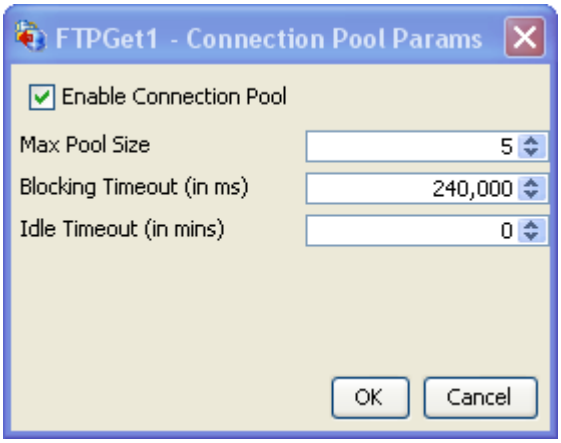
Key file password

The private key file's password in case of protocol SFTP and Client Authentication Type as Public Key or Both.

Note: Key File Password is different from client's password. The properties "Client Authentication Type", "Private Key File", "Key Type" and "Key File Password" are hidden in case of protocol FTP.

Connection Pool Params

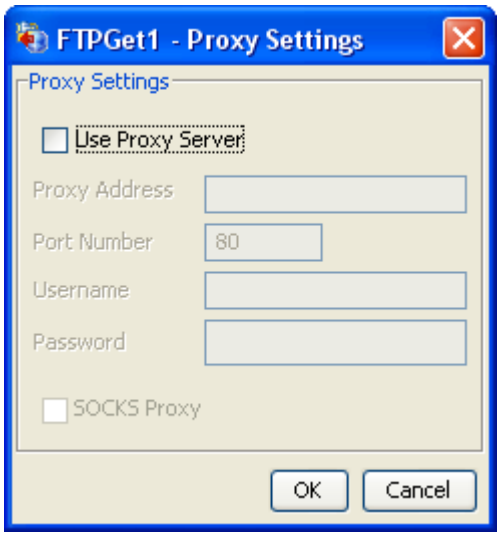
Here the user can specify the details for maintaining the pool of connections in the component. When we click the eclipse (..) button, Connection Pool Params dialog box appear as shown in the figure below.



- **Enable Connection Pool:** If this property was enabled, the connections created are cached in to a pool and used whenever required and available. If disable a new connection will be created for each request. By enabling this we can reduce the time for creating a new connection for every input request. If we disable this a new connection will be created for every request and it will be closed after completion of that request.
- **Max Pool Size:** The maximum number of connections that can be allocated for the pool.
- **Blocking Timeout (in ms):** The time after which the call to fetch a connection from the pool will timeout if there is no unused connection available.
- **Idle Timeout (in ms):** The time after which the idle connections are returned back to the pool

Proxy Settings

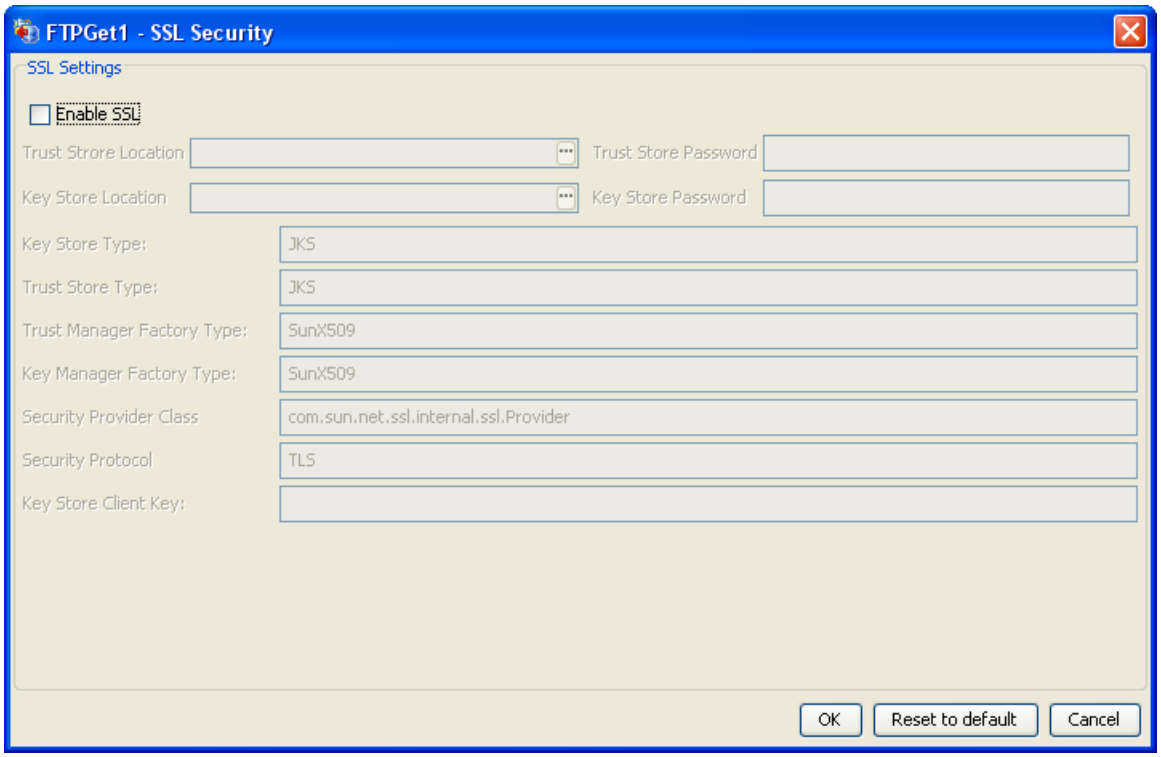
FTP adapters supports HTTP and SOSCKS proxies. HTTP is the default option. Here, the user can configure the proxy server settings.



- **Use Proxy Server:** If enabled Component will use proxy server settings.
- **Proxy Address:** The IP address or the host name of the machine where the proxy server is running. .
- **Port Number:** Port number on which the proxy server is running.
- **Username:** The user name by which the user can login into the proxy server.
- **Password:** Password of the user name by which the user can login into the proxy server.
- **SOCKS Proxy:** Enable this property if the specified proxy server was a socks proxy server.

SSL Security

This property is visible only if we select protocol as FTP.



- **Enable SSL:** By checking this we can enable ssl settings and we can access FTPS (FTP Over SSL) server.
- **Trust store location:** Determines Location of the trust store
- **Trust store Password:** Determines Trust store password
- **Key store location:** Determines Key Store location
- **Key store Password:** Determines Key Store password
- **Key store Type:** Determines Key store type
- **Trust store Type:** Determines Trust store type
- **Trust Manager Factory Type:** Determines Trust Manager Factory type
- **Key Manager Factory Type:** Determines Key Manager Factory type.
- **Security Provide Class:** Determines Security provider class.
- **Security Protocol:** Determines Security protocol
- **Key Store Client Key:** Determines Key Store Client Key

4.1.2.1.2 Advanced Settings tab

Advanced Settings	
Timeout (in ms)	**0 milli second(s)**
Current directory	
Connect mode	Passive
Transfer type	**Ascii**
Resume Transfer	no
Extensions of the files to be filtered	**.dat, .Z, .jar, .exe, .gz, .zip**
Debug responses	no
SITE command parameters	
Use specified format for parsing Director...	no

Timeout (in ms)

The TCP timeout in milliseconds for the sockets. Any operation which takes longer than the timeout value is killed with a java.io.InterruptedException.

Current directory

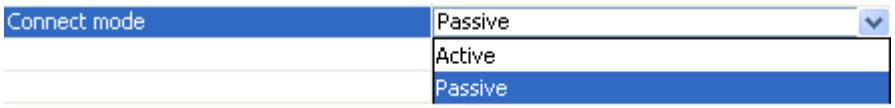
When a user logs in to the FTP server, then the directory to which it is changed to.
All relative paths in the server that are computed by the FTPGet component are relative to this directory.

Example: The user's home directory is /home/user and current directory is set to /home/user/Fiorano then when the user logs into the FTP server, the directory will be changed to /home/user/Fiorano.

If the properties Working directory, Error directory and Processed directory of the Interaction Configurations panel are left to the default values inQueue, errorQueue and processedQueue respectively as shown in Figure 4.10, these directories are created under the directory specified by this property.

Connect mode

We can select the type of FTP connection – Active or Passive. This property will be ignored if we select protocol as SFTP.



- **Active:** In Active mode the FTP client specifies the data port that the FTP server is going to connect on and waits for the FTP server to connect. The IP address and port numbers are sent to the FTP server by the FTP client using the PORT command.

- **Passive:** In passive mode the FTP server specifies the data port that the FTP client will connect on and waits for the FTP client to connect. The FTP client will ask the FTP server for the server's IP address and port number by issuing the PASV command to the FTP server. This will usually solve the problem of firewalls filtering the incoming data connection.

Transfer type

Specifies the connection type

- **Ascii:** When we select Ascii mode the transferred data is considered to contain only ASCII formatted text. If we select Ascii transfer mode then the component is responsible for translating the format of the received text to one that is compatible with the operating system of fps(The fps on which the FTPGet component is running). Text files and files containing HTML, CSS mark-up are suitable for Ascii mode transfer.
- **Binary:** When we select Binary mode of transfer the component transmits raw bytes of the file being transferred. All audio, video and image files are suitable for Binary mode transfer.

Resume Transfer

Resumes ftp transfer from the point where download had stopped in case transfer is broken. Resume of the broken transfer depends on the FTP server. If the FTP server does not support this then the FTP adapter will start from the beginning otherwise it will start from where it was stopped.

Note: Broken transfers will be resumed only when the Transfer type is Binary.

Extensions of the files to be filtered

When files of specific extension should not be downloaded from the server, the file extension has to be specified here. This property accepts comma separated list of file extensions. Example: *.zip, *.exe, *.dat.

Example: If this property is set to .exe and the user specifies to get the file named "installer.exe" in the request, then the component ignores that request.

Debug responses

When FTP responses are needed, enabling this property logs all the FTP responses to the Output Log of the component. Figure 4.2 illustrates a sample snapshot of the debug responses when some download happens.

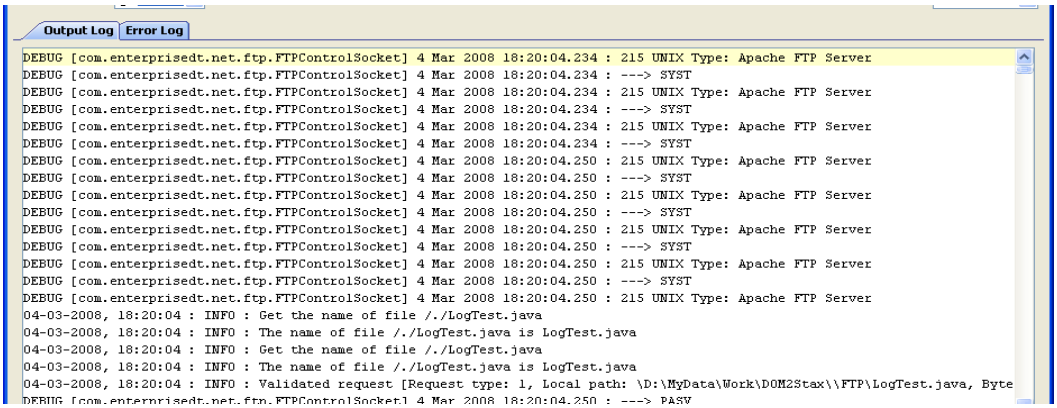


Figure 4.2: FTP responses in FTPGet's output log

SITE command parameters

Site commands are sets of extended commands that can be issued by an ftp client, and they are not defined in RFC. However, they are supported by various ftp servers, and different servers usually have different supported site commands. SITE command is used by the server to provide services specific to the system. All the server administrative tasks can be performed by the SITE command.

This property accepts a semicolon separated list of SITE command parameters that have to be executed immediately after login. These parameters are server dependent.

Example: For OS/400 platform, the server specific format of lists or names can be changed to UNIX type formats by specifying the value **LISTFMT 1; NAMEFMT 1** for this property.

Use specified format for parsing Directory Listing?

This property is used to parse the directory listing of the ftp server. For example in case of Unix the directory is listed as follows

drwxrwxr-x 3 user group 4096 2008-10-23 14:13 fiorano
drwxr-xr-x 14 user group 4096 2008-12-18 14:41 Fiorano

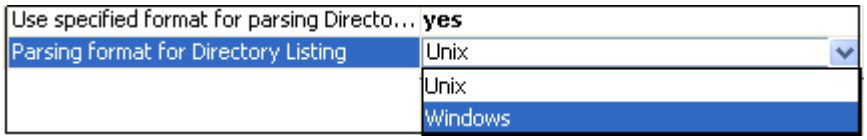
But on Windows the directory is listed in different format. This listing is the output from the ftp server after executing "dir" command.

- **No:** When this property is disabled, the parsing format will be chosen depending on the Operating System on which the FTP server is running.
- **Yes:** Enable this property if a specific parsing format is to be used for parsing the Directory listing returned by the FTP server.

Example: If a FTP server is running on IIS on a Windows machine and its directory listing style is set to UNIX, enable this property and set 'Parsing format for Directory Listing' to UNIX for this property.

Parsing format for Directory Listing

This property is used to determine the format to use for parsing directory listing in the ftp server. The formats supported by the component are Windows and UNIX.



- **Unix:** Select to use Unix format to parse the directory listing
- **Windows:** Select to use Windows format to parse the directory listing

4.1.2.1.3 Testing the Connection

Server connection can be tested from within the CPS by clicking on the Test button in the Managed Connection Factory panel.

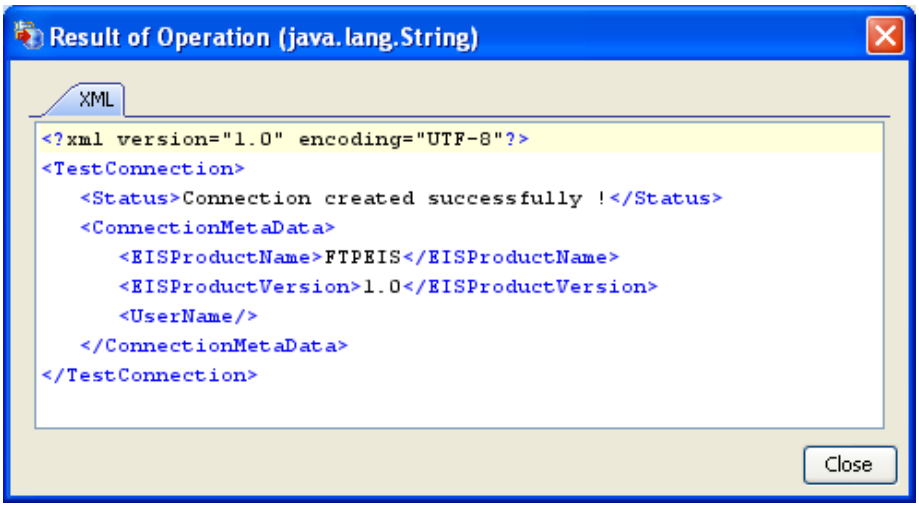


Figure 4.3: Result of successful connection creation

4.1.2.2 Interaction Configurations Panel

Interaction settings can be configured in the Interaction Configurations panel shown in Figure 4.4.

Source Settings	
Response type	File
Send Transfer Progress Notifications	no
Validate Input	no
Cleanup resources (excluding conn...	yes
Delete file after transfer	no
Target Namespace	http://www.fiorano.com/fesb/activity/...
Monitoring configuration	disabled: 5000 milli seconds
Monitoring Settings	
Monitor Directory	no
Target Settings	
Parent directory on the local system	
Target directory	
Use Temporary target directory	yes
Temporary target directory	temp
Action if same file exists	OVERWRITE
Auto create Target Directory	no
Miscellaneous Settings	
Auto create temporary directories	yes

Figure 4.4: Interaction Configurations panel

4.1.2.2.1 Source Setting

Response type

When the response type is File, the file to which the downloaded content is to be written should be specified in the input request to the adapter. The figure below illustrates the input and output schema structures when the response type is a File. Table 1 and Table 2 provide the descriptions for the schema elements of the input and output schema structures respectively.

Note: Input port appears only when Monitoring is disabled (Set the value of Monitor property to **No**).

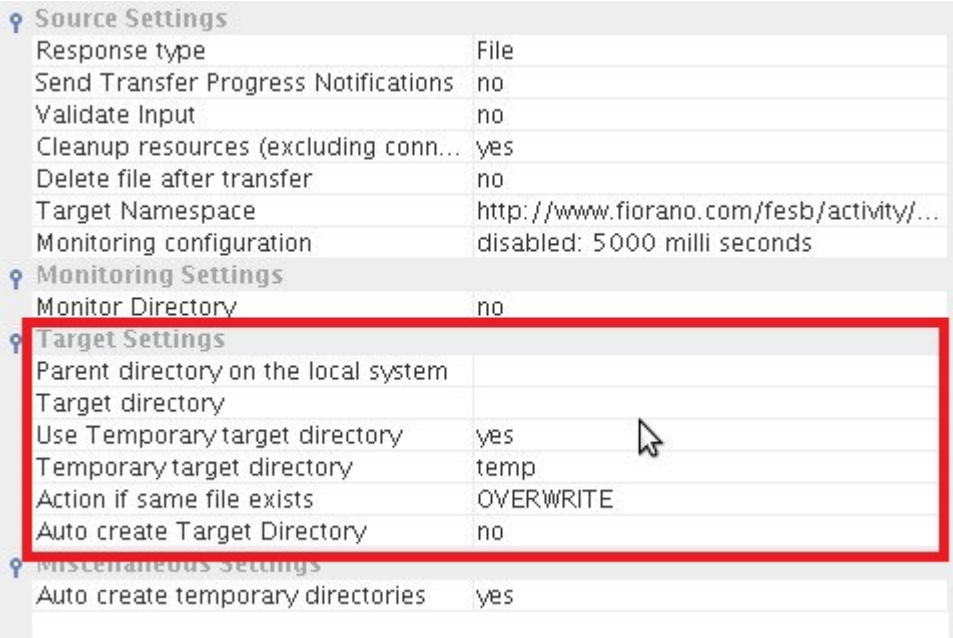


Figure 4.5: Interaction Configurations with response type – File

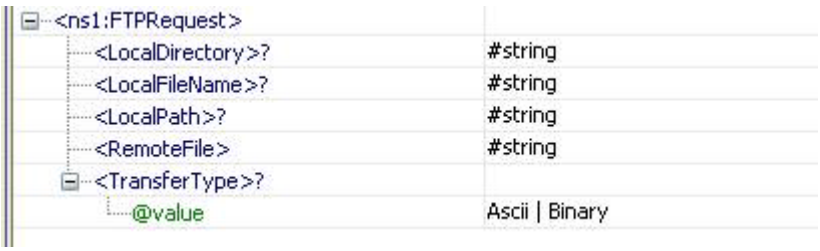


Figure 4.6: Input schema structure for the response type – File

Schema Element	Description
LocalDirectory	The directory on the local file system where the file needs to written
LocalFileName	The name of the file to which the downloaded content is to be written
LocalPath	Path of the local file to which the downloaded content is to be written. Note: When LocalPath is specified, LocalDirectory and LocalFileName need not be specified.
RemoteFile	File on the FTP server which is to be downloaded.
TransferType	Type of data transfer (ASCII or Binary)

Table 1: Input schema element descriptions for - File response type

<ns1:FTPResponse>	
<ns1:LocalPath>	#string
<ns1:RemoteFile>	#string
<ns1:TransferType>	
@value	Ascii | Binary
<ns1:BytesTransferred>?	#string
<ns1:TotalBytes>?	#string
<ns1:ReplyCode>	#string
<ns1:ReplyText>	#string

Figure 8: Output schema structure for the response type – File

Schema Element	Description
LocalPath	Path of the local file to which the downloaded content is written
RemoteFile	File on the FTP server which has been downloaded
TransferType	TransferType mentioned in the input
BytesTransferred	The number of bytes transferred
TotalBytes	The total number of bytes transferred
ReplyCode	The reply code sent by the FTP server
ReplyText	The reply text sent by the FTP server

Table 2: Output schema element descriptions for - File response type

When the response type is chosen as Data, the Target settings in the Interaction Configurations panel change as shown in Figure 9. Now, only one property **Send XML output?** appears in the Target Settings. When the property **Send XML output?** is enabled, the component sends out an XML message which comprises the downloaded data and download status details. When disabled, the component sends out just the downloaded content.

Notes:

- Input port appears only when Monitoring is disabled (Value of the property Monitor Directory set to No).
- Output port do not have any schema defined on it when the Send XML output? is disabled.

Source Settings	
Response type	**Data**
Send Transfer Progress Notifications	no
Delete file after transfer	no
Monitoring configuration	disabled: 5000 milli seconds
Monitoring Settings	
Monitor Directory	no
Target Settings	
Send XML output?	yes

Figure 4.8: Interaction Configurations with response type - Data

Figure 4.9 and Figure 4.10 show the input and output schema structures respectively when the response type is Data. Please refer Table 1 for the description of input schema elements and Table 2 for the description of output schema elements except Data. When the response type is Data, this element holds all the downloaded content.

<ns1:FTPRequest>	
<RemoteFile>	#string
<TransferType>?	
@value	Ascii | Binary

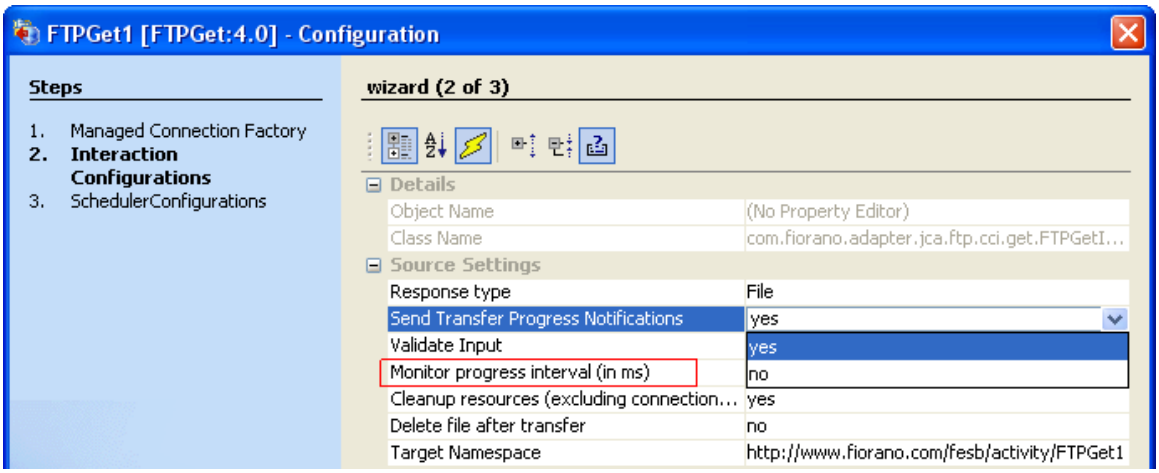
Figure 4.9: Input schema structure for the response type - Data

<ns1:FTPResponse>	
<ns1:Data>	
@dataType	Text | Bytes
#string	
<ns1:RemoteFile>	#string
<ns1:TransferType>	
@value	Ascii | Binary
<ns1:BytesTransferred>?	#string
<ns1:TotalBytes>?	#string
<ns1:ReplyCode>	#string
<ns1:ReplyText>	#string

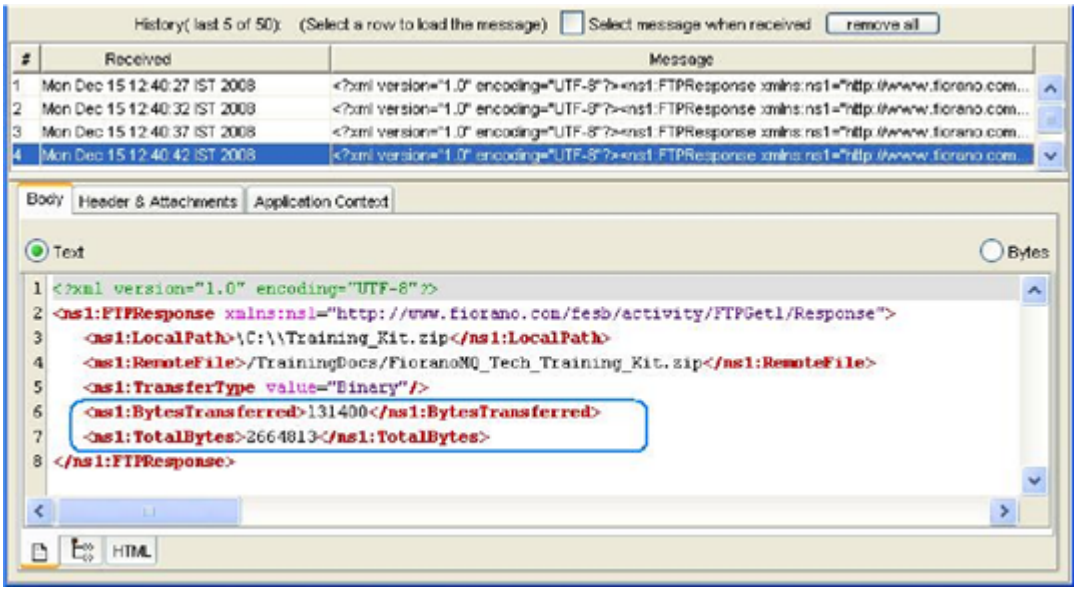
Figure 4.10: Output schema structure for the response type - Data

Send Transfer Progress Notification

If selected **Yes**, then one more options appears as Monitor Progress interval as shown in the figure below. When large files are being downloaded from the FTP server, the progress of the transfer can be obtained by specifying yes to this property.



If this property is selected as **Yes** then the FTPGet adapter will send the notifications of the downloaded process at regular time intervals and this time interval can be specified by the property **Monitor Progress interval** (in ms). The example output of this notification is shown below. Now, observe the BytesTransferred and TotalBytes fields in the output xml.



Validate Input

If this property is set to true, FTPGet adapter will validate the input request with the input port xsd.

Monitor Progress interval (in ms)

The time interval (in milliseconds) between any two progress notifications. This is enabled when the property **Send Transfer Progress Notification** is set to true.

Cleanup resources (excluding connection) after each document

This closes all the resources except to connection used by the FTPGet adapter after every request. If the less processing time is more important the less memory usage, then it is recommended to set this property to **No** and vice versa.

Delete file after transfer

Specifies if the remote file is to be deleted after it is completely downloaded.

Target Namespace

Target Namespace for the FTP request and response XML messages.

4.1.2.2.2 Monitor Settings

- **Monitor Directory**: This property can be used to make the FTPGet adapter poll a directory on the server for files matching a particular pattern and download all such files from the server. Enabling this property makes the FTPGet adapter poll the Source directory using the polling configuration specified in **Scheduler Configurations** panel. The user has to make sure that the Source directory exists on server.

Note: The properties Source directory, File name patterns, Move to working directory, Working directory, Processed directory, Error directory, and Time-based file filtering type are visible only Monitoring is enabled, that is, the property Monitor set to yes. Shown in Figure 4.10 is a sample screen shot with monitoring enabled and the monitoring settings configured. FTPGet takes care of the creation of Working directory, Processed directory and Error directory on the server if the directories do not exist. Working, Processed and Error directories get created under the Current directory specified in Managed Connection Factory panel. If user doesn't prefer moves and the creation of these extra directories then user can set the "No" to the property "**Move to working directory".**

Monitoring is done by monitoring the source directory in regular scheduling interval. The scheduler configurations can be defined by the user in Scheduler Configurations Panel and this is the only case that FTPGet component uses scheduler configurations.

Monitoring Settings	
Monitor Directory	yes
Source directory	source
File name patterns	*.*
Move to working directory	yes
Working directory	inQueue
Processed directory	processedQueue
Error directory	errorQueue
Time-based file filtering type	NONE

Figure 4.12: Sample monitoring configuration

- **Source directory**: The directory on the FTP server containing the files to be downloaded. FTPGet polls this directory using the polling settings configured in SchedulerConfigurations panel.
- **File name patterns**: The type of files in the Source directory which are to be picked up and downloaded. This property accepts multiple file name patterns separated by pipes. Example: *.txt|*.xml|*.exe
- **Move to working directory**: Enabling this property moves a file to Working directory when the file download starts, to Processed directory when the file download is successful, to Error directory when the file download has failed.

  Note: When the user does not possess the move permissions, this property should be set to No.
- **Working directory**: This directory holds the files for which the file transfer is in progress.
- **Processed directory**: This directory holds the files for which the download has been successful.
- **Error directory**: This directory holds the files for which the download has failed.
- **Time-based file filtering type**: This property provides the capability of monitoring only specific files depending on their modification times. This property provides 4 options (as shown in Figure 4.12) based on which the files to be monitored could be filtered.

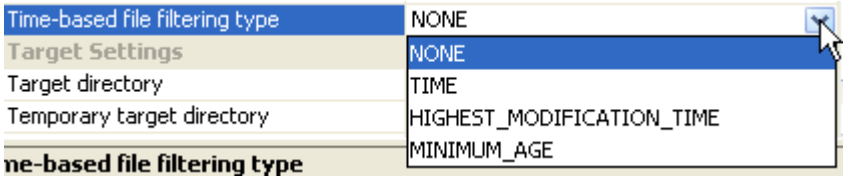


Figure 4.12: Time-based file filtering types

The behavior is as follows:

- **NONE:** No filtering is applied on the files. Every file present in the Source directory is monitored.
- **TIME:** Files whose last modification time is greater than the last polling time is monitored. This ensures that only the files modified/added after the last polling cycle are monitored.

Time-based file filtering type	**TIME**
Base Time	01:01:1970 00:00
Remote host time offset	+00:00

  - **Base Time:** Base time in **dd:MM:yyyy hh:mm** format after which the changed files are to be downloaded
  - **Remote host time offset:** If the FTP server and the component are not in the same time zone, the difference in the time zone of FTP Server time zone from the component's time zone should be specified in (+/-) hh:mm format.

- **HIGHEST_MODIFICATION_TIME:** Files whose last modification time is greater than the highest last modification time found in the last polling cycle is monitored. This ensures that only files which are newer than the newer file already polled are selected.
- **MINIMUM_AGE:** Files whose last modification time is less than the current polling time minus the age is monitored. This ensures that the file modification time is at least Minimum Age earlier than the current time.

Time-based file filtering type	**MINIMUM_AGE**
Minimum age	0 milli seconds
Remote host time offset	+00:00

  - **Minimum age**: The minimum age of the files which are to be monitored.

    Note: This property is visible only when the property Time-based file filtering type is set to **MINIMUM_AGE**.

  - **Remote host time offset:** If the FTP server and the component are not in the same time zone, the difference in the time zone of FTP Server time zone from the component's time zone should be specified in (+/-)hh:mm format.

    Note: This property is visible only when Time-based file filtering type is set to TIME/HIGHEST_MODIFICATION_TIME/MINIMUM_AGE.

    Example: Let us say if the source directory contains 4 files named a.txt, b.txt, c.txt and d.txt. The polling interval is 3 min and the first poll is going to start at 11:00:00 (This polling settings can be configure in Scheduler Configuration Panel, please refer the section "Scheduler Configuration Panel" for more details)

At first poll all files will be monitored irrespective of the value of the property "Time-based file filtering type".

If the files have last modification time like this:

- **a.txt 11:00:16**
- **b.txt 11:00:30**
- **c.txt 11:02:10**
- **d**.txt 11:02:50

If the property **Time-based file filtering type** is set to **TIME**, then all the files will be monitored in the next poll (which is going to poll at 11:03:00), since all files are modified after the last poll.

If the property **Time-based file filtering type** is set to **HIGHEST_MODIFICATION_TIME**, then also all files will be monitored in the next poll (which is going to poll at 11:03:00), since all files are having the last modification time greater than the highest last modification time found on last poll (Component will keep the track of highest last modification time found in the poll).

If the property "**Time-based file filtering type**" is set to **MINIMUM_AGE** and **Minimum age** is set to 5 min, then no files will be monitored in the next poll (which is going to poll at 11:03:00). Files a.txt and b.txt will be monitored in the polling which will be going to poll at 11:06:00, and the files c.txt and d.txt will be monitored in the polling which will be going to poll at 11:09:00, since the files monitored in the particular polling have been modified at least 5 min ago from the polling time.

4.1.2.2.3 Target Settings

- **Send XML output?**: This property appears only when the response type is Data. When disabled, the component sends out only the downloaded content on the output port. When enabled, an XML message containing the downloaded content and download status details is sent out.

  Example: Refer Figure 4.13 and Figure 4.14 which show the sample input and output when this property is enabled. Refer Figure 4.15 and Figure 4.16 which show the sample input and output when this property is disabled.

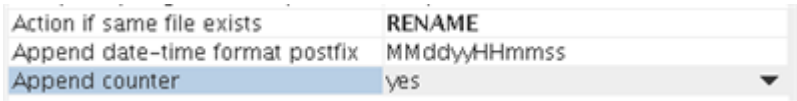
- **Parent Directory on the local system**: Path of the parent directory relative to which a relative path of Target directory would be computed.
- **Target directory**: Directory on the local system to which the file(s) is/are to be downloaded. Note that this property allows relative paths which would be computed relative to the directory specified for Parent Directory on the local system.
- **Use Temporary target Directory:** If this property is set to true, then the FTPGet adapter will use a temporary target directory for intermediate processing. If you do not prefer to create extra directories in ftp server, you can set this property to **No**.
- Temporary target directory: This property is visible only when the property **Use temporary target Directory** is set to **Yes**. Directory on the local machine which the FTPGet component uses for intermediate processing during file downloads.

Note: This directory should not be same as Target directory.

- **Auto Create Target Directory:** Set this property to create target directory. If this is not set, the directory has to be created explicitly by the user.
- **Action if same file exists:** Action that must be taken if the target directory already has a file with name same as the file that is being downloaded. The behavior will be dependant on the selection as shown below.

Figure 4.12: Append settings

- **OVERWRITE:** The file being downloaded overwrites the one in the target directory.
- **RENAME:** The name of the file that is being downloaded is changed based on the properties given below.



  - **Append date-time format postfix:** When existing files in the Target directory are not to be overwritten, FTPGet provides the flexibility of downloading the content into a new file whose name is in the format <NameOfExistingFile_CurrentDateTime>. The format in which the date and time is to be appended should be specified as a value for this property.

    Example: If the date-time format is specified as MMddyyyyHHmmssss for the file Sample.txt, the target file created would be Sample_0305200811300013.txt.

    Note: This property is visible only when Overwrite target file is set to No as shown in Figure 4.12.
  - **Append counter:** Enabling this property appends a counter along with the Date and Time to target file name, when the target file is not to be overwritten.

    Example: A sample file name can be Sample_0305200811300013_0.txt.
- **APPEND:** The file being downloaded gets appended to the one in the target directory.

4.1.2.2.4 Using Named Configurations

Changes are done in eStudio wrt to Named Configurations. With the implementation of Named Configurations MCF panel is changed as shown in Figure below.

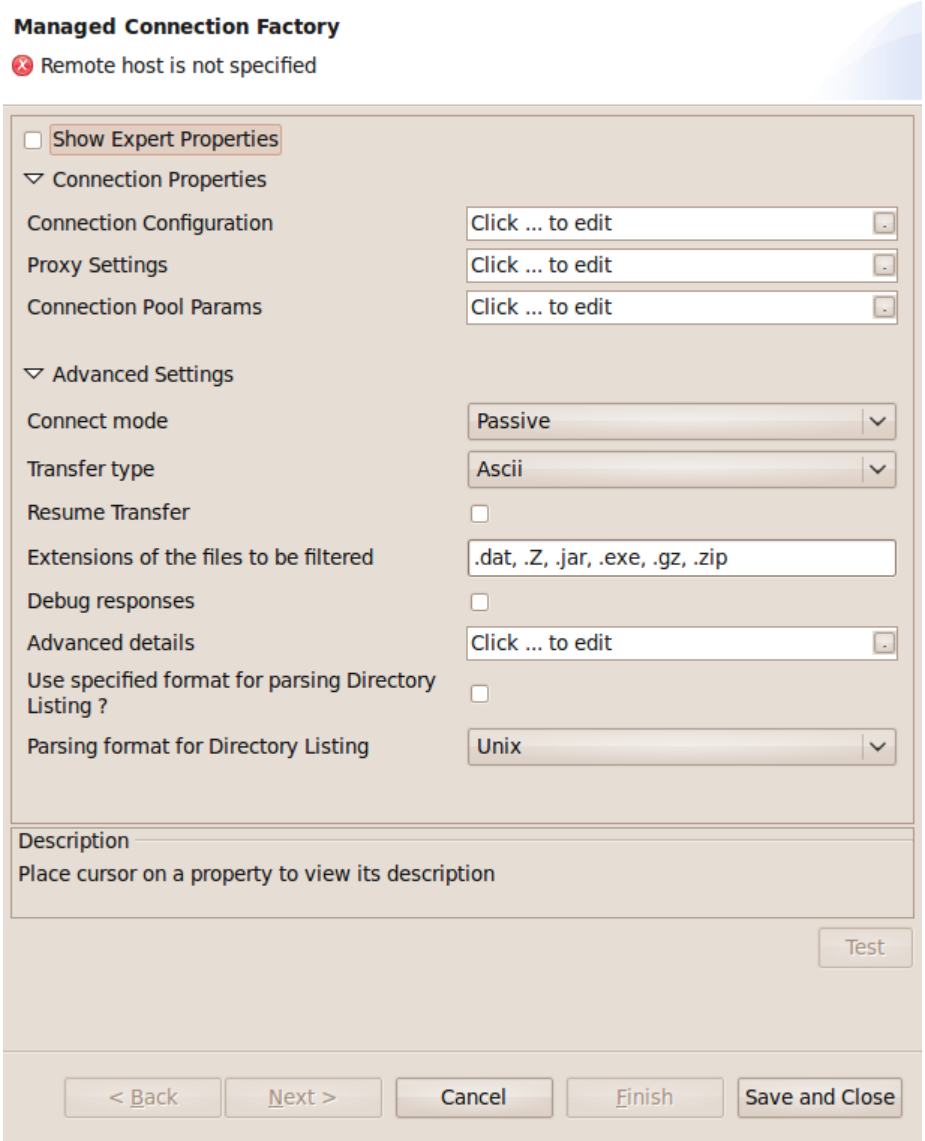


Figure: Modified Managed Connection Factory

Some properties are grouped as one property in the modified configuration panel.Protocol, Remote host, Port, Login, Password and Timeout which were present previously in the MCF panel are grouped under the name Connection Configuration. Current Directory and SITE command parameters which were present previously in the MCF panel are grouped under the name Advanced Details.Proxy Settings and SSL Settings are also modified to use Named Configurations.

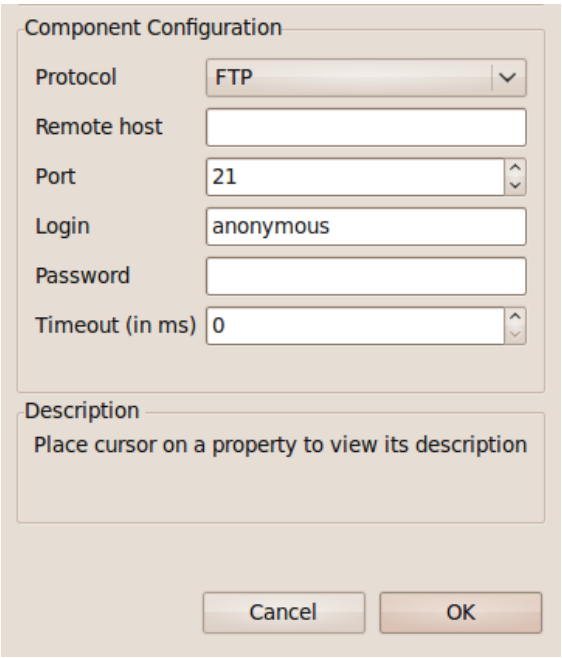


Figure: Connection Configuration

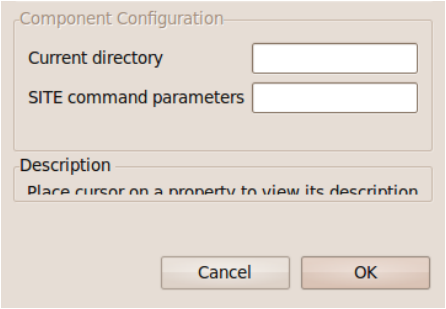


Figure: Advanced Details

4.1.2.2.5 Miscellaneous Settings

Auto create temporary directories: The temporary directories created for processing of the component are created based on this property. If this is not set then the temporary directories must be created explicitly.

4.1.2.3 Miscellaneous Settings

Auto create temporary directories: The temporary directories created for processing of the component are created based on this property. If this is not set then the temporary directories must be created explicitly.

4.1.2.4 Testing the Interaction Configurations

The interaction configurations can be tested by clicking the **Test** button in the panel.

Example 1: Sample input and output when the Response type is set to Data and Send XML Output? is enabled.

Input Message | Output Message

```
<ns1:FTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/FTPGet1/Request">
   <RemoteFile>Sample.txt</RemoteFile>
   <TransferType value="Binary"/>
</ns1:FTPRequest>
```

Figure 4.13: Sample input sent from CPS

Input Message | Output Message

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:FTPResponse xmlns:ns1="http://www.fiorano.com/fesb/activity/FTPGet1/Response">
   <ns1:Data dataType="Bytes">SGVsbG8=</ns1:Data>
   <ns1:RemoteFile>/Sample.txt</ns1:RemoteFile>
   <ns1:TransferType value="Binary"/>
   <ns1:ReplyCode>226</ns1:ReplyCode>
   <ns1:ReplyText>Transfer complete.</ns1:ReplyText>
</ns1:FTPResponse>
```

Figure 4.14: Sample output

Example 2: Sample input and output when the Response type is set to Data and Send XML Output is disabled.

Input Message | Output Message

```
<ns1:FTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/FTPGet1/Request">
   <RemoteFile>Sample.txt</RemoteFile>
   <TransferType value="Ascii"/>
</ns1:FTPRequest>
```

Figure 4.15: Sample input sent from CPS

Input Message | Output Message

```
Hello
```

Figure 4.16: Sample output

The FTP server can be configured in the connection properties panel of CPS.

Connection Properties	
Remote host	localhost
Port	21
Login	anonymous
Password	
Connection Pool Params	Click here to edit...
Proxy Settings	Click ... to edit
Advanced Settings	
Timeout (in ms)	**0 milli second(s)**
Connect mode	Passive
Transfer type	**Ascii**
Resume Transfer	no
Extensions of the files to be filtered	**.dat, .Z, .jar, .exe, .gz, .zip**
Debug responses	no

Figure 4.17: Sample FTP server configuration

Server connection can be tested from within the CPS by clicking on **Test** in the connection properties panel.

Figure 4.18: Sample connection test result indicating success

Sender information can be configured in the **Interaction Properties** panel.

Source Settings	
Response type	File
Send Transfer Progress Notifications	no
Validate Input	no
Cleanup resources (excluding conn...	yes
Delete file after transfer	no
Target Namespace	http://www.fiorano.com/fesb/activity/...
Monitoring configuration	disabled: 5000 milli seconds
Monitoring Settings	
Monitor Directory	no
Target Settings	
Parent directory on the local system	
Target directory	
Use Temporary target directory	yes
Temporary target directory	temp
Action if same file exists	OVERWRITE
Auto create Target Directory	no
Miscellaneous Settings	
Auto create temporary directories	yes

Figure 4.19: Sample Get information configuration

The configuration can be tested by getting a test file when you click on the **Test** option in the **Interaction Properties** panel.

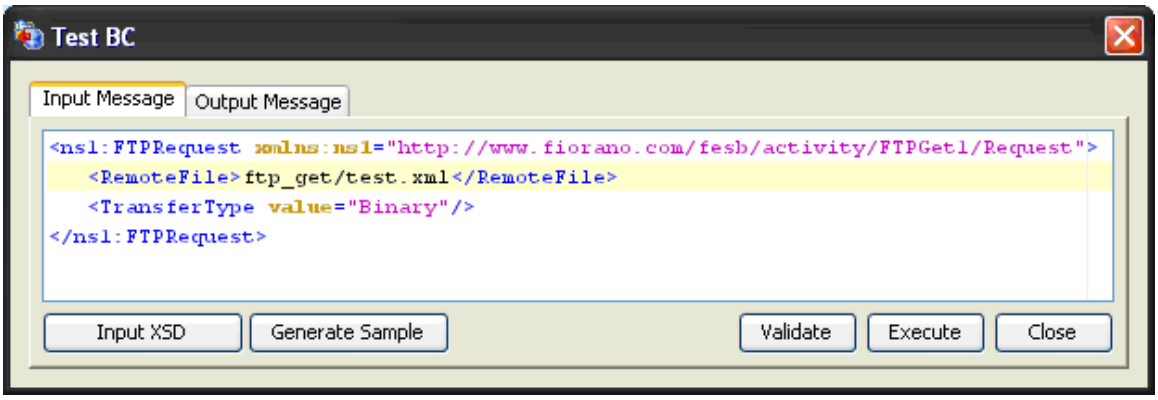


Figure 4.20: Sample input

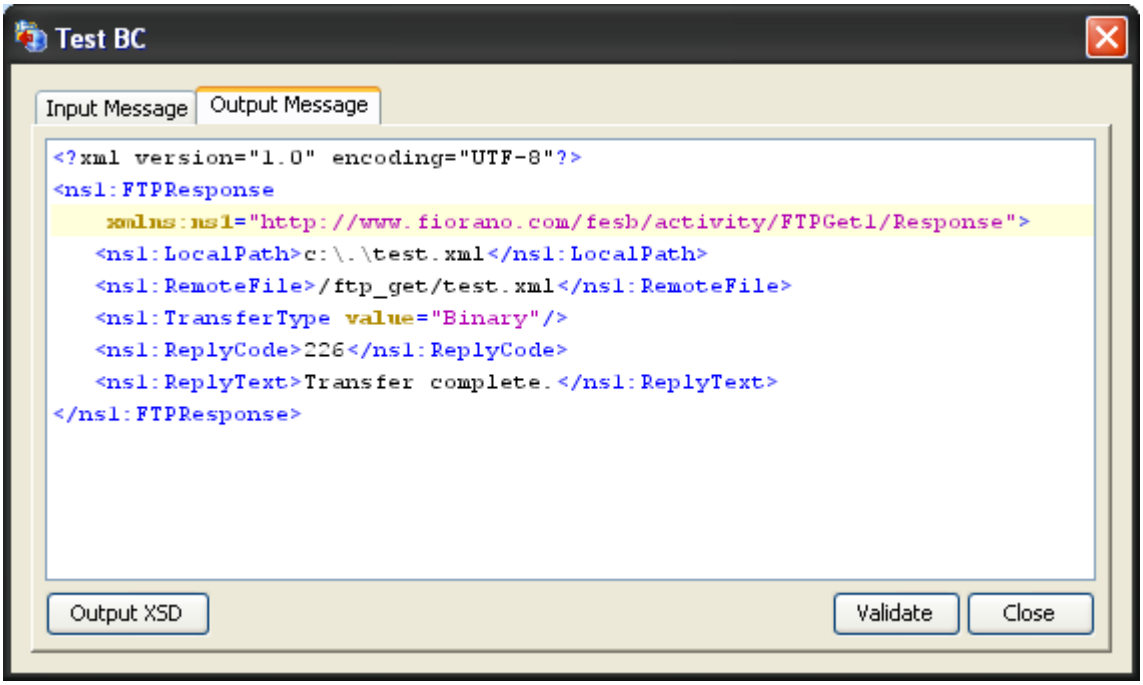


Figure 4.21: Sample response

4.1.2.5 Input Schema

Schema Element	Description
<LocalDirectory>	The directory on the local file system where the file needs to written
<LocalFileName>	The name with which the file needs to be written
<LocalPath>	The local path of the file
<RemoteFile>	The name of the remote file (including path)
<TransferType>	Type of data transfer (Ascii or Binary)

When the property Use Connection details from input is chosen, an additional element ConnectionFactorySettings is added to the input schema, as shown in the figure. Properties that are used to create the connection are present under this element.

XML Item	Value Type
<ns1:FTPRequest>	
<ConnectionFactorySettings>?	
<ConnectionProperties>?	
<Protocol>?	FTP | SFTP
<RemoteHost>?	#string
<Port>?	#int
[Choice]	
<FTPProperties>?	
<Login>?	#string
<Password>?	#string
<SFTPProperties>?	
<ClientAuthenticationtype>?	Password | Public Key | Both
<Login>?	#string
<Password>?	#string
<PrivateKeyFile>?	#string
<KeyType>?	DSA | RSA
<KeyFilePassword>?	#string
<AdvancedProperties>?	
<Timeout>?	#int
<CurrentDir>?	#string
<ConnectMode>?	Active | Passive
<TransferType>?	Ascii | Binary
<FileExtensionsTobeFiltered>?	#string
<DebugResponses>?	true | false
<SiteCommandParameters>?	#string
<ParsingFormatForDirectoryPath>?	
@Enable	true | false
#string	
<SSLSettings>?	

Figure 4.22: Input schema with ConnectionFactorySettings

4.1.2.6 Output Schema

Schema Element	Description
<LocalPath>	The path including file name where the file has been downloaded
<RemoteFile>	The name of the remote file including path
<TransferType>	Type of data transfer (Ascii or Binary)
<BytesTransferred>	The number of bytes transferred
<TotalBytes>	The total number of bytes transferred
<ReplyCode>	The reply code sent by the FTP server
<ReplyText>	The reply text sent by the FTP server

4.1.2.7 Functional Demonstration

4.1.2.7.1 Scenario 1

Receive files from a remote directory on the FTP server and save it in local directory.

Configure the FTP Get as described in *Configuration and Testing* section and use feeder and display component to send sample input and check the response respectively. In the interaction configuration choose the option **File**.

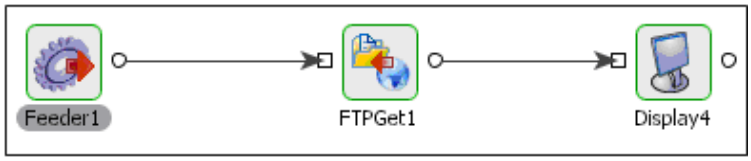


Figure 4.22: Configuration the FTP Get

Sample Input

```
<ns1:FTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/FTPGet1/Request">
    <RemoteFile>ftp_get/test.xml</RemoteFile>
    <TransferType value="Binary"/>
</ns1:FTPRequest>
```

Figure 4.23: Demonstrating Scenario 1 with sample input

Sample Output

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:FTPResponse xmlns:ns1="http://www.fiorano.com/fesb/activity/FTPGet1/Response">
    <ns1:LocalPath>c:\.\test.xml</ns1:LocalPath>
    <ns1:RemoteFile>/ftp_get/test.xml</ns1:RemoteFile>
    <ns1:TransferType value="Binary"/>
    <ns1:ReplyCode>226</ns1:ReplyCode>
    <ns1:ReplyText>Transfer complete.</ns1:ReplyText>
</ns1:FTPResponse>
```

Figure 4.24: Demonstrating Scenario 1 with sample output

4.1.2.7.2 Scenario 2

Receive files from a remote directory on the FTP Server and send it out as data in the output message.

Configure the FTP Get as described in *Configuration and Testing* section and use feeder and display component to send sample input and check the response respectively. In the interaction configuration choose the option Data

Sample Input

```
<ns1:FTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/FTPGet1/Request">
   <RemoteFile>ftp_get/test.xml</RemoteFile>
   <TransferType value="Ascii"/>
</ns1:FTPRequest>
```

Figure 4.25: Demonstrating scenario 2 with sample input

Sample Output

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:FTPResponse xmlns:ns1="http://www.fiorano.com/fesb/activity/FTPGet1/Response">
   <ns1:Data dataType="Text">&lt;?xml version="1.0" encoding="UTF-8"?&gt;&#xD;
&lt;Sample&gt;&#xD;
        &lt;test&gt;test1&lt;/test&gt;&#xD;
        &lt;test&gt;test2&lt;/test&gt;&#xD;
        &lt;test&gt;test3&lt;/test&gt;&#xD;
&lt;/Sample&gt;</ns1:Data>
   <ns1:RemoteFile>/ftp_get/test.xml</ns1:RemoteFile>
   <ns1:TransferType value="Ascii"/>
   <ns1:ReplyCode>226</ns1:ReplyCode>
   <ns1:ReplyText>Transfer complete.</ns1:ReplyText>
</ns1:FTPResponse>
```

Figure 4.26: Demonstrating scenario 2 with sample output

4.1.2.7.3 Scenario 3 (Scenario 1 Using SFTP Protocol)

The following steps give a brief description about server settings and SFTP protocol. Here, we provide steps to test FTPGet adapter using **vsftpd** server which is installed in Linux.

Steps to produce:

1. Install **vsftpd** server by executing the following command on command prompt:

   yum install vsftpd

2. Generate keys pairs (both RSA and DSA) and store in ~/.ssh/ directory. For key generation we can use the following commands:

   ssh-keygen -t rsa (for RSA type key generation)

   ssh-keygen -t dsa (for DSA type key generation)

   Note: While executing the above commands, file name and password for the file has to be provided. If the file name provided is `id_rsa`, then two files will be generated named `id_rsa.pub` and `id_rsa` which are public key and private key respectively.

3. Next, install the public keys in server. Private key has to be with the client which is used to login. The server will authenticate the private key using its public key. This type of client-authentication is called **Public Key Authentication**. The installation of the public key in the server can be done by the executing following command

ssh-copy-id -i ~/.ssh/id_rsa.pub root@localhost

`~/.ssh/id_rsa.pub` is rsa public key file path and **root@localhost** is the server (here we are using the same machine).

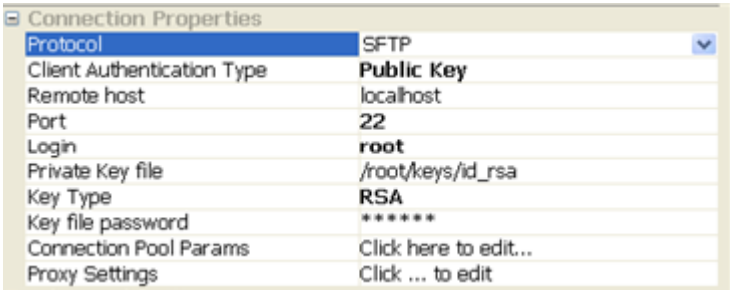
4. Now, change the following in **sshd_config** file (this file can be located in etc/ssh/ folder). Set **RSAAuthentication** or **DSAAuthentication** to yes based on the key file type used.
5. Add the following line IdentityFile ~/.ssh/id_rsa in the **ssh_config** file which was located in /etc/ssh. (if already Identity File is set to some other file, then it has to be modified)
6. Now, restart the servers using the following commands.

   /etc/rc.d/init.d/sshd restart
   /etc/rc.d/init.d/vsftpd restart

Now the server is ready to accept SFTP protocol to login.

4.1.2.8 Configuring FTPGet Component

The configuration of the connection properties of FTPGet component for SFTP protocol is shown in the figure below, and the remaining procedures are same as explained in Scenario 1.



4.1.2.9 Use Case Scenario

In the retail television example media production requests are received on a FTP server and are downloaded using the FTP Get component.

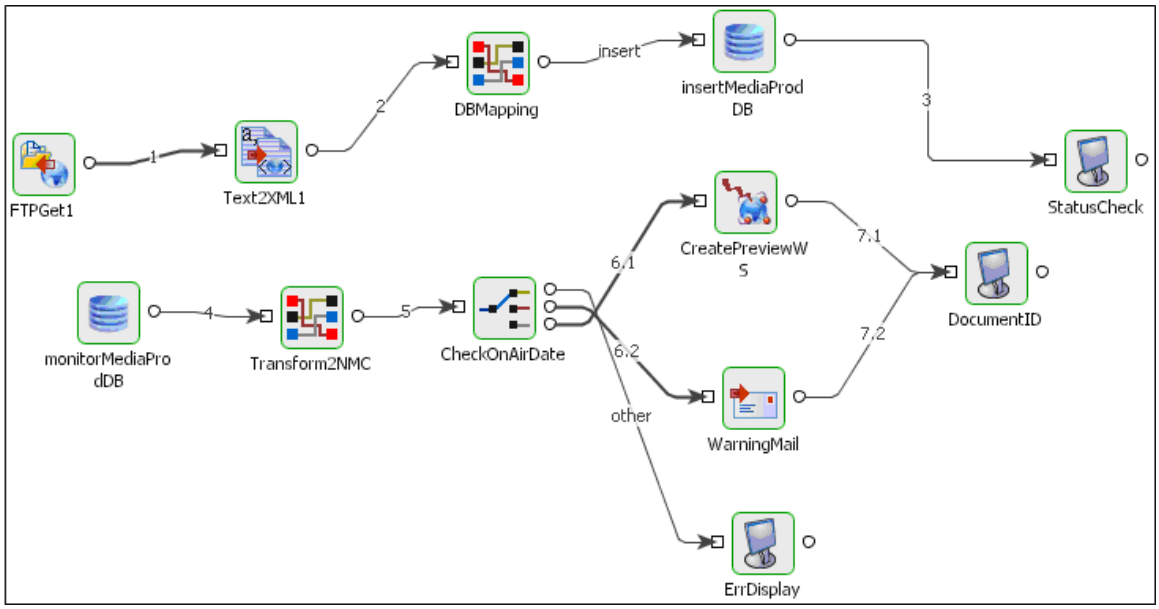


Figure 4.27: Demonstrating scenario

The event process demonstrating this scenario is bundled with the installer. The bundled process shows it as a File Reader component instead of a FTP Get component.

Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Studio.

4.1.2.10 Scheduling

In the FTPGet component, scheduling cannot be directly enabled from the scheduling panel. To enable scheduling, in the interaction configuration panel, set the property Monitor Directory to 'yes' under the Monitoring Settings option as shown in figure 26. This asks for other details which need to be provided in the configuration to do monitoring. Once this is enabled, the scheduling panel details can be modified as required.

Monitoring Settings	
Monitor Directory	yes
Source directory	source
File name patterns	*.*
Move to working directory	yes
Working directory	inQueue
Processed directory	processedQueue
Error directory	errorQueue
Time-based file filtering type	NONE

Figure 28: Monitoring Settings

4.1.2.11 Useful Tips

- In case local file name is not specified in the input message, then the local file name is extracted from the remote file name. The directory shall be the one specified in the CPS for **Target Directory** property.
- The remote file path is relative to the ftp server.
- The component runs on the peer server and therefore the file paths and directories mentioned in the CPS should be valid on the machine where the peer server is runs. If the component fails over to another peer, ensure that the machine on which the secondary peer server is runs does have the same path available.
- If error with error code 550 occurs during request execution, the component logs the message as resource warning but not as error. To get these messages as errors, the property "Throw fault on warnings" must be selected for request processing error in error configuration panel.

4.1.3 FTPPut

The FTP Put component is used for uploading files to the FTP Server. It can be used for uploading one or more files in a directory.

Using the FTP Put component, you can upload the files in any one of the following methods:

- By monitoring file(s) in a local directory for any additions or modifications and uploading the corresponding files to the desired location on the remote host.
- By uploading file(s) specified in the input message.
- By the data contained in the input message.

The FTP Put component uses the FTP protocol for file transmission. The component ensures uninterrupted upload of file by attempting to reconnect to the remote server in case the connection to the server is lost.

Points to note

- It is not mandatory to provide the remote file name. If the remote file name is not provided or it is null, then the remote file name is extracted from the Local File Path. Providing local file path is mandatory.
- The remote file path is relative to the ftp server.
- The component runs on the peer server and therefore the file paths and directories mentioned in the CPS should be valid on the machine where the peer server is running. If the component fails over to another peer, ensure that the machine on which the secondary peer server is running does have the same path available.

4.1.3.1 Managed Connection Factory Panel

The connection properties can be configured using the properties of Managed Connection Factory panel as shown in Figure 1.

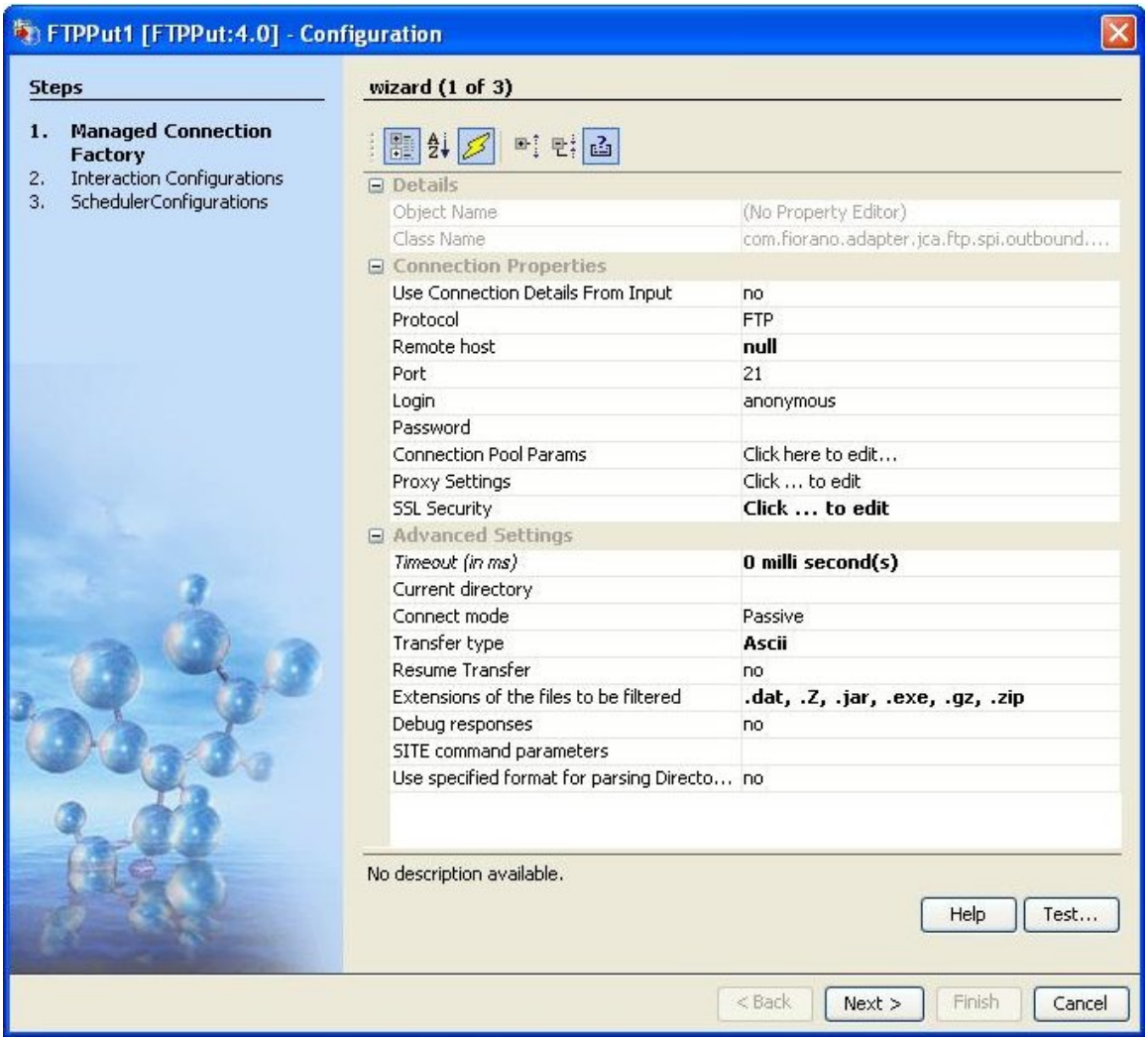


Figure 1: Managed Connection Factory panel

4.1.3.1.1 Connection Properties

Use Connection Details From Input

Parameters to create the connection can be specified in the input message when this property is set to true. If this property is selected the validation errors in the managed connection factory panel of the CPS are treated as warnings. So user can bypass this step without giving valid configuration and complete the configuration of the component. If valid properties are not provided even in the input message exception will be thrown at runtime.

Protocol

- **FTP** - File Transfer Protocol is used for transferring files to FTP server.
- **SFTP** - Secure File Transfer Protocol is used for file transfers. If selected, then the property Client Authentication Type is enabled. If Client Authentication Type is set to Password, then client's user name and password are sufficient to login successfully. These details can be set by using the properties **Login** and **Password**. If Client Authentication Type is set as **Public Key** or **Both**, then the details of Private Key File, Key File Type and Key File Password has to be provided. For detail explanation of SFTP setting, please refer Scenario 3 under section Functional Demonstrations.

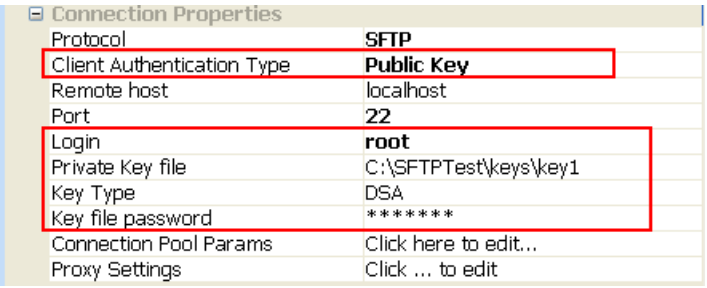


Figure 2: SFTP enabled

Client Authentication Type

This property determines the authentication type for the client validation in case of SFTP Protocol.

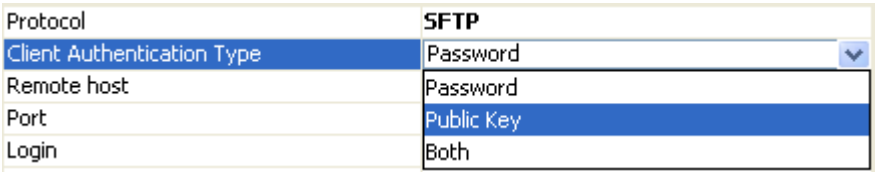


Figure 3: Client Authentication Type - SFTP

- **Password:** If selected, client's username and password are used for client's authentication.
- **Public Key:** If selected, client's **Private Key** and the **Key File Password** are used for client's authentication. (Key file password is different from client password)
- **Both:** This option authenticates using a private/public key-pair, followed by password authentication. If the authentication fails while using client's private key, then it will try to authenticate using password authentication.

Remote Host

The host name/IP address of the machine where the FTP server is running.

Port

The port number on which the FTP server is running.

Login

User name of the FTP user.

Password

Password of the FTP user. This field will be disabled if user selects Protocol as `SFTP` and Client Authentication Type as `Public Key`.

Private Key file

The private key file path in the local machine used for client authentication in case of protocol SFTP. The path should include the file name also. The key file should be present on the machine where the peer server (on which peer the component is running) is running. This property is visible when the Protocol is selected as SFTP and Client Authentication Type as Public Key or Both.

Key Type

Determines the private key type either 'DSA' or 'RSA'. This property is visible when the Protocol is selected as SFTP and Client Authentication Type as Public Key or Both.

Key file password

The private key file's password when the property **protocol** is set to **SFTP** and the property **Client Authentication Type** is set to **Public Key** or **Both**.

Note: Key File password is different from client's password.

Connection Pool Params

Here the user can specify the details for maintaining the pool of connections in the component. On clicking the eclipsis (..) button, Connection Pool Params dialog box appear as shown in the Figure 4.

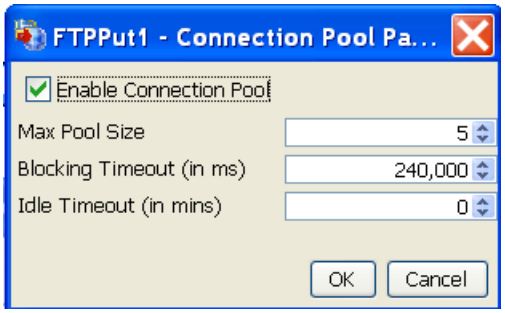


Figure 4: Connection Pool Params

- **Enable Connection Pool:** If enabled, the connections created are cached in to a pool and used whenever required and available. This can reduce the time for creating a new connection for every input request. If disabled, a new connection is created for each request and it will be closed after completion of that request.
- **Max Pool Size:** The maximum number of connections that can be allocated for the pool.

- **Blocking Timeout (in ms):** The time after which the call to fetch a connection from the pool will timeout if there is no unused connection available.
- **Idle Timeout (in ms):** The time after which the idle connections are returned back to the pool

Proxy Settings

FTP adapters support HTTP and SOCKS proxies. HTTP is the default option. Here, the user can configure the proxy server settings.

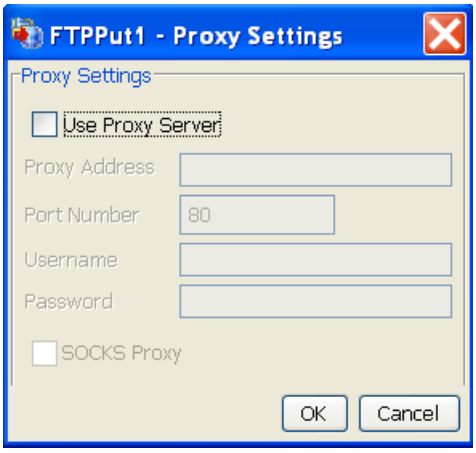


Figure 5: Proxy Settings

- **Use Proxy Server:** If enabled, component will use proxy server settings.
- **Proxy Address:** The IP address or the host name of the machine where the proxy server is running.
- **Port Number:** Port number on which the proxy server is running.
- **Username:** The user name by which the user can login into the proxy server.
- **Password:** Password of the user name by which the user can login into the proxy server.
- **SOCKS Proxy:** Enable this property if the specified proxy server is a SOCKS proxy server.

SSL Security

This property is visible only if the property **Protocol** is set to FTP.

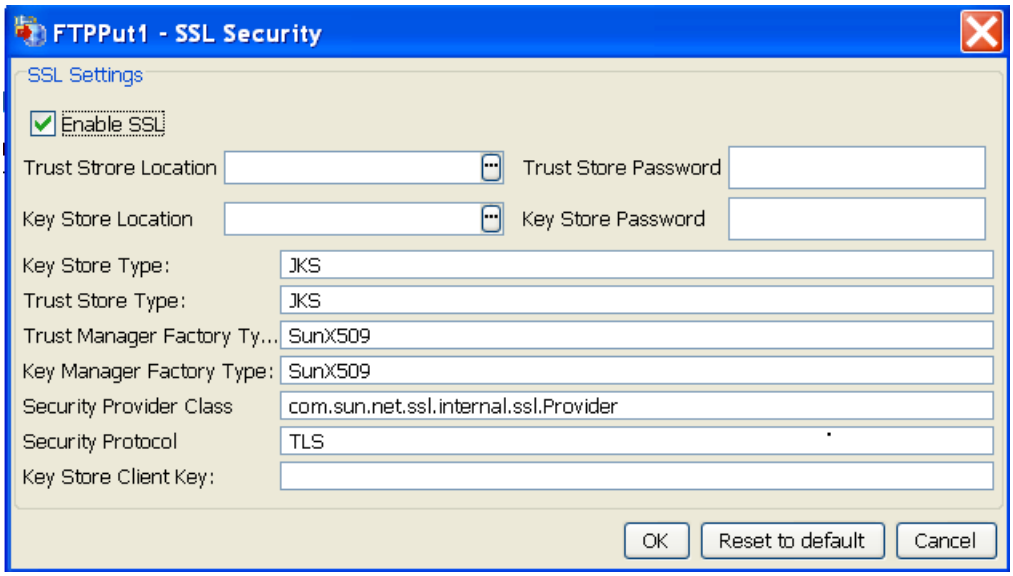


Figure 6: SSL Settings

- **Enable SSL:** Enable SSL settings to access FTPS (FTP Over SSL) server.
- **Trust store location:** Determines Location of the trust store
- **Trust store Password:** Determines Trust store password
- **Key store location:** Determines Key Store location
- **Key store Password:** Determines Key Store password
- **Key store Type:** Determines Key store type
- **Trust store Type:** Determines Trust store type
- **Trust Manager Factory Type:** Determines Trust Manager Factory type
- **Key Manager Factory Type:** Determines Key Manager Factory type.
- **Security Provide Class:** Determines Security provider class.
- **Security Protocol:** Determines Security protocol
- **Key Store Client Key:** Determines Key Store Client Key

4.1.3.1.2 Advanced Settings tab

Advanced Settings	
Timeout (in ms)	**0 milli second(s)**
Current directory	
Connect mode	Passive
Transfer type	**Ascii**
Resume Transfer	no
Extensions of the files to be filtered	**.dat, .Z, .jar, .exe, .gz, .zip**
Debug responses	no
SITE command parameters	
Use specified format for parsing Director...	no

Figure 7: Advanced Settings

Timeout (in ms)

The TCP timeout in milliseconds for the sockets. Any operation which takes longer than the timeout value is killed with a `java.io.InterruptedException`.

Current directory

The directory on FTP server to which the user's current working directory will be changed after the login to FTP. The behavior is similar to executing the command **cd <directory_name>** after logging in, where <directory_name> is the value provided for this property.
All relative paths in the server that are computed by the FTPPut component are relative to this directory.

Example: If the default working directory for the user is /home/user and current directory is set to /home/user/Fiorano, then the working for the user will be changed to /home/user/Fiorano after the user logs into the FTP server.

If the value of property **Use Temporary target directory** is set to **yes** and the value for property **Temporary target directory** is set to **temp**, then a directory **temp** will be created under the directory specified by this property.

Connect mode

Determines the type of FTP connection – Active or Passive**.** This property is ignored if **Protocol** is set as **SFTP.**

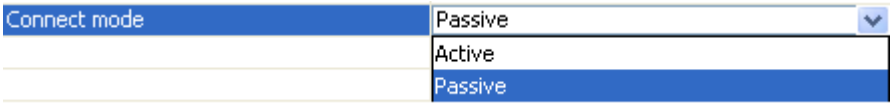


Figure 8: Connect Mode

- **Active:** In Active mode the FTP client specifies the data port that the FTP server is going to connect on and waits for the FTP server to connect. The IP address and port numbers are sent to the FTP server by the FTP client using the PORT command.

- **Passive:** In passive mode the FTP server specifies the data port on which the FTP client connects and waits for the FTP client to connect. The FTP client will ask the FTP server for the server's IP address and port number by issuing the PASV command to the FTP server. This will usually solve the problem of firewalls filtering the incoming data connection.

Transfer type

Specifies the data transfer type

Figure 9: Transfer Type

- **Ascii:** The transferred data is considered to contain only ASCII formatted text. The component is responsible for translating the format of the received text to one that is compatible with the operating system of fps (The fps on which the FTPGet component is running). Text files and files containing HTML, CSS mark-up are suitable for Ascii mode transfer.
- **Binary:** The component transmits raw bytes of the file being transferred. All audio, video and image files are suitable for Binary mode transfer.

Resume Transfer

Resumes FTP transfer from the point where download has stopped if the transfer is broken. Resume of the broken transfer depends on the FTP server. If the FTP server does not support this then the FTP adapter will start from the beginning otherwise it will start from where it was stopped.

Note: The process of transfer is resume from the broken point only when the **Transfer type** is Binary.

Extensions of the files to be filtered

If files with specific extensions have to be restricted for download from the server, the file extension has to be specified here. This property accepts comma separated list of file extensions. Example: *.zip, *.exe, *.dat.

Example: If this property is set to .exe and the user specifies to put the file named "installer.exe" in the request, then the component ignores that request.

Debug responses

When FTP responses are needed, enabling this property logs all the FTP responses to the Output Log of the component. Figure 10 illustrates a sample snapshot of the debug responses when some download happens.

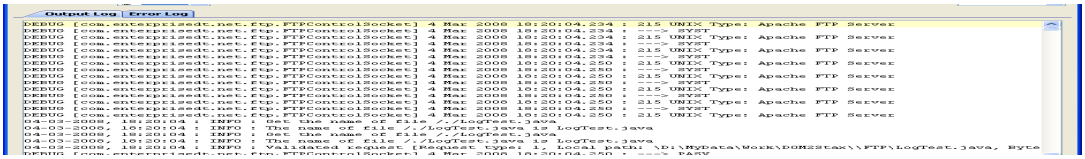


Figure 10: FTP responses in FTPPut's output log

SITE command parameters

Site commands are sets of extended commands that can be issued by a FTP client, and they are not defined in RFC. However, they are supported by different FTP servers, and different servers usually have different supported site commands. SITE command is used by the server to provide services specific to the system. All the server administrative tasks can be performed by the SITE command.

This property accepts a semicolon separated list of SITE command parameters that have to be executed immediately after login. These parameters are server dependent.

Example: For OS/400 platform, the server specific format of lists or names can be changed to Unix type formats by specifying the value **LISTFMT 1; NAMEFMT 1** for this property.

Use specified format for parsing Directory Listing?

This property is used to parse the directory listing of the FTP Server. For example, in case of Unix, the directory is listed as follows:

drwxrwxr-x 3 user group 4096 2008-10-23 14:13 fiorano
drwxr-xr-x 14 user group 4096 2008-12-18 14:41 Fiorano

But on Windows, the directory is listed in different format. This listing is the output from the FTP server after executing the **dir** command.

- **No:** When this property is disabled, the parsing format will be chosen depending on the operating system on which the FTP Server is running.
- **Yes:** Enable this property if a specific parsing format is to be used for parsing the Directory listing returned by the FTP Server.

Example: If a FTP Server is running on IIS on a Windows machine and its directory listing style is set to Unix, enable this property and set '**Parsing format for Directory Listing**' to Unix for this property.

Parsing format for Directory Listing

This property is used to determine the format to use for parsing directory listing in the FTP Server. The formats supported by the component are Windows and Unix.

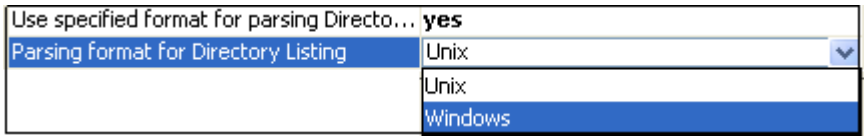


Figure 11: Parsing format for Directory Listing

- **Unix:** Select to use Unix format to parse the directory listing
- **Windows:** Select to use Windows format to parse the directory listing

4.1.3.1.3 Using Named Configurations

Changes are done in eStudio wrt to Named Configurations. With the implementation of Named Configurations MCF panel is changed as shown in Figure below.

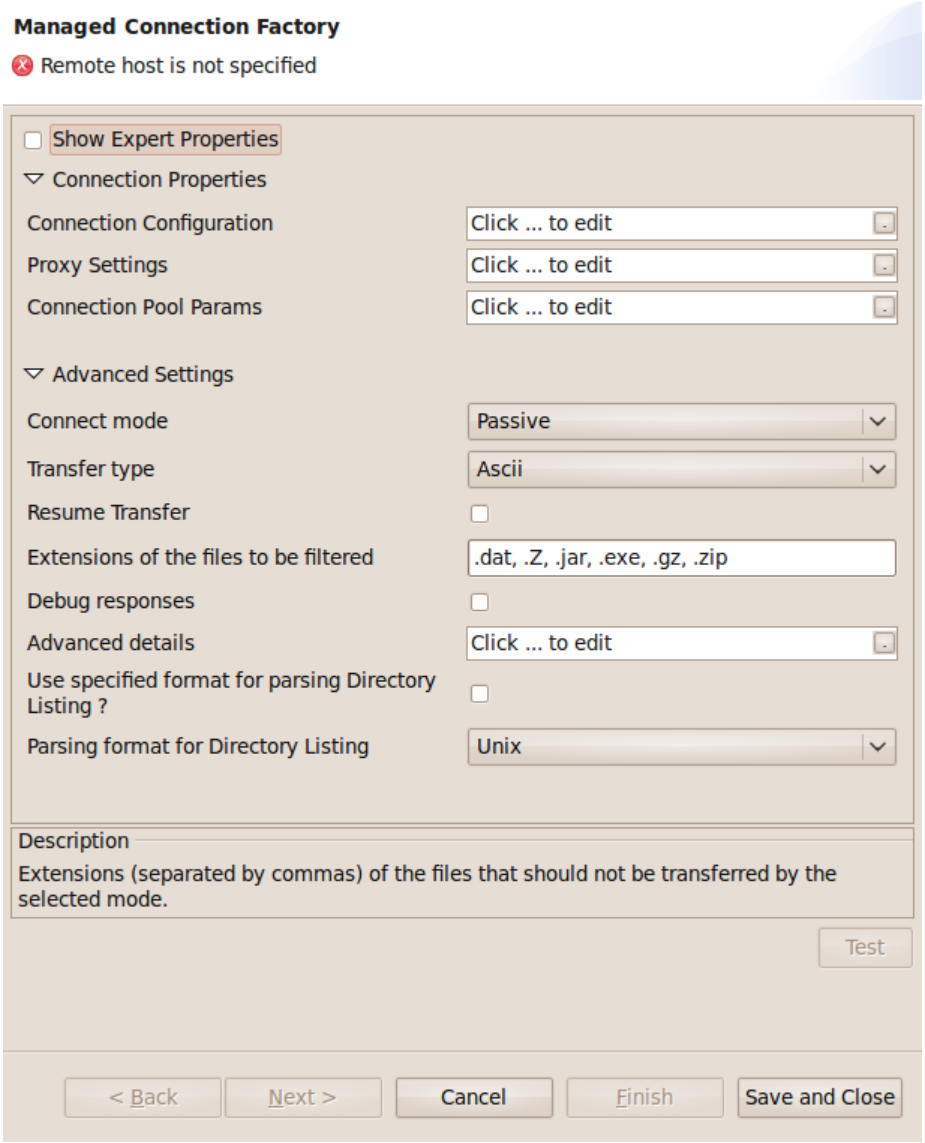


Figure: Modified Managed Connection Factory

Some properties are grouped as one property in the modified configuration panel. Protocol, Remote host, Port, Login, Password and Timeout which were present previously in the MCF panel are grouped under the name Connection Configuration. Current Directory and SITE command parameters which were present previously in the MCF panel are grouped under the name Advanced Details.Proxy Settings and SSL Settings are also modified to use Named Configurations.

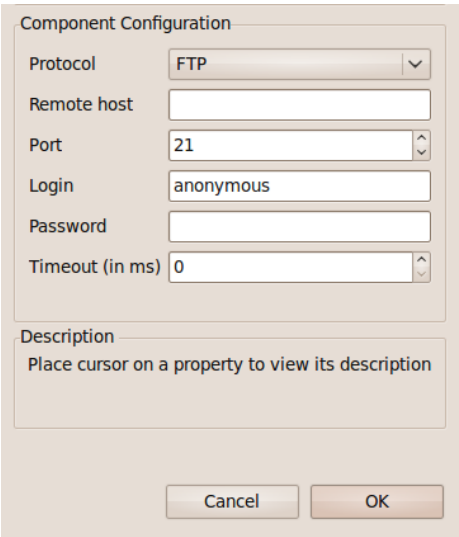


Figure: Connection Configuration

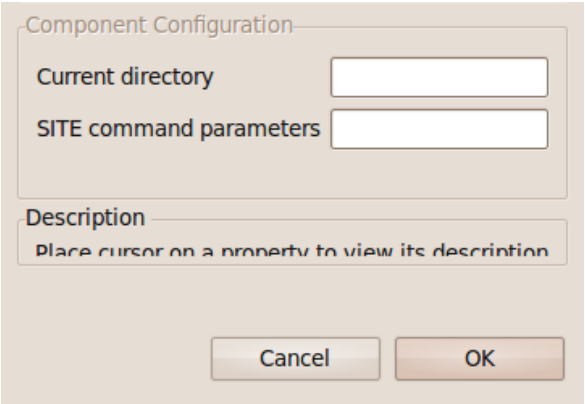


Figure: Advanced Details

4.1.3.1.4 Testing the Connection

Server connection can be tested from within the CPS by clicking on the Test button in the Managed Connection Factory panel.

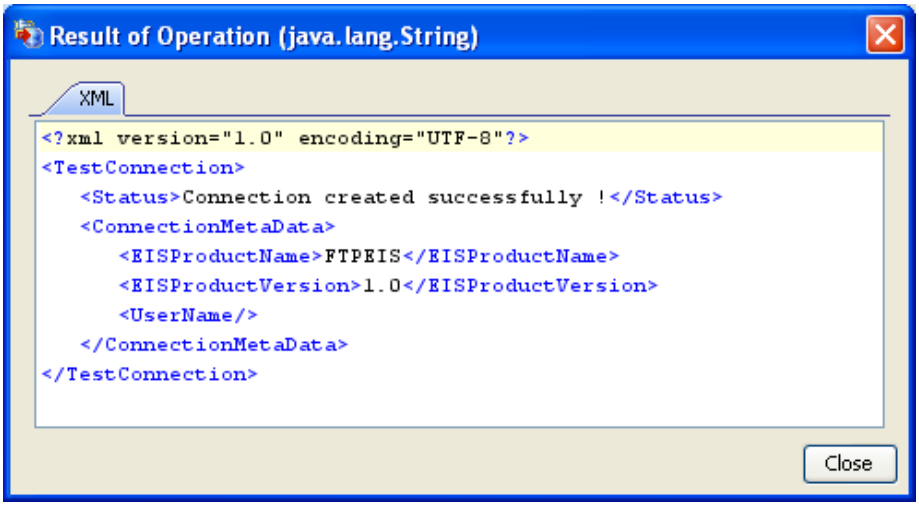
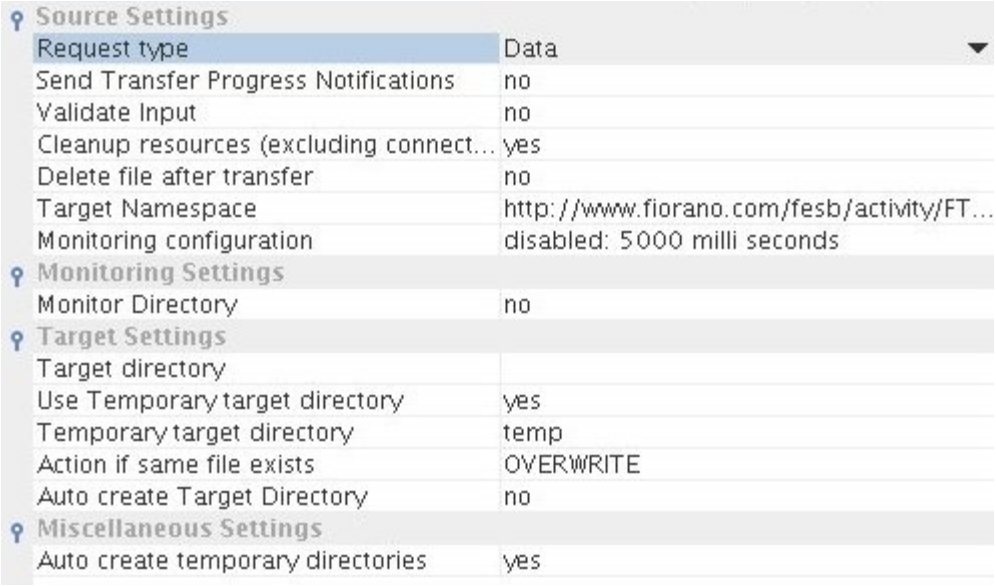


Figure 12: Result of successful connection creation

After selecting the appropriate parameters, click the **Next** button. The **Interaction Configurations** panel appears as shown in the Figure 12.

4.1.3.2 Interaction Configurations Panel



Source Settings	
Request type	Data
Send Transfer Progress Notifications	no
Validate Input	no
Cleanup resources (excluding connect...	yes
Delete file after transfer	no
Target Namespace	http://www.fiorano.com/fesb/activity/FT...
Monitoring configuration	disabled: 5000 milli seconds
Monitoring Settings	
Monitor Directory	no
Target Settings	
Target directory	
Use Temporary target directory	yes
Temporary target directory	temp
Action if same file exists	OVERWRITE
Auto create Target Directory	no
Miscellaneous Settings	
Auto create temporary directories	yes

Figure 13: Interaction Configurations

4.1.3.2.1 Source Settings

Request type

Specifies the type of user input to the adapter. This property provides two options:

- File
- Data

When the request type is **File**, the path of the local file which is to be transferred is specified in the input.

Note: Input port appears only when Monitoring is disabled (Value of the property **Monitor** set to **No**).

Refer to section **Input and Output** for details about the effects of these configurations on input and output structures.

Send Transfer Progress Notification

When large files are being uploaded to the FTP server, the progress of the transfer can be obtained by specifying yes to this property. If this property is selected as **Yes** then the FTP adapter will send the notifications of the uploaded process at regular time intervals and this time interval can be specified by the property **Monitor Progress interval (in ms)**. The example output of this notification is shown below. Now, observe the `BytesTransferred` and `TotalBytes` fields in the output XML.

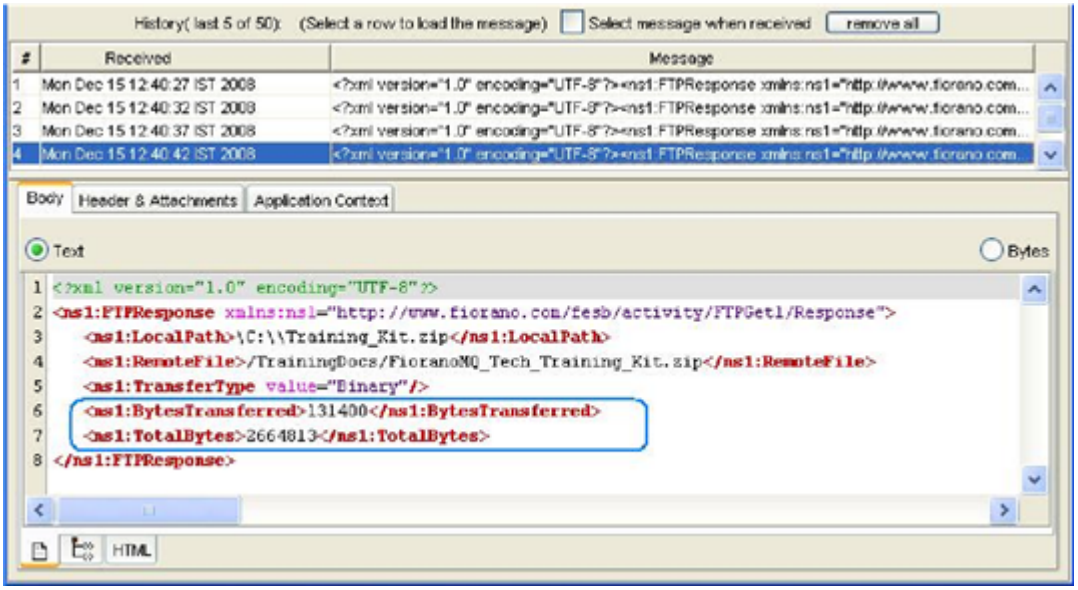


Figure 14: Sample output when 'Send Transfer Progress Notification' is set.

Validate Input

If this property is set to true, FTPGet adapter will validate the input request with the input port xsd.

Monitor Progress interval (in ms)

The time interval (in milliseconds) between any two progress notifications. This is visible when the property **Send Transfer Progress Notification** is set to true.

Cleanup resources (excluding connection) after each document

This closes all the resources except to connection used by the FTPPut adapter after every request. If the less processing time is more important than the less memory usage, then it is recommended to set this property to **No** and vice versa.

Delete file after transfer

Specifies if the remote file is to be deleted after it is completely uploaded.

Target Namespace

Target Namespace for the FTP request and response XML messages.

4.1.3.2.2 Monitor Settings

Monitor Directory

This property can be used to make the FTPPut adapter poll a directory on the peer server's system(on which the component is launched) for files matching a particular pattern and upload all such files to the server. Enabling this property makes the FTPPut adapter poll the Source directory using the polling configuration specified in **Scheduler Configurations** panel. The user has to make sure that the Source directory exists.

Note: Properties Source directory, File name patterns, Move to working directory, Working directory, Processed directory, Error directory and Time-based file filtering type are visible only when Monitoring is enabled, that is, the property Monitor is set to yes. Shown in Figure 15 is a sample screen shot with monitoring enabled and the monitoring settings configured. FTPut takes care of the creation of Working directory, Processed directory and Error directory on the system where peer server is running (the peer on which the component is launched), if the directories do not exist Working, Processed and Error directories get created under the Current directory specified in Managed Connection Factory panel. If user does not prefer moves and the creation of these extra directories, then user can set the value for property Move to working directory to NO

Monitoring is done by monitoring the source directory in regular scheduling interval. The scheduler configurations can be defined by the user in Scheduler Configurations Panel and this is the only case that FTPPut component uses scheduler configurations.

Monitoring Settings	
Monitor Directory	yes
Source directory	source
File name patterns	*.*
Move to working directory	yes
Working directory	inQueue
Processed directory	processedQueue
Error directory	errorQueue
Time-based file filtering type	NONE

Figure 15: Sample monitoring configuration

Source directory

The directory which contains the files to be uploaded. FTPPut component polls this directory using the polling settings configured in Scheduler Configurations panel.

File name patterns

The type of files in the Source directory which are to be picked up and downloaded. This property accepts multiple file name patterns separated by pipes. **Example**: *.txt|*.xml|*.exe

Move to working directory

When this property is set to **yes**, the file that has to be uploaded to FTP server will be moved from the directory specified by property **Source directory** to the directory specified by the property **Working directory** before the upload begins. The file is read from the working directory and uploaded to the FTP server.

- If the upload is successful the file is moved from working directory to the directory specified by property **Processed directory**.
- If the upload is not successful the file is moved from working directory to the directory specified by property **Error directory**.
- When this property is set to **no**, the file is read directly from the directory specified by property **Source directory**.

Note: When the user has read-only permission to the file system, this property should be set to **no**.

Working directory

This directory holds the files for which the file transfer is in progress.

Processed directory

This directory holds the files for which the upload has been successful.

Error directory

This directory holds the files for which the upload has failed.

Time-based file filtering type

This property provides the capability of monitoring only specific files depending on their modification times. This property provides 4 options (as shown in Figure 16) based on which the files to be monitored could be filtered.

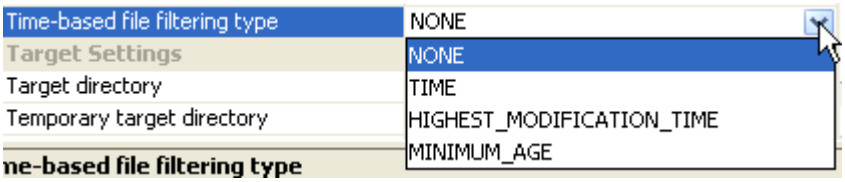


Figure 16: Time-based file filtering types

The behavior is as follows:

- **NONE:** No filtering is applied on the files. Every file present in the Source directory is monitored.

- **TIME:** Files whose last modification time is greater than the last polling time is monitored. This ensures that only the files modified/added after the last polling cycle are monitored.

Time-based file filtering type	**TIME**
Base Time	01:01:1970 00:00
Remote host time offset	+00:00

Figure 17: Time-based file filtering type - TIME

  - **Base Time:** Base time in **dd:MM:yyyy hh:mm** format after which the changed files are to be uploaded
  - **Remote host time offset:** If the FTP server and the component are not in the same time zone, the difference in the time zone of FTP Server time zone from the component's time zone should be specified in (+/-) hh:mm format.

- **HIGHEST_MODIFICATION_TIME:** Files whose last modification time is greater than the highest last modification time found in the last polling cycle is monitored. This ensures that only files which are newer than the newer file already polled are selected.
- **MINIMUM_AGE:** Files whose last modification time is less than the current polling time minus the age is monitored. This ensures that the file modification time is at least Minimum Age earlier than the current time.

Time-based file filtering type	**MINIMUM_AGE**
Minimum age	0 milli seconds
Remote host time offset	+00:00

Figure 18: Time-based file filtering type – MINIMUM AGE

  - **Minimum age**: The minimum age of the files which are to be monitored.

    Note: This property is visible only when the property Time-based file filtering type is set to **MINIMUM_AGE**.
  - **Remote host time offset:** If the FTP server and the component are not in the same time zone, the difference in the time zone of FTP Server time zone from the component's time zone should be specified in (+/-)hh:mm format.

Note: This property is visible only when Time-based file filtering type is set to TIME/HIGHEST_MODIFICATION_TIME/MINIMUM_AGE.

Example: If the source directory contains 4 files named a.txt, b.txt, c.txt and d.txt. The polling interval is 3 min and the first poll is going to start at 11:00:00 (This polling settings can be configure in Scheduler Configuration Panel, please refer the section **Scheduler Configuration Panel** for more details)

At first poll all files will be monitored irrespective of the value of the property "**Time-based file filtering type".**

- If the files have last modification time like:
  - a.txt 11:00:16
  - b.txt 11:00:30
  - c.txt 11:02:10
  - d.txt 11:02:50

- If the property "**Time-based file filtering type**" is set to TIME, then all the files will be monitored in the next poll (which is going to poll at 11:03:00), since all files are modified after the last poll.
- If the property "**Time-based file filtering type**" is set to HIGHEST_MODIFICATION_TIME, then also all files will be monitored in the next poll (which is going to poll at 11:03:00), since all files are having the last modification time greater than the highest last modification time found on last poll (Component will keep the track of highest last modification time found in the poll).
- If the property "**Time-based file filtering type**" is set to MINIMUM_AGE and Minimum age is set to 5 min, then no files will be monitored in the next poll (which is going to poll at 11:03:00). Files a.txt and b.txt will be monitored in the polling which will be going to poll at 11:06:00, and the files c.txt and d.txt will be monitored in the polling which will be going to poll at 11:09:00, since the files monitored in the particular polling have been modified at least 5 min ago from the polling time.

4.1.3.2.3 Target Settings

- **Target directory:** Directory on the FTP server to which the file(s) is/are to be transferred. Note that this property allows relative paths which would be computed relative to the directory specified for Parent Directory on the local system.
- **Use Temporary target Directory:** If this property is set to true, then the FTPPut adapter will use a temporary target directory for intermediate processing. If you do not prefer to create an extra directory in the FTP server, you can set this property to **No**.
- **Temporary target directory:** This property is visible only when the property **Use temporary target Directory** is set to **Yes**. Directory on the ftp server which the FTPPut component uses for intermediate processing during file downloads.

Note: This directory should not be same as Target directory.

Auto Create Target Directory

Set this property to create target directory. If this is not set, the directory has to be created explicitly by the user.

Action if same file exists

Action that must be taken if the target directory already has a file with name same as the file that is to be uploaded. The behavior will be dependent on the selection as shown below:

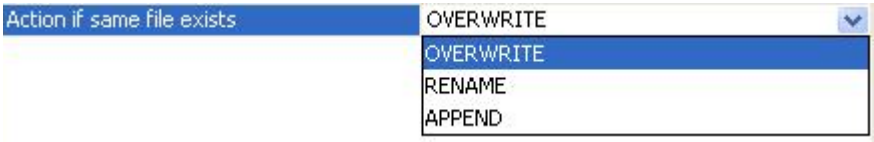


Figure 19: Action if same file exists

- **OVERWRITE:** The file to be uploaded overwrites the one in the target directory.
- **RENAME:** The name of the file that is being uploaded is changed based on the properties given below:

Action if same file exists	RENAME
Append date-time format postfix	MMddyyHHmmss
Append counter	yes

Figure 20: Appender counter

- **Append date-time format postfix:** When existing files in the Target directory are not to be overwritten, FTPPut provides the flexibility of uploading the content into a new file whose name is in the format <NameOfExistingFile_CurrentDateTime>. The format in which the date and time is to be appended should be specified as a value for this property.

  Example: If the date-time format is specified as MMddyyyyHHmmssss for the file Sample.txt, the target file created would be Sample_0305200811300013.txt.

- **Append counter:** Enabling this property appends a counter along with the Date and Time to target file name, when the target file is not to be overwritten.

  Example: A sample file name could be Sample_0305200811300013_0.txt.

- **APPEND:** The file being uploaded gets appended to the one in the target directory.

  Note: The same behavior is reflected in the processed directory when **Move to working directory** is set to Yes.

4.1.3.2.5 Miscellaneous Settings

Auto create temporary directories

The temporary directories created for processing of the component are created based on this property. If this is not set then the temporary directories must be created explicitly.

Note: When the user does not have listing permissions on the server, it is recommended that this property is not set and to create directories manually.

4.1.3.3 Input and Output

The input and output structures depend the configuration of property **Request type**.

When **Request type** is set to **File**, input and output structures are defined as shown in Figure 21 and Figure 22 respectively.

<ns1:FTPRequest>	
<LocalPath>	#string
<RemoteFile>?	#string
<Append>?	
@value	true | false
<TransferType>?	
@value	Ascii | Binary

Figure 21: Input schema structure for the request type - File

Table 1: Input schema element descriptions for - File request type

Schema Element	Description

LocalPath	Path of the local file which is to be uploaded
RemoteFile	File on the FTP server to which the data is to be written
Append	Whether to append data if the file already exists
TransferType	Type of data transfer (ASCII or Binary)

Figure 22: Output schema structure for the request type - File

Table 2: Output schema element descriptions for - File request type

Schema Element	**Description**
LocalPath	Path of the local file which was transferred to the FTP server
RemoteFile	File on the FTP server to which the data has been written
Append	Append value mentioned in the input
TransferType	TransferType mentioned in the input
BytesTransferred	The number of bytes transferred.
TotalBytes	The total number of bytes transferred
ReplyCode	The reply code sent by the FTP server
ReplyText	The reply text sent by the FTP server

When the type of input is **Data**, data to be transferred is provided under the **Data** element in the input message and the data will be written to a file with the name specified under the element **RemoteFile**.

If text data has to be transferred, then the text content should provided under the **Data** element in the input message and the attribute **dataType** should be set to **Text**.

If binary data has to be transferred, then the **base64** encoded string created from binary data should be provided under the **Data** element in the input message and the attribute **dataType** should be set to **Text**.

Figure 23 and 24 show the input and output schema structures respectively, when the request type is **Data**. The only difference in these schema structures against the ones for **File** request type is the replacement of the schema element **LocalPath** with **Data**. Please refer Table 1 and 2 for the remaining schema elements.

Note: Input port appears only when Monitoring is disabled (Value of the property **Monitor** set to **No**).

<ns1:FTPRequest>	
<Data>	
@dataType	Bytes | Text
#string	
<RemoteFile>	#string
<Append>?	
@value	true | false
<TransferType>?	
@value	Binary | Ascii

Figure 23: Input schema structure for the request type - Data

<ns1:FTPResponse>	
<ns1:Data>	
@dataType	Text | Bytes
#string	
<ns1:RemoteFile>	#string
<ns1:Append>?	
@value	true | false
<ns1:TransferType>	
@value	Ascii | Binary
<ns1:BytesTransferred>?	#string
<ns1:TotalBytes>?	#string
<ns1:ReplyCode>	#string
<ns1:ReplyText>	#string

Figure 24: Output schema structure for the request type – Data

When the property Use Connection details from input is chosen, an additional element ConnectionFactorySettings is added to the input schema, as shown in the figure. Properties that are used to create the connection are present under this element.

XML Item	Value Type
<ns1:FTPRequest>	
<ConnectionFactorySettings>?	
<ConnectionProperties>?	
<Protocol>?	FTP | SFTP
<RemoteHost>?	#string
<Port>?	#int
[Choice]	
<FTPProperties>?	
<Login>?	#string
<Password>?	#string
<SFTPProperties>?	
<ClientAuthenticationtype>?	Password | Public Key | Both
<Login>?	#string
<Password>?	#string
<PrivateKeyFile>?	#string
<KeyType>?	DSA | RSA
<KeyFilePassword>?	#string
<AdvancedProperties>?	
<Timeout>?	#int
<CurrentDir>?	#string
<ConnectMode>?	Active | Passive
<TransferType>?	Ascii | Binary
<FileExtensionsTobeFiltered>?	#string
<DebugResponses>?	true | false
<SiteCommandParameters>?	#string
<ParsingFormatForDirectoryPath>?	
@Enable	true | false
#string	
<SSLSettings>?	
<TrustStoreLocation>?	#string

Figure 25: Input schema with ConnectionFactorySettings

4.1.3.4 Testing the Interaction Configurations

The configuration can be tested by sending a test file when you click on the Test option in the interaction properties panel.

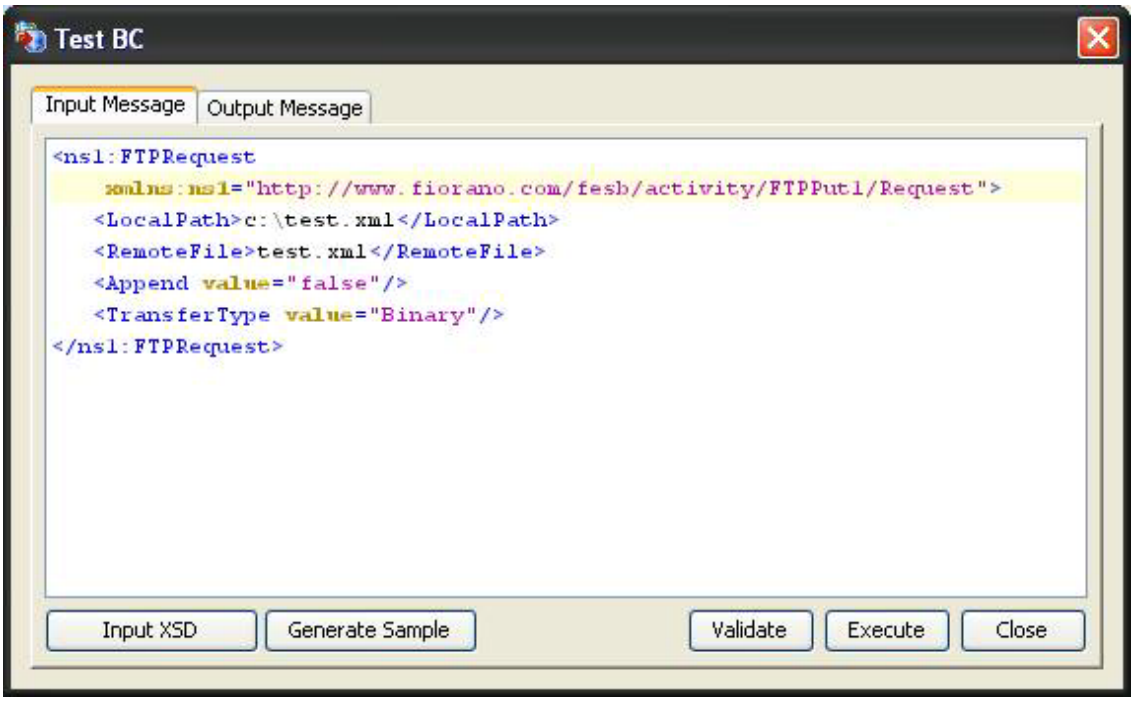


Figure 26: Sample input sent from CPS

The FTP server can be configured in the connection properties panel of CPS.

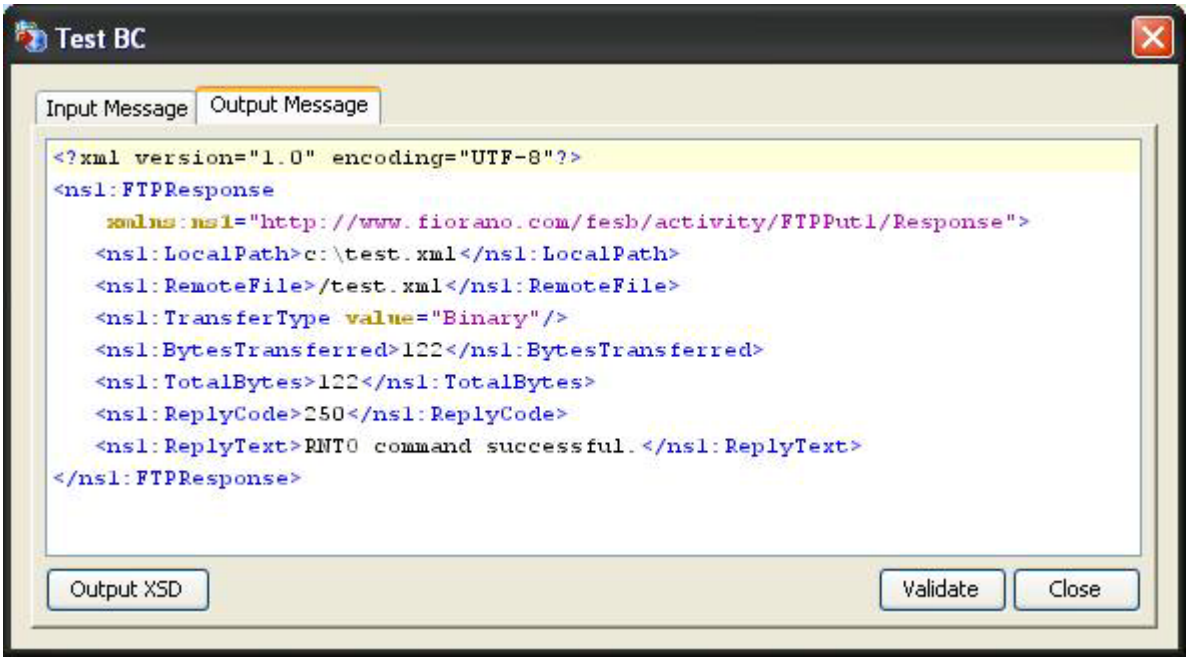


Figure 27: Sample output

4.1.3.5 Functional Demonstration

4.1.3.5.1 Scenario 1

Send files from a local directory to the FTP server's remote directory.

Configure the FTP Put as described in *Configuration and Testing* section and use feeder and display components to send sample input and check the response respectively. In the interaction configuration choose the option **File** for property **Request type.**

Figure 28: Event Process demonstrating Scenario 1

Sample Input

```
<ns1:FTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/FTPPut1/Request">
    <LocalPath>c:\test.xml</LocalPath>
    <RemoteFile>test.xml</RemoteFile>
    <Append value="false"/>
    <TransferType value="Binary"/>
</ns1:FTPRequest>
```

Figure 29: Input sample when Request type is File.

Sample Output

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:FTPResponse xmlns:ns1="http://www.fiorano.com/fesb/activity/FTPPut1/Response">
    <ns1:LocalPath>c:\test.xml</ns1:LocalPath>
    <ns1:RemoteFile>/test.xml</ns1:RemoteFile>
    <ns1:TransferType value="Binary"/>
    <ns1:BytesTransferred>122</ns1:BytesTransferred>
    <ns1:TotalBytes>122</ns1:TotalBytes>
    <ns1:ReplyCode>250</ns1:ReplyCode>
    <ns1:ReplyText>RNTO command successful.</ns1:ReplyText>
</ns1:FTPResponse>
```

Figure 30: Output message received for input shown in Figure 28.

4.1.3.5.2 Scenario 2

Send data to the FTP Server and save it as a file in the remote directory

Configure the FTP Put as described in **Configuration and Testing** section and use feeder and display component to send sample input and check the response respectively. In the interaction configuration choose the option **Data** for the property **Request type**.

Sample Input

```
<ns1:FTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/FTPPutl/Request">
   <Data dataType="Text">This is the data to be written into the file</Data>
   <RemoteFile>sample.txt</RemoteFile>
   <Append value="false"/>
   <TransferType value="Ascii"/>
</ns1:FTPRequest>
```

Figure 31: Input sample when Request type is Data

Sample Output

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:FTPResponse xmlns:ns1="http://www.fiorano.com/fesb/activity/FTPPutl/Response">
   <ns1:Data dataType="Text">This is the data to be written into the file</ns1:Data>
   <ns1:RemoteFile>/sample.txt</ns1:RemoteFile>
   <ns1:TransferType value="Ascii"/>
   <ns1:BytesTransferred>44</ns1:BytesTransferred>
   <ns1:TotalBytes>44</ns1:TotalBytes>
   <ns1:ReplyCode>250</ns1:ReplyCode>
   <ns1:ReplyText>RNTO command successful.</ns1:ReplyText>
</ns1:FTPResponse>
```

Figure 32: Output message received for input shown in Figure 30.

4.1.3.5.3 Scenario 3 (Scenario 1 Using SFTP Protocol)

The following steps give a brief description about server settings and SFTP protocol. Here, we provide steps to test FTPPut adapter using **vsftpd** server which is installed in Linux.

Steps to produce:

Install **vsftpd** server by executing the following command on command prompt:

yum install vsftpd

Generate keys pairs (both RSA and DSA) and store in ~/.ssh/ directory. For key generation we can use the following commands:

ssh-keygen -t rsa (for RSA type key generation)

ssh-keygen -t dsa (for DSA type key generation)

Note: While executing the above commands, file name and password for the file has to be provided. If the file name provided is id_rsa, then two files will be generated named id_rsa.pub and id_rsa which are public key and private key respectively.

Next, install the public keys in server. Private key has to be with the client which is used to login. The server will authenticate the private key using its public key. This type of client-authentication is called **Public Key Authentication**. The installation of the public key in the server can be done by the executing following command

ssh-copy-id -i ~/.ssh/id_rsa.pub root@localhost

~/.ssh/id_rsa.pub is rsa public key file path and **root@localhost** is the server (here we are using the same machine).

Now, change the following in **sshd_config** file (this file can be located in etc/ssh/ folder). Set **RSAAuthentication** or **DSAAuthentication** to yes based on the key file type used.

Add the following line IdentityFile ~/.ssh/id_rsa in the **ssh_config** file which was located in /etc/ssh. (if already Identity File is set to some other file, then it has to be modified)

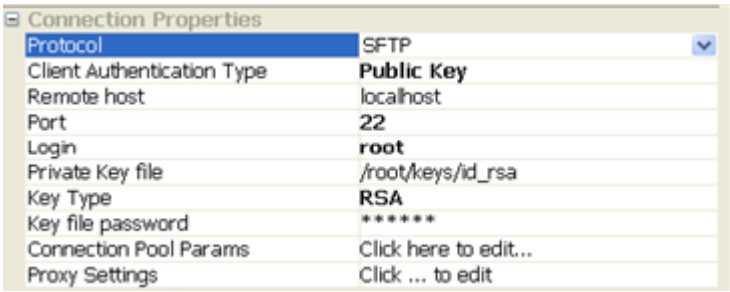
Now, restart the servers using the following commands.

/etc/rc.d/init.d/sshd restart
/etc/rc.d/init.d/vsftpd restart

Now the server is ready to accept SFTP protocol to login.

4.1.3.6 Configuring FTPPut Component

The configuration of the connection properties of FTPPut component for SFTP protocol is shown in the Figure 32, and the remaining procedures are same as explained in Scenario 1.



Connection Properties	
Protocol	SFTP
Client Authentication Type	**Public Key**
Remote host	localhost
Port	**22**
Login	**root**
Private Key file	/root/keys/id_rsa
Key Type	**RSA**
Key file password	******
Connection Pool Params	Click here to edit...
Proxy Settings	Click ... to edit

Figure 33: Protocol type selection

4.1.3.7 Use Case Scenario

In the revenue control packet example error messages are sent to a FTP server and are stored there for tracking using the FTP Put component.

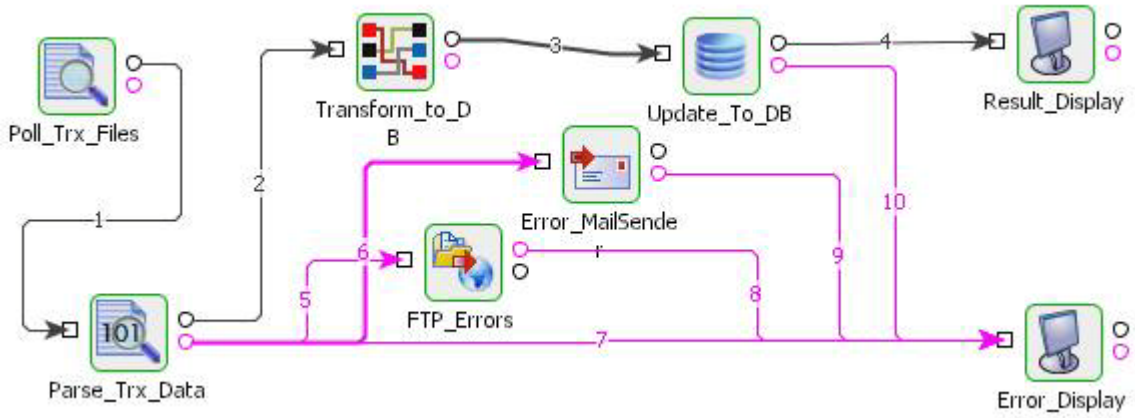


Figure 34: Demonstration scenario

The event process demonstrating this scenario is bundled with the installer. The bundled process shows it as a File Writer component instead of a FTP Put component.

Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Studio.

4.1.3.8 Useful Tips

- Some FTP Servers have an idle timeout limit for the client connections configured at server side. When a client connection is idle beyond this time limit, server closes the connection. So a new connection has to be made, if a request is made after being idle for time greater than this idle timeout. This means, if the server has a idle timeout value set to 120s and the component does not receive a message at any point of time for more than 120s, the connection would be closed.

  There are multiple solutions to this problem -

  - If the component has to wait for more than 120 seconds before a new message is delivered to the component, then disabling connection pool is a better option.
  - If the component has to wait for more than 120 seconds in very few cases before a new message is delivered to the component, then enabling reconnection is a better option.

  Increase the timeout at server side to a higher value, if possible.

- If error with error code 550 occurs during request execution, the component logs the message as resource warning but not as error. To get these messages as errors, the property "Throw fault on warnings" must be selected for request processing error in error configuration panel

4.1.4 IWay

The iWay component may be used to utilize the adapter library of the iWay Adaptive Framework for service oriented architecture (SOA).

You need to deploy a web application (iWayHelper.ear) on the application server on which the iWay JCA resource adapter to be used has been deployed.

The iWay component of Fiorano requests the above mentioned web application at design time to retrieve iWay configuration parameters such as available adapter names, target names, input/output schema of the available targets.

Note: iWay adapter requests are executed through the deployed iWayHelper in the application server.

4.1.4.1 Configuration and Testing

The connection parameters can be configured in the connection properties panel of CPS.

Figure 4.54: Sample IWay connection configuration

4.1.4.1.1 Using Named Configurations

Changes are done in eStudio wrt to Named Configurations. With the implementation of Named Configurations, Component Configuration panel is changed as shown in Figure below.

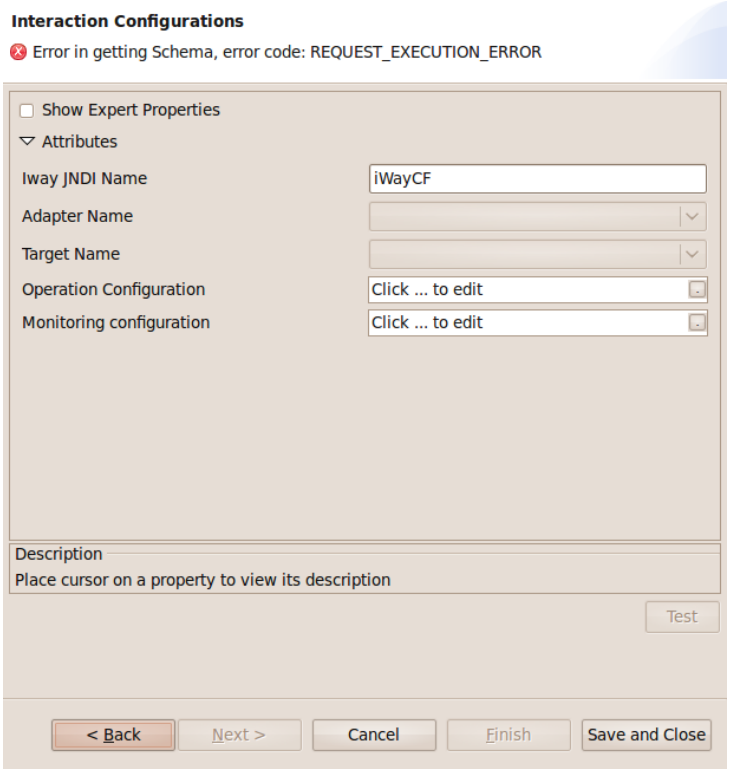


Figure: Modified Component Configuration panel

Some properties are grouped as one property in the modified configuration panel. Iway Operation Location, Deployment Mode and Iway HelperURL which were present previously in the Interaction Configurations panel are grouped under the name Operation Configuration.

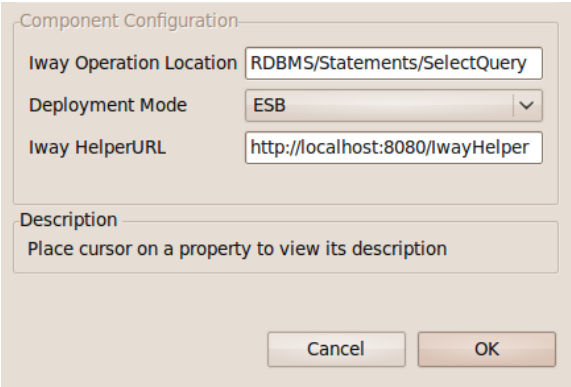


Figure: Operation Configuration

Server connection can be tested from within the CPS by clicking on **test** in the connection properties panel.

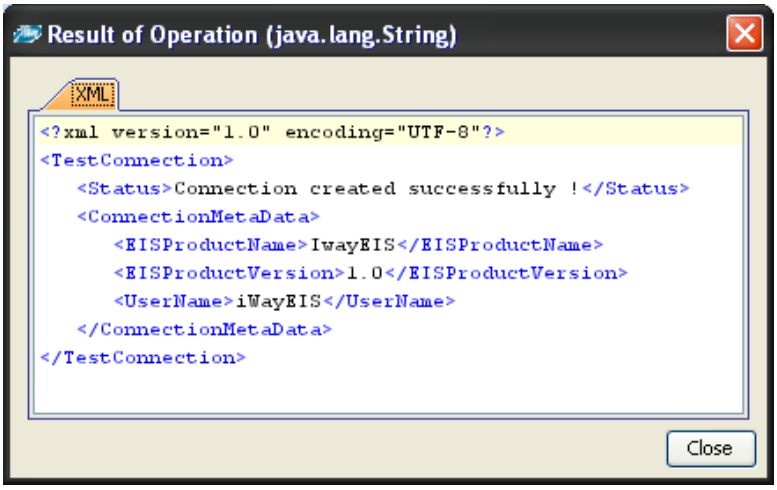


Figure 4.55: Sample connection test result indicating success

IWay adapter related configuration can be captured in the interaction configuration panel

Attributes	
Iway JNDI Name	iWayCF
Adapter Name	RDBMS
Target Name	OracleTest
Iway Operation Location	RDBMS/Statements/SelectQuery
Iway HelperURL	http://localhost:8080/IwayHelper

Figure 4.56: IWay adapter related configuration

4.1.4.2 Input Schema

There is no input schema for this adapter.

4.1.4.3 Output Schema

There is no output schema for this adapter.

4.1.4.4 Functional Demonstration

This requires a licensed IWay adapter.

4.1.4.5 Use Case Scenario

In the purchasing system example the record purchase details are sent to an external inventory management system server for processing using an IWay adapter.

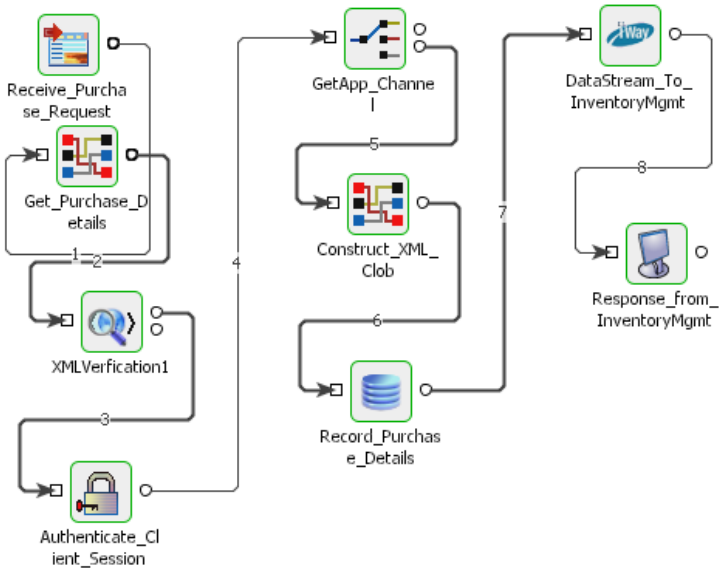


Figure 4.57: Purchasing system example

The event process demonstrating this scenario is bundled with the installer. The bundled process shows it as a HTTP component instead of an IWay component.

Documentation of the scenario and instructions to run the flow can be found in the Help tab of flow when open in Studio.

4.1.4.6 Useful Tips

- IWay adapter requests are executed through the deployed IWayHelper in the application server.

4.1.5 POP3

POP3 component is used to connect to a mail server/News server and retrieve emails/news articles using POP3/IMAP/NNTP protocol. It uses the JavaMail API.

The component supports these functions:

- Retrieve emails/news articles from the mail/news server.
- Fetch the email/news count on the mail/news server.
- Verify signature, decrypt mail and MDN.

4.1.5.1 Managed Connection Factory

The following properties can be configured in the Managed Connection Properties panel of CPS.

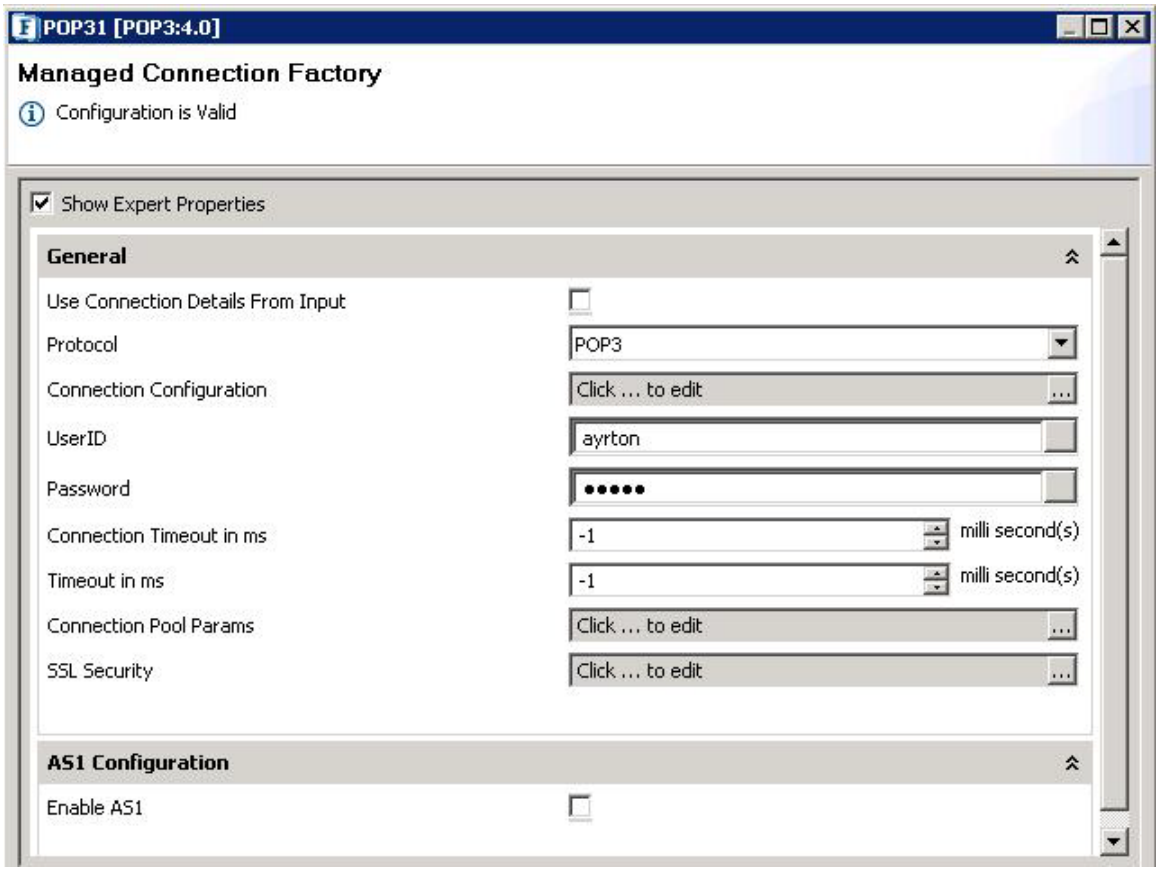


Figure 1: Managed Connection Panel of POP3

4.1.5.1.1 Attributes

- **Protocol**

  Specifies **the** protocol used to retrieve emails.

  - **POP3** – POP3 protocol is used to retrieve emails. The property **Folder to pick mails** is not visible if this value is selected.

  - **IMAP** – IMAP protocol is used to retrieve emails. The property **Folder to pick mails** is visible if this value is selected.
  - **NNTP** – NNTP protocol is used to retrieve news articles. The property **Newsgroup to pick news articles** is visible if this value is selected.
- **MailserverURL**

  Specifies URL at which the mail server is hosted.
- **MailServer Port**

  Specifies port number on which the mail server accepts POP3/IMAP/NNTP connections**.** Default port for NNTP is 119 and IMAP is 143.
- **UserID**

  User ID or Login name used to connect to the mail server. The user must have an email account with the server specified by the property **MailserverURL**.
- **Password**

  Password for the user specified by the property **UserID**.
- **Folder to pick mails**

  Specifies folder name from which emails are retrieved. This option is displayed only when the property **Protocol** is set to IMAP
- **Newsgroup to pick articles**

  Specifies Newsgroup name from which news articles are retrieved. This option is displayed only when the property **protocol** is set to **NNTP.**
- **Connection Timeout in ms**

  The duration in milliseconds for which the component waits to connect to the mail server (Socket **connection timeout** value in milliseconds). If the component fails to create a connection in the specified interval, then the retry behavior will depend on the configuration of **Error Configuration Panel**.
- **Timeout in ms**

  Socket **I/O timeout** value in milliseconds. If any operation requires more than the specified time then the operation will fail.

AS1 Configuration

- **Enable AS1:** Select this option to enable AS1 and provide signing, encryption and MDN configuration by clicking the ellipsis button of AS1 SSL Configuration property.

Signature Configuration

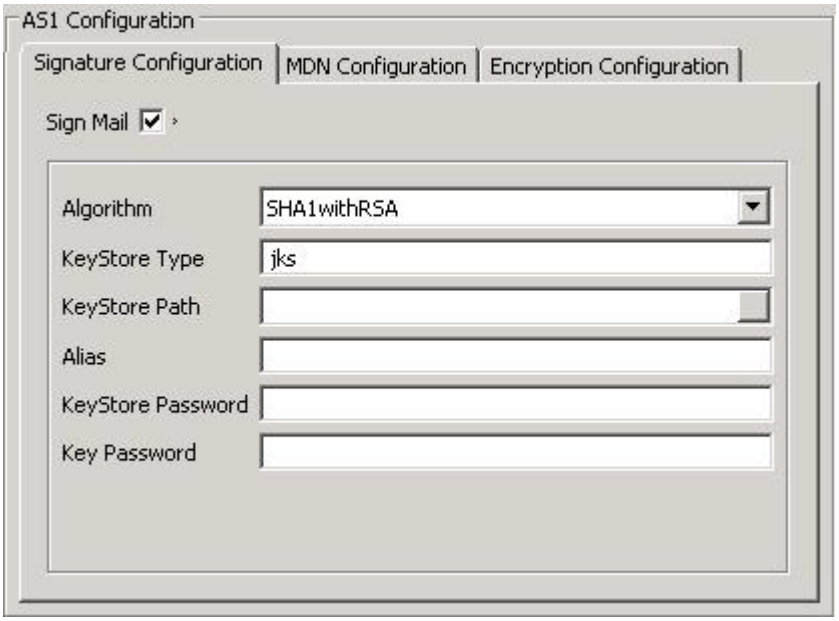


Figure: Signature configuration details

- **Sign Mail:** Select to sign mail and provide algorithm and key store details to be used for signing.

MDN configuration

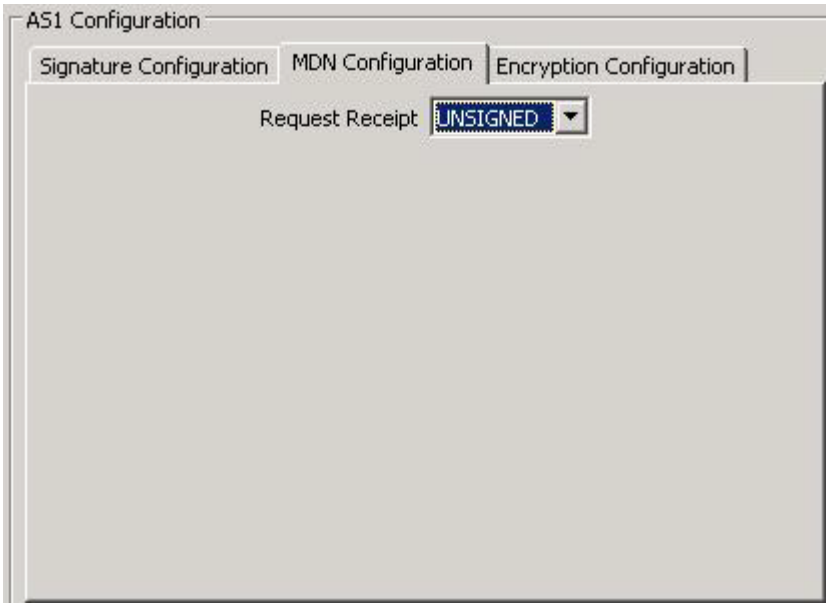


Figure: MDN configuration details

- **Request Receipt:** Choose whether MDN should be signed or unsigned.

Encryption Configuration

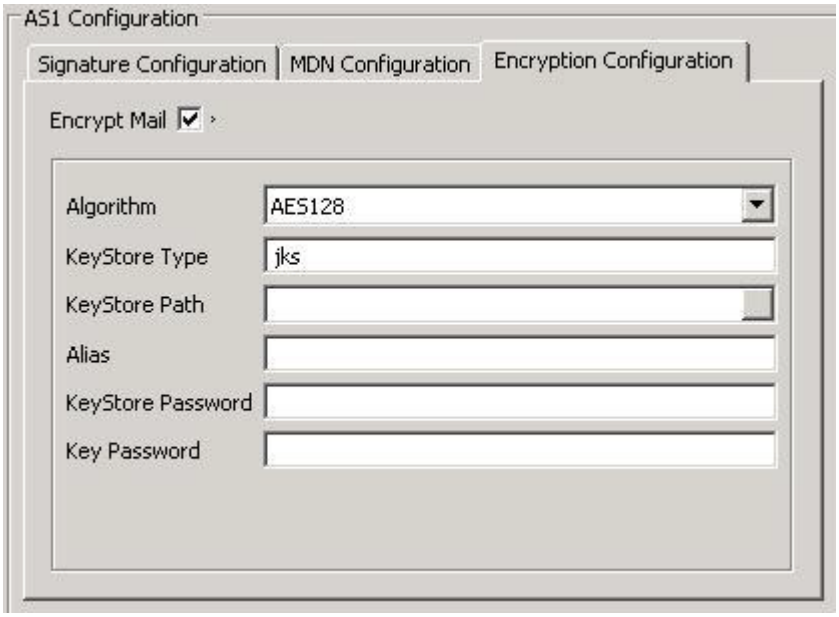


Figure: Encryption configuration details

- **Encrypt Mail:** When this option is selected, mail will be encrypted with the provided algorithm and Keystore details.

Server connection can be tested from within the CPS by clicking the **Test** button in the Connection Properties panel.

4.1.5.2 Interaction Configuration

POP3 supports various options to retrieve mails from the server. The way the component interacts with the server can be configured in the **Interaction Configurations** panel.

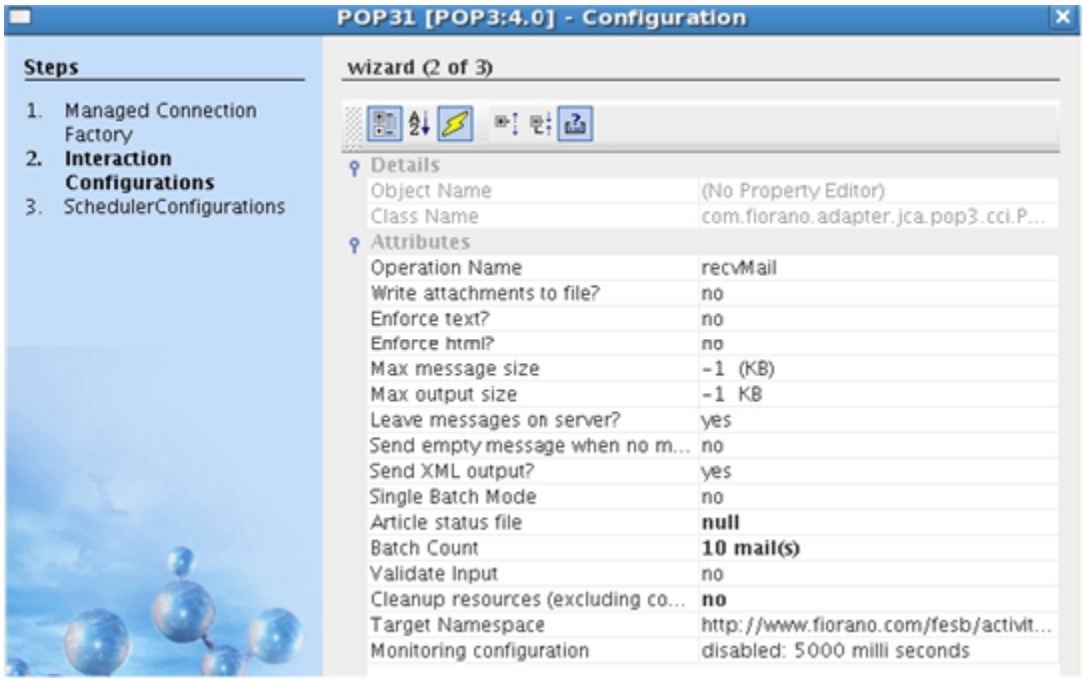


Figure 2: Interaction Configuration panel

4.1.5.2.1 Attributes

Operation Name

The operation that should be done after connecting to the server is specified here.

- **recvMail** - retrieves and displays all the content in the email. If this operation is selected, the following properties are visible – **Write attachments to file?**, **Enforce Text?**, **Enforce html?**, **Max message size**, **Max output size**, **Leave messages on server?**, **Send empty message when no other mails are present**, **Send XML output?, Article status file**
- **mailCount** - retrieves the total number of mails in the account specified. If this operation is selected, the following properties are not visible – **Write attachments to file?**, **Enforce Text?**, **Enforce html?**, **Max message size**, **Max output size**, **Leave messages on server?**, **Send empty message when no other mails are present**, **Send XML output?, Article status file**

Write attachments to file?

Specifies whether the attachments in the mail are to be written to a file.

- yes – Attachments in retrieved email are written to files in the folder configured for property Attachments folder. Property **Attachments folder** is visible when this option is selected. **savedToFile** attribute of **Attachment** element in the output message is set to **yes** and the file name is added as content for **Attachment** element.
- **no** – Each attachment in retrieved email is converted to a **base64** encoded string and added as content for **Attachment** element in output XML. Property **Attachments folder** is not visible when this option is selected. **savedToFile** attribute of **Attachment** element in the output message is set to **no**.

Attachments folder

Specifies the path of the folder where attachments are saved. This property is visible only when the property **Write attachments to file?** is set to yes. The folder should exist in the machine on which the component is running.

- If the attachment in the mail was loaded from a file, then the same file name is used to save attachment to the attachments folder.
- If the attachment in the mail was not loaded from a file and the attachment contains a description, then the description text is used a file name to save attachment to the attachments folder.
- If both the conditions are not met or the attachments folder is not present in the system or if there is any file system dependent error while saving , the attachment is not saved and resource warning is raised. For more information on resource warning, please refer to section Error Handling.
- If the folder already contains a file with same name that is written by this component using the same connection that processed the current request then the file name is appended with time-stamp, else the file is over written.

Note: The folder path provided is not validated in the CPS.

Enforce Text?

This option is used when the property **Send XML output?** is set to **yes** and the email does not contain any text content. When this option is set to yes, html content in the email is converted to simple text. This property is visible only if property **Send XML output?** is set to **yes**.

Example: Each <BR> element is replaced with a new line character.

Enforce html?

This option is used when the property **Send XML output?** is set to **yes** and the email does not contain any html content. When this option is set to yes, text content in the email is converted to HTML. This property is visible only if property **Send XML output?** is set to **yes**.

Example: If the mail does not contain html content and the property **Enforce html?** Is set to no then the output message XML does not contains element HtmlBody as shown in Figure 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:Emails xmlns:ns1="http://www.fiorano.com/fesb/activity/POP31/Out">
   <ns1:Email>
      <ns1:To>poorna.chand@in.fiorano.com</ns1:To>
      <ns1:From>poorna &lt;poorna.chand@in.fiorano.com&gt;</ns1:From>
      <ns1:Subject>Test Enforce html</ns1:Subject>
      <ns1:Body>
         <ns1:TextBody>This is to verify "Enforce html?" property&#xD;
&#xD;
</ns1:TextBody>
      </ns1:Body>
      <ns1:Headers>
         <ns1:Header name="Content-Transfer-Encoding" value="7bit"/>
```

Figure 3: Output message of retrieval of a mail which has no html body when Enforce Html? is set to no

When **Enforce html?** Is set to yes, then the existing text body will be converted to html and it will be included in output XML as HtmlBody. The behavior is depicted in Figure 4. New line characters (/n or &#xD) in text content are converted to break line symbols (<BR>) in html content.

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:Emails xmlns:ns1="http://www.fiorano.com/fesb/activity/POP31/Out">
   <ns1:Email>
      <ns1:To>poorna.chand@in.fiorano.com</ns1:To>
      <ns1:From>poorna &lt;poorna.chand@in.fiorano.com&gt;</ns1:From>
      <ns1:Subject>Test Enforce html</ns1:Subject>
      <ns1:Body>
         <ns1:TextBody>This is to verify "Enforce html?" property&#xD;
&#xD;
</ns1:TextBody>
         <ns1:HtmlBody>This is to verify "Enforce html?" property&lt;BR&gt;&lt;BR&gt;</ns1:HtmlBody>
      </ns1:Body>
      <ns1:Headers>
         <ns1:Header name="Content-Transfer-Encoding" value="7bit"/>
```

Figure 4: Output message of retrieval of a mail which has no html body when Enforce Html? is set to yes

Max message size

Specifies the maximum size limit on an email that is sent in the output. E-mails whose size is larger than the value specified for this property is not sent.

Max output size

Specifies the maximum limit on the combined size of emails that are included in the output message. Since more than one email can be retrieved in a message, this option allows control on the total size of all mails that may be returned in a single response.

Note: If the property Single Batch Mode set to No and Batch Count set to 1, then the properties Max Message size and Max output size will behave similarly.

Example: If mail server contains 10 mails each of size 10Kb and the property Max output size set to 30kb then, if the component receives a request to retrieve all the 10 mails then the component sends four messages to the output port.

- First message --- contains first, second and third mails
- Second message --- contains fourth, fifth and sixth mails
- Third message --- contains seventh, eighth and ninth mails
- Fourth message --- contains tenth mail

Leave Messages on Server

Specifies if the emails retrieved by this component are removed from the mail server or not.

- **yes** – The emails retrieved are not deleted from the mail server and can be seen and fetched using other email clients.
- **no** – The emails retrieved are deleted from the mail server as soon as the email is retrieved by the component.

Warning:

- If connection pooling is not enabled and this property is set to yes, then emails are repeatedly retrieved with each request as the mails are not deleted from the mail server.
- If connection pooling is enabled and the component is running in multi-threaded mode, then each thread will retrieve all the mails.
- If connection pooling is enabled and a connection error happens, then all the mails are retrieved again when a new connection is made.

Send Empty message when no mails are present

If this property is set to yes, then an empty message is sent out for each request when there are no mails on the mail server, else no message is sent out.

Send XML output?

Species the format the output message of the component.

- **yes** – email(s) are formatted as XML text containing **To**, **CC**, **From**, **Subject**, **TextBody**, **HtmlBody**, **Headers** and **Attachments** elements with values from the email. When this property is selected, property **Send as HTML, if content has HTML** is not visible and properties **Enforce text?**, **Enforce html?**, **Batch Count** and **Single Batch Mode** are visible.
- **no** – text content or HTML content is extracted and sent as output message. In case of multi-part message, only the one of the parts is sent as output. When this value is selected, properties **Enforce text?**, **Enforce html?**, **Batch Count** and **Single Batch Mode** are not visible and property **Send as HTML, if content has HTML** is visible. Value for property **Batch Count** is set to 1 and value for property **Single Batch Mode** is set to **no**.

Send as HTML, If Content has HTML

Specifies whether HTML content or plain text content is sent in the output.

- **yes** – If HTML content is present in email, then HTML content is sent in the output. If HTML content is not present but plain text content is present, then plain text content is sent in the output.
- **no** – If plain text content is present in email, then plain text content is sent in the output. If plain text content is not present but HTML content is present, then HTML content is sent in the output.

If set to yes**,** then the html content will be preferred to be sent in output, if exists. If set to no then text/plain content will be preferred to be sent in output, if exists. This property is visible only if **Send XML output?** is set to no.

Single Batch Mode

This option is used to specify whether the component should send each mail in a separate message or in a single XML message.

- When this option is set to yes, all emails retrieved are combined into a single XML message and sent to the output.
- When this is set to **n**o and value for property **Batch size** is set to 1, each mail is sent as a separate message.
- When this is set to **n**o and value for property **Batch Size** is set to n – where n is any positive number, n mails are combined into a single XML and sent as a separate message.
- This property is only visible when property **Send XML output?** is set to yes. When property **Send XML output?** is set to no value for this property is set to **no**.

Batch Count

This option specifies the number of messages to be combined into a single message. This property is used only if value for property **Single Batch Mode** is set to **no**. This property is only visible when property **Send XML output?** is set to yes. When property **Send XML output?** is set to no value for this property is set to **1**.

4.1.5.3 Sample Input and Output

For **recvMail** operation, the input message contains a MessageCount attribute which takes an integer value.

- If the message count provided in sample input is less than or equal -1, then it fetches all of the mails from the server.
- If the Message Count provided is greater than -1, then the minimum of the Message Count provided and then actual number of messages present in the mail box is computed and fetches that number of mails.

Input Message | Output Message

```
<ns1:Input xmlns:ns1="http://www.fiorano.com/fesb/activity/POP31/In">
   <MessageCount>2092129998</MessageCount>
</ns1:Input>
```

Figure 5: Sample Input

Input Message | Output Message

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:Emails xmlns:ns1="http://www.fiorano.com/fesb/activity/POP31/Out">
   <ns1:Email>
      <ns1:To>Internal Gestioni Mailer &lt;ayrton@fiorano.com&gt;</ns1:To>
      <ns1:From>"Ed Narayanan (Fiorano Software)" &lt;ed.narayanan@fiorano.com&gt;</n
      <ns1:Subject>Re: Order 2456 has been received.</ns1:Subject>
      <ns1:Body>
         <ns1:TextBody>Accepted&#xD;
&#xD;
</ns1:TextBody>
      </ns1:Body>
      <ns1:Headers>
         <ns1:Header name="Content-Transfer-Encoding" value="7bit"/>
         <ns1:Header name="Message-ID" value="&lt;4652F8F4.1090107@fiorano.com&gt;"/>
         <ns1:Header name="Subject" value="Re: Order 2456 has been received."/>
```

Figure 6: Sample Output

4.1.5.3.1 Input Schema

No input and Output schema are generated for the **Mail Count** operation. For **Receive Mail** operation, the input schema contains the following elements.

Schema Element	Description
<MessageCount>	Number of messages to be fetched. If the Message Count provided in sample input is less than -1, then it searches in the mailbox for total number of messages and fetches all of them. If the Message Count provided is greater than -1, then the minimum of the Message Count provided and then actual number of messages present in the mail box is computed and fetches those numbers of mails.

4.1.5.3.2 Output Schema

For Receive Mail operation, the output schema looks like:

Schema Element	Description
<Emails>	Emails retrieved from mail box
<Email>	Email
<To>	Email address of the recipient(s)
<From>	Email address of the sender

Schema Element	Description
<CC>	Email address of recipient(s) to be copied in the mail
<Subject>	Subject of the mail
<Body>	Body of the mail
	Text Body
	HTML body
<Attachments>	Attachments Attachment name in the email Boolean to specify whether attachment is saved.
<Headers>	Headers in Email Header name Header value

4.1.5.4 Functional Demonstration

4.1.5.4.1 Scenario 1

This scenario demonstrates the retrieving of e-mails with attachments.

Configure POP3 as described in section Configuration and Testing for Receive Mail and select the attachment folder in the Interaction Configuration panel to save the attachments, if any. Use Feeder and Display components to send sample input and check the response respectively.

Note: Receive Mail operation receives each mail as a separate message in this case.

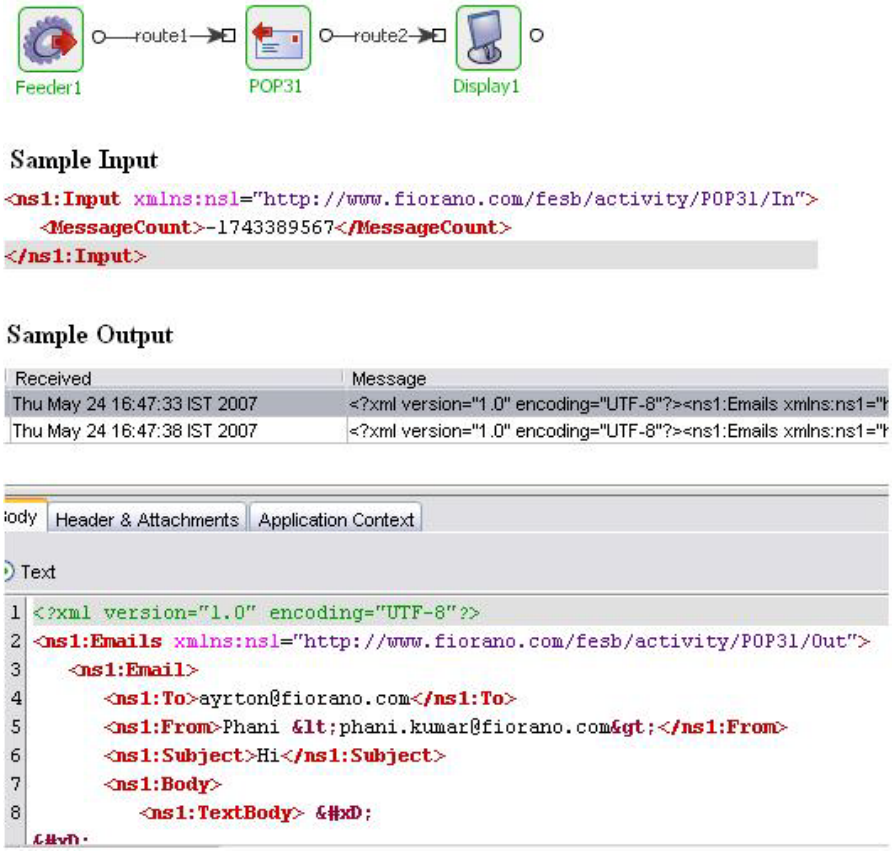
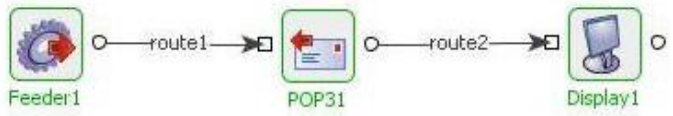


Figure 8: Demonstrating scenario 1 with sample input and output

4.1.5.4.2 Scenario 2

This scenario demonstrates the retrieval of Mail count.

Configure POP3 as described in section Configuration and Testing for **Mail Count** operation and use **Feeder** and **Display** components to send sample input and check the response respectively.



Sample Input

test message

Sample Output

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:TestOutput xmlns:ns1="http://www.fiorano.com/fesb/activity/POP31/Out">
   <ns1:MessagesInServer>3</ns1:MessagesInServer>
</ns1:TestOutput>
```

Figure 9: Demonstrating scenario 2 with sample input and sample output

4.1.5.5 Useful Tips

- The component runs on the Peer Server and therefore the file paths and directories mentioned in the CPS should be valid on the machine where the Peer Server is running. If the component fails over to another peer, ensure that the machine on which the secondary Peer Server is running does have the same path available.
- If emails contain large attachments, set property **Write attachments to file?** to **yes**. The files can be later read using File Reader component, if required.
- If emails have to be available for other clients set property **Leave messages on Server?** to **yes**.
- If property **Leave messages on Server?** is set to **yes**, connection pooling should be disabled.

4.1.6 SAPR3

The SAPR3 connects and executes the various services deployed on the SAP system. This bridge receives a request XML and executes SAP BAPIs and RFCs. Every component instance can be customized to access any BAPI using the BAPI browser which allows you to choose any BAPI (pre-built and custom-built) by browsing the business objects in the SAP repository.

Points to note

The following third-party files need to be downloaded to use this component in windows. The files can be downloaded from the SAP Service Marketplace. The following URL may be used to navigate to the third-party website from where these files can be downloaded: http://service.sap.com/. After downloading the files, you need to add them as resources to the SAPR3 component.

- sapjco.jar
- librfc32.dll
- sapjcorfc.dll

4.1.6.1 Configuration and Testing

4.1.6.1.1 Connection Properties

- **Connection Pool Name**: Name of the pool from which to take the connection (used by Pool Manager).
- **Maximum Connections**: max. number of connections allowed for the pool.
- **Client**: SAP Logon Client.
- **User Name**: SAP Logon user.
- **Password**: SAP Logon Password.
- **Language**: SAP Logon Language.
- **Commit Transaction**: If yes, it commits the transaction after each request.
- **Host**: Host name of the application server.
- **System**: SAP system number.
- **Repository Name**: SAP Repository Name.
- **Connection Timeout**: Socket connection timeout value in milliseconds. This is the time duration (in milliseconds) for which the component waits while trying to establish connection with the SAP server.

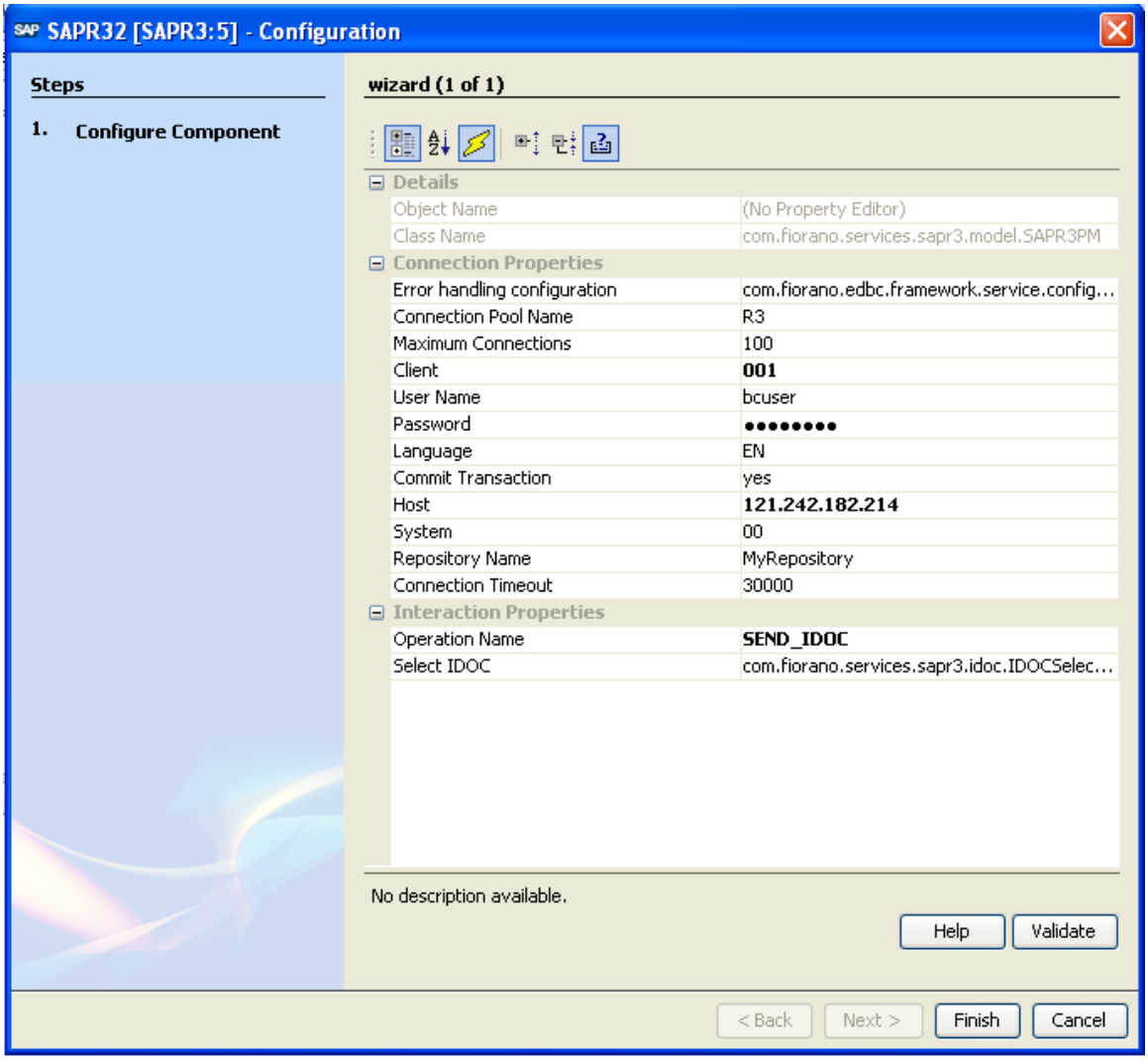


Figure 1:SAPR3 CPS

4.1.6.1.2 Interaction Properties

Operation Name: Operations which need to be called upon SAP System can be configured here. The operations SEND_IDOC & INVOKE_BAPI can be selected from drop-down list.

INVOKE_BAPI

- **BAPI/RFC Name**: When we click the ellipses(…) button, a dialog box appears from which we can select multiple BAPIs. For each BAPI selected, input and output ports are generated with their names as "<BAPI NAME>_IN", "<BAPI NAME>_OUT". *Multiple BAPIs can be entered manually semi-colon separated. Refer Figure 5 for example.

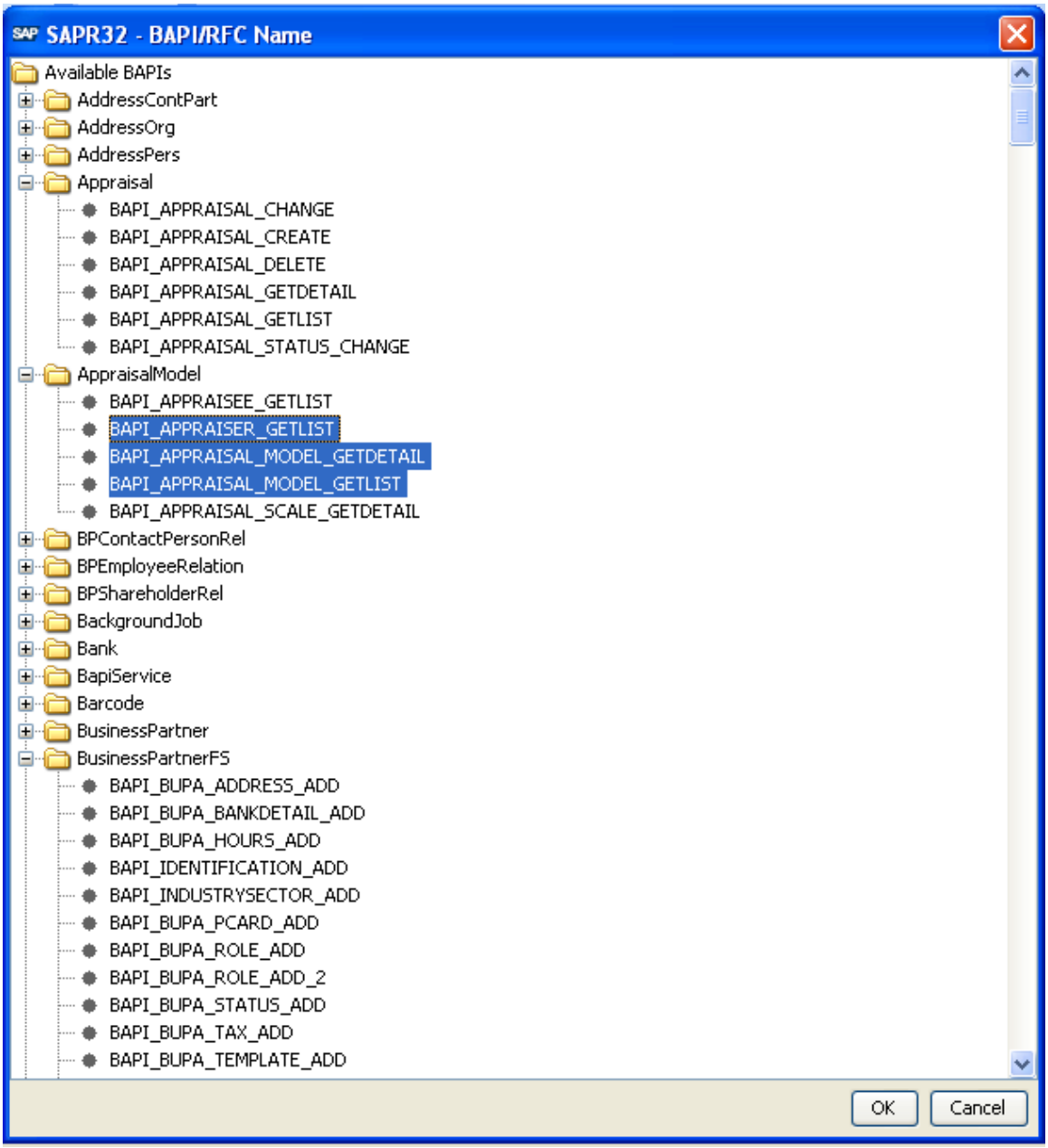


Figure 2: Dialog to select BAPI/RFC Name

SEND IDOC

- **Select IDOC**: When user clicks the ellipses(…) button, a dialog box appears from asking for IDOC TYPE, Version and a load button which loads IDOCs of type mentioned. User can select an IDOC.
- **IDOC Type**: Accepts wildcards '%' and '*', to fill all IDOCs in dropdown list. User can also enter IDOC Type. After entering wildcards/IDOC Type press the load button to populate drop-down with IDOCs of desired IDOC Type. For IDOC Type selected, input and output ports are generated with their names as "<DOC TYPE>_IN", "<DOC TYPE>_OUT".

- **Select IDOC**: Select IDOC from dropdown list.
- **Version**: Enter the version of IDOC.

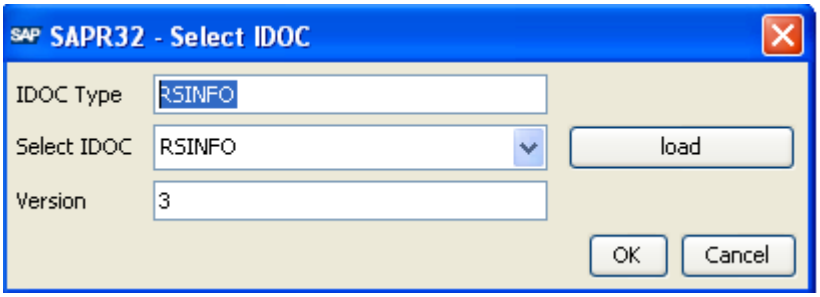


Figure 3: Select IDOC

4.1.6.2 Functional Demonstration:

This flow contains SAPR3 component configured to perform three calls to SAP System i.e., to fetch User list, check existence of a particular user & Details of a user.

4.1.6.2.1 Scenario

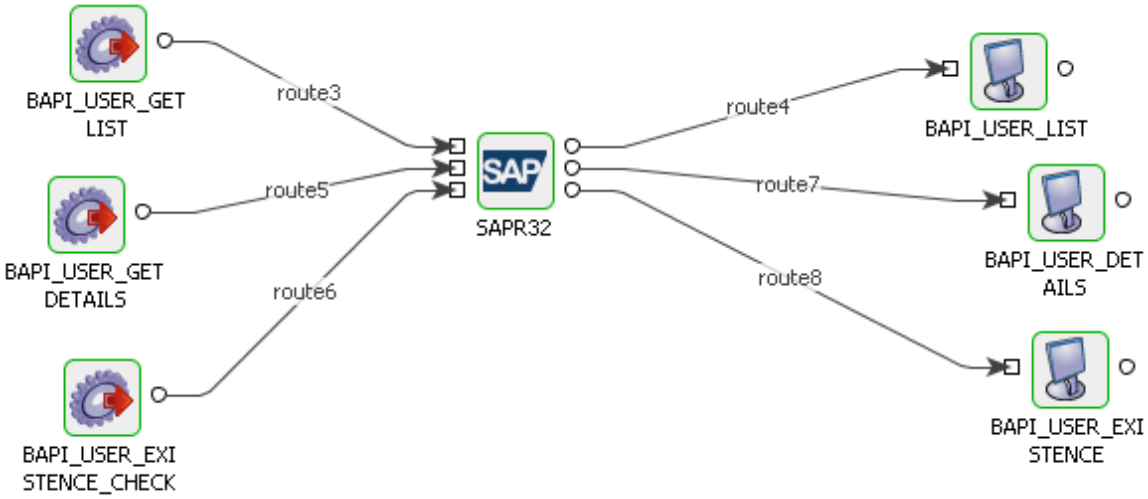


Figure 4: SAPR3 Component configured to invoke multiple BAPIs

Details	
Object Name	(No Property Editor)
Class Name	com.fiorano.services.sapr3.model.SAPR3PM
Connection Properties	
Error handling configuration	com.fiorano.edbc.framework.service.configuration.ErrorHandlingConfiguration...
Connection Pool Name	R3
Maximum Connections	100
Client	**001**
User Name	bcuser
Password	••••••••
Language	EN
Commit Transaction	yes
Host	**121.242.182.214**
System	00
Repository Name	MyRepository
Connection Timeout	30000
Interaction Properties	
Operation Name	INVOKE_BAPI
BAPI/RFC Name	BAPI_USER_GET_DETAIL;BAPI_USER_GETLIST;BAPI_USER_EXISTENCE_CHECK

Figure 5: Configuration

```
<ns1:SAPR3_BAPI_INPUT xmlns:ns1="http://www.fiorano.com/fesb/activity/sapr3/IN">
   <INPUTPARAMS>
      <USERNAME>ANOOP</USERNAME>
   </INPUTPARAMS>
</ns1:SAPR3_BAPI_INPUT>
```

Figure 6: Sample Input

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ns1:SAPR3_BAPI_OUTPUT xmlns:ns1="http://www.fiorano.com/fesb/activity/sapr3/OUT">
   <OUTPUTPARAMS>
      <RETURN type="constraint_structure">
         <TYPE>I</TYPE>
         <ID>01</ID>
         <NUMBER>088</NUMBER>
         <MESSAGE>User AN00P exists</MESSAGE>
         <LOG_NO/>
         <LOG_MSG_NO>000000</LOG_MSG_NO>
         <MESSAGE_V1>AN00P</MESSAGE_V1>
         <MESSAGE_V2/>
         <MESSAGE_V3/>
         <MESSAGE_V4/>
         <PARAMETER/>
         <ROW>0</ROW>
         <FIELD/>
         <SYSTEM/>
      </RETURN>
   </OUTPUTPARAMS>
   <TABLEPARAMS/>
</ns1:SAPR3_BAPI_OUTPUT>
```


Figure 7: Sample Output

4.1.6.3 Useful Tips

After migration, save the route transformations present on the routes connecting the SAPR3 component before reconfiguring the component

4.1.7 SMS Bridge

The SMS component enables you to send short messages or SMS using a configured SMS Server. This component is extremely useful in sending SMS to notify specified recipient mobile phone users to initiate corrective action in the event of an error or any other configured event.

Points to note

- To use the component, an account with http://www.simplewire.com/ is required.
- If you are a beta-tester with http://www.simplewire.com/ then you should use the server name as **wmp**.

4.1.7.1 Configuration and Testing

To test this adapter you need an account with http://www.simplewire.com which gives you the PIN number, Subscriber ID, Password and Callback number to send messages.

4.1.7.2 Managed Connection Factory Configuration

In the Managed Connection Properties panel of CPS, the following attributes can be configured.

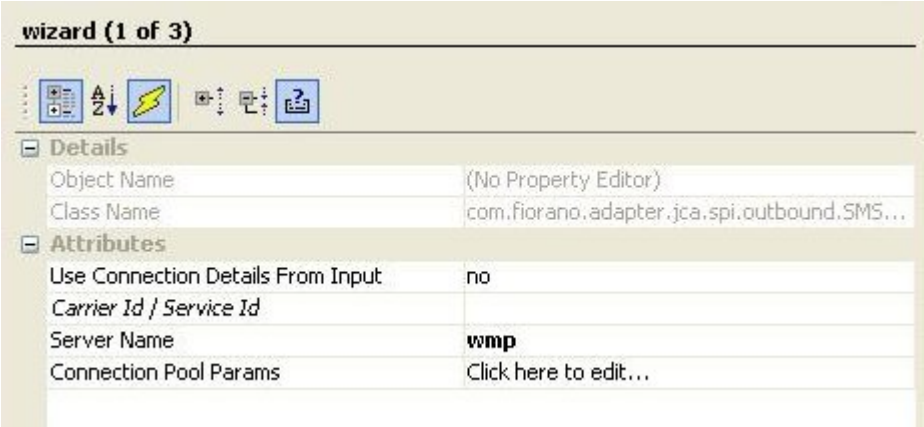


Figure: Managed Connection Factory

Use Connection Details From Input: Parameters to create the connection can be specified in the input message when this property is set to true. If this property is selected the validation errors in the managed connection factory panel of the CPS are treated as warnings. So user can bypass this step without giving valid configuration and complete the configuration of the component. If valid properties are not provided even in the input message exception will be thrown at runtime.

Carrier Id/ Service ID: This is an Optional Property used to set the message carrier ID of the recipient's wireless device. The message carrier ID is the ID number that **Simple wire** uses to identify carriers.

Server Name: Sets the name of the server for use in the connection. The server name works in conjunction with the server domain to produce the URL to which the current message gets posted. This value is preset and should not need to be changed, unless you are a Beta-Tester default (wmp).

Connection Pool Params: Parameters which are used in connection pooling of the EIS Connection.

Note: You can use the default values of these parameters if you use a Simple Wire Login.

Server connection can be tested from within the CPS by clicking on **test** button in the connection properties panel.

Interaction Configuration

- **Pager Identification/Mobile No (PIN)**: This is the intended recipient of the message.
- **From Field**: Name of the Sender of the Message.
- **Callback Number**: message callback is the number that gets dialed when a recipient presses 'talk' on their device after viewing a message.

- **Subscriber ID**: The subscriber ID is an ID number provided to paid subscribers that gives access to all of Simple wire's resources. The appropriate password must also be set.
- **Subscriber Password**: Sets the password that goes along with the subscriber ID. Each paid subscriber is given a unique ID. Each subscriber ID has an associated password.

4.1.7.2.1 Using Named Configurations

Changes are done in eStudio wrt to Named Configurations.With the implementation of Named Configurations MCF panel is changed as shown in Figure below.

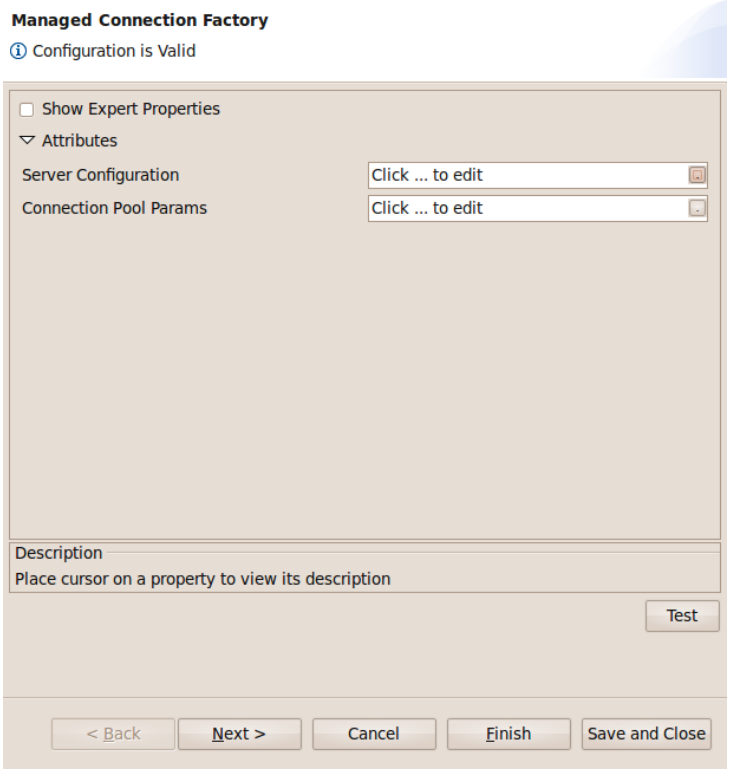


Figure: Modified Managed Connection Factory

Some properties are grouped as one property in the modified configuration panel. Carrier ID/Service ID, Server Name which were present previously in the MCF panel are grouped under the name Server Configuration.

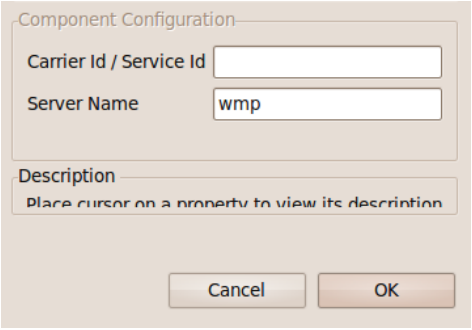


Figure: Server Configuration

Properties Pager Identification/Mobile No.(PIN),From Field, Callback Number, Subscriber Id, Subscriber Password which were present previously in the Interaction panel are grouped under the name Subscriber Configuration.

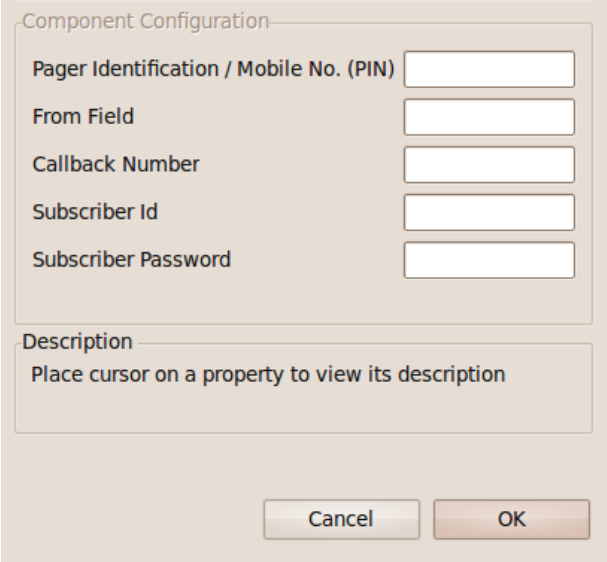


Figure: Subscriber Configuration

Sample Input and Output

The configuration can be tested by clicking the **Test** button in the interaction Configuration panel.

```
testMessage
```

Figure 4.66: Sample Input Message

```
SMS sent successfully
```

Figure 4.67: Response Generated

Input Schema

When the property Use Connection details from input is chosen, the following input schema with the element ConnectionFactorySettings, is generated. Properties that are used to create the connection are present under this element.

XML Item	Value Type
<ns1:SMSRequest>	
<ConnectionFactorySettings>?	
<ConnectionProperties>?	
<ServiceID>?	#string
<ServerName>?	#string
<SMSMessage>	#string

Figure: Input schema with ConnectionFactorySettings

4.1.7.3 Functional Demonstration

4.1.7.3.1 Scenario 1

This scenario demonstrates the usage of SMSBridge component to send short messages.

Configure SMSBridge as described in *Configuration and Testing* section and use feeder and display components to send sample input and check the response respectively.

For the sake of convenience the configuration used in this scenario is presented here.

MCF Configuration

Carrier Id / Service Id (Optional)	
Server Name	**wmp**
Connection Pool Params	Click here to edit...

Interaction Configuration

Pager Identification / Mobile No. (PIN)	+11005140730
From Field	**Fiorano**
Callback Number	**+11005101234**
Subscriber Id	**149-918-851-38820**
Subscriber Password	*********

Figure 4.68: Configuration Used

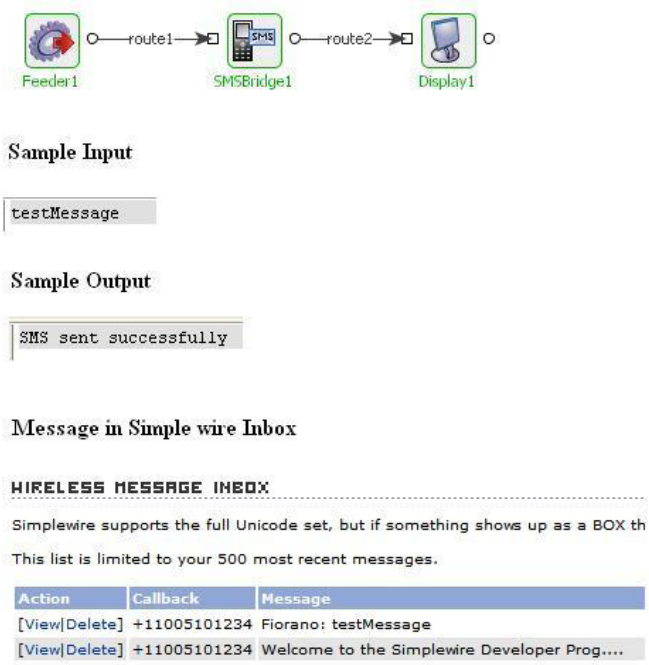


Figure 4.69: Scenario demonstration with sample input, output and the received in Inbox

4.1.7.4 Useful Tips

- To use the component, an account with http://www.simplewire.com/ is required.
- If you are a beta-tester with http://www.simplewire.com/ then you should use the server name as wmp.

4.1.8 SMTP

SMTP Bridge component allows you to connect to a remote email/news server and send emails/news articles. SMTP is capable of handling:

- Simple text e-mails/news articles
- HTML e-mails/news articles
- E-mails/news articles with attachments
- Sign, encrypt mail and MDN

SMTP Bridge uses SMTP protocol for transmission of emails, NNTP protocol for transmission of news articles and AS1 protocol for signing, encrypting mail and MDN request. It uses the SMTP implementation from the **JavaMail** API and for NNTP, GNU a service provider for NNTP protocol to JavaMail API.

Note: When sending attachments with the mail, if readFromFile attribute is set to yes, then the content of the tag is treated as the filename and if it is set to no, then content of the tag is treated as the content of the attachment with a dummy filename.

4.1.8.1 Managed Connection Factory Panel

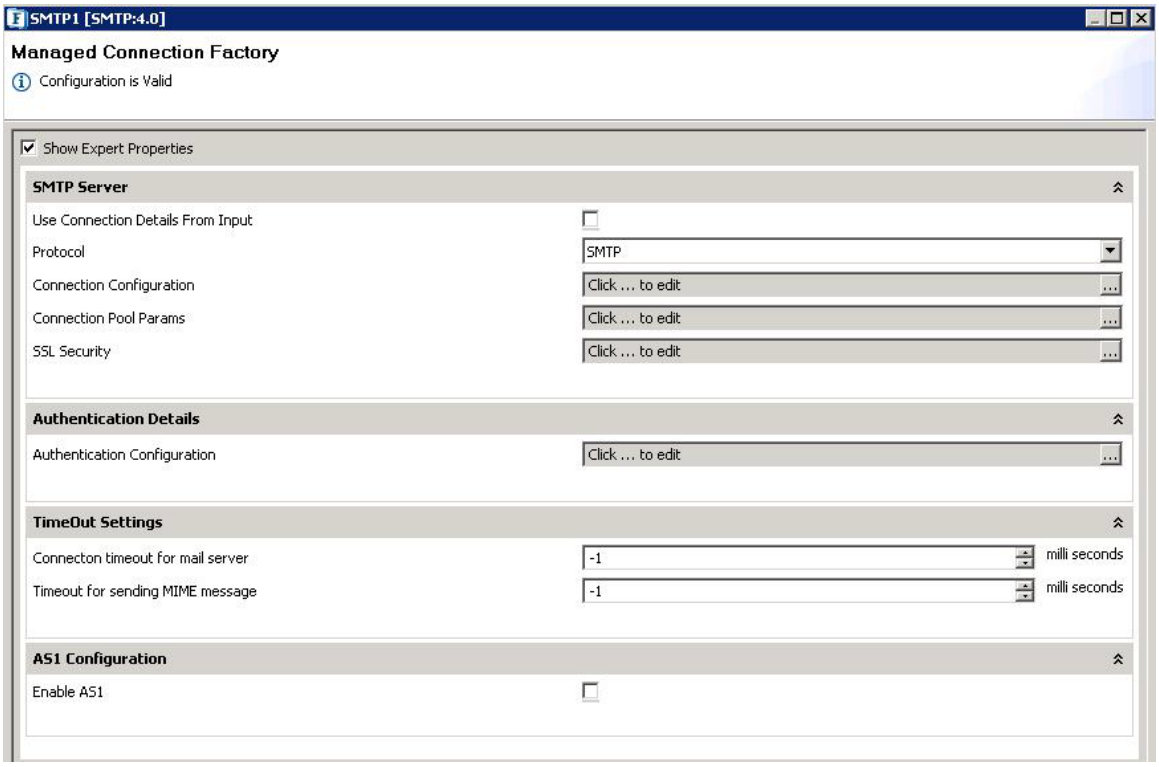


Figure 1: Managed Connection Factory

4.1.8.1.1 SMTP Server

MailServer IP/Name

The IP address or name of the SMTP mail server/NNTP news server.

MailServer Port

The port on which the SMTP server is running on the host specified by the property **MailServer IP/Name**. Default port for NNTP is 119 and SMTP is 25

Use DefaultSession

The component uses a session (javax.mail.Session) to represent a mail session. A session collects together properties and defaults used by the mail APIs.

1. If this option is set to Yes, the default session object (javax.mail.Session) is used to connect to the mail server. If a default is not setup, a new Session object is created and installed as default.
2. If this option is set to No, then the default session object is not used and a new session object is created for every connection attempt.

4.1.8.1.2 Authentication Details

- **Authentication required**

  Specifies whether the connection has to be authenticated by the SMTP/NNTP server. The properties **Login Name** and Password for authentication are relevant only if Authentication is required. In case of NNTP, new server authenticates users against their preset account ids and passwords. The user needs to provide authentication information in order to post news articles to the news server.

- **Login Name**

  The Login Name with which the connection to SMTP/NNTP server is made. An user with login name specified must be valid with respect to the server whose URL is specified by property **MailServer/NewsServer IP/Name**.

- **Password for authentication**

  Password for the user as specified by the property Login Name.

4.1.8.1.3 TimeOut Settings

- **Connection Timeout for mail server**

  Socket **connection timeout** value in milliseconds. This is the time duration (in milliseconds) for which the component waits while trying to establish connection with the server. If the component fails to get a valid connection in the specified connection timeout interval, then the retry behavior depends on the configuration specified in **Error Configuration Panel**. Default value **-1** indicates infinite timeout.

- **Timeout for sending MIME message**

  Socket **I/O timeout** value in milliseconds while sending MIME message. If sending a MIME message requires more time, the current connection will be lost. For example, if attaching a file takes more than the specified timeout period then connection will be timed out by the component. Default value **-1** indicates infinite timeout.

4.1.8.1.4 AS1 Configuration

- **Enable AS1**: Select this option to enable AS1 and provide signing, encryption and MDN configuration by clicking the ellipsis button of AS1 SSL Configuration property.

Signature Configuration

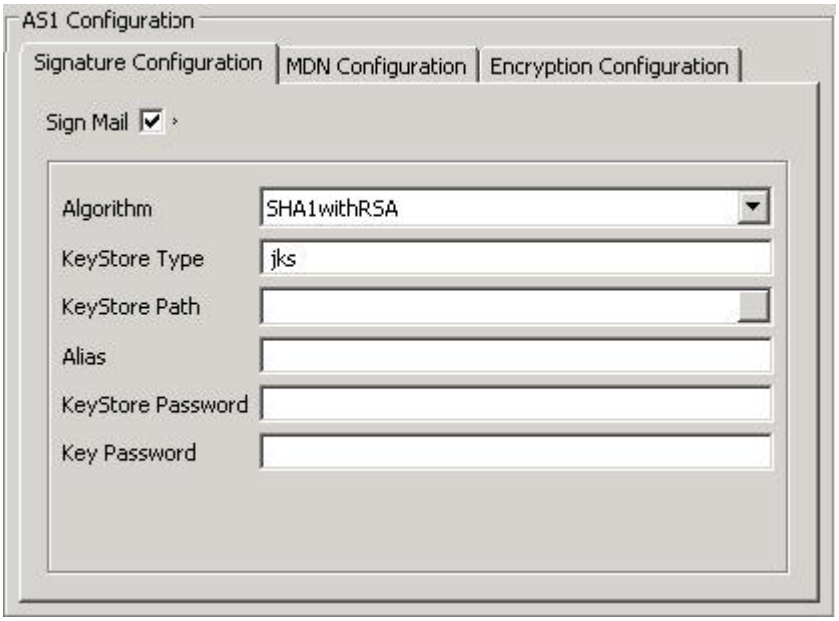


Figure: Signature configuration details

- **Sign Mail: Select** to sign mail and provide algorithm and key store details to be used for signing.

MDN configuration

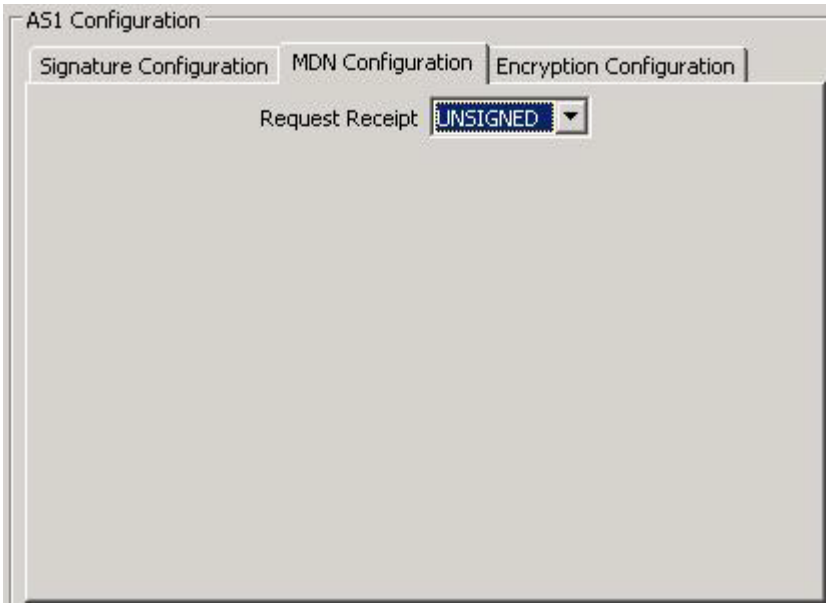


Figure: MDN configuration details

- **Request Receipt: Choose** whether MDN should be signed or unsigned.

Encryption Configuration

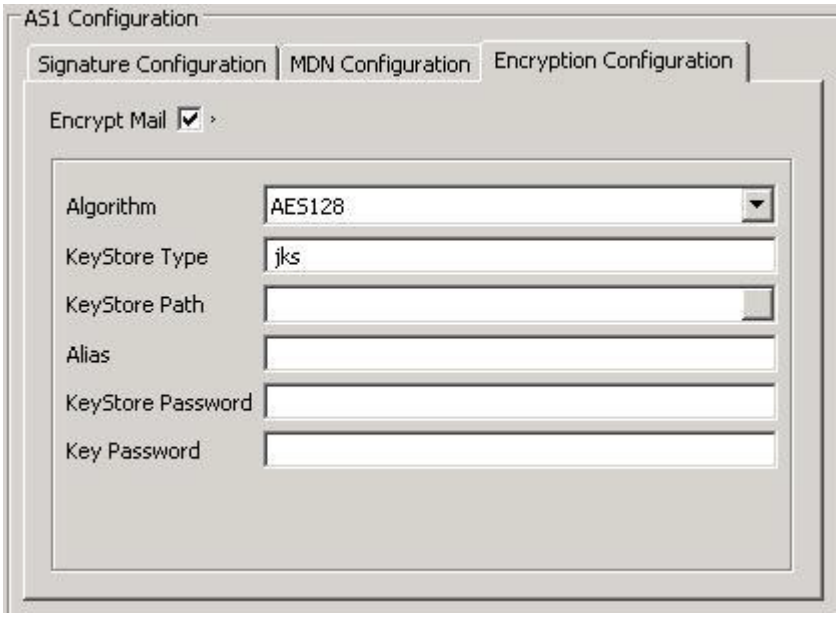


Figure: Encryption configuration details

- **Encrypt Mail:** When this option is selected, mail will be encrypted with the provided algorithm and Keystore details.

4.1.8.2 Interaction Configurations Panel

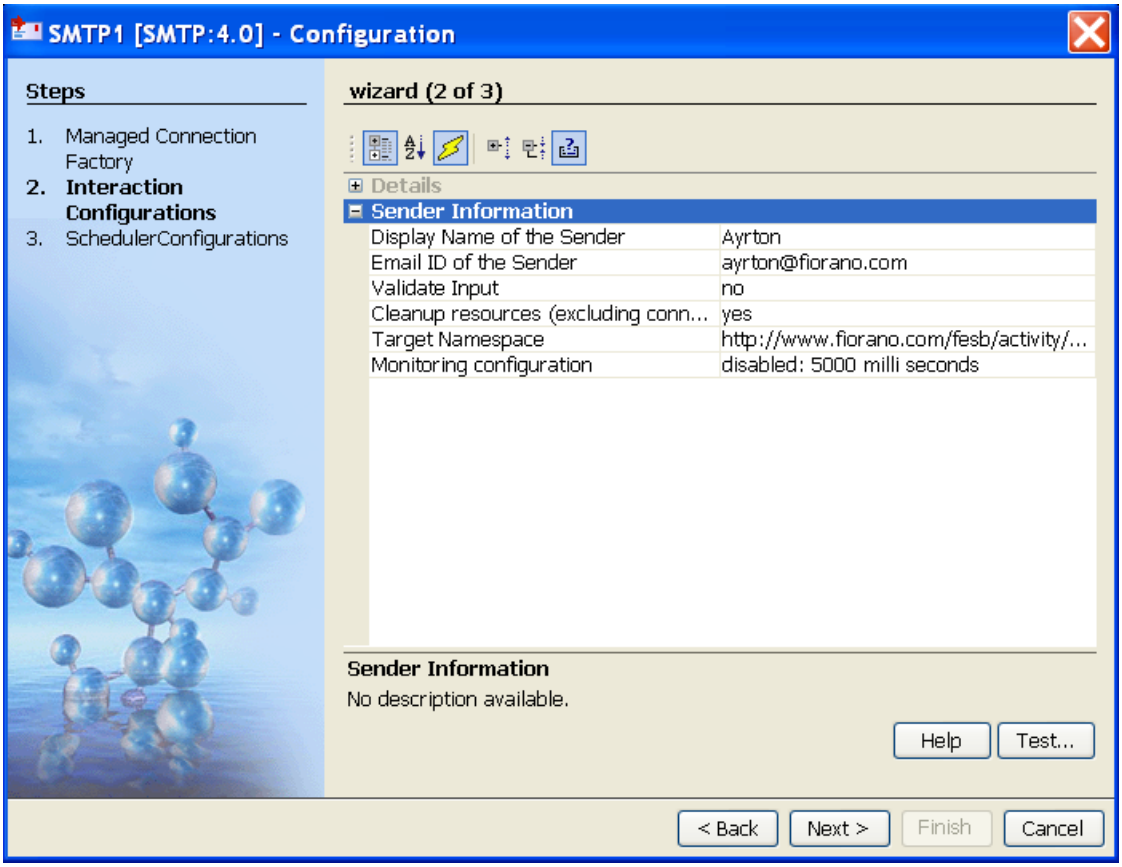


Figure 2: Interaction Configurations

4.1.8.2.1 Sender Information

Display Name of the Sender:

Name of the Sender to be displayed in the mail that is sent using the component.

Note: If the input message has <From> element, its value overrides the display name specified by this property.

Email ID of the Sender:

The sender's e-mail ID. This ID must be valid with respect to the server details provided in the **Managed Connection Factory** Panel.

Note: If the input message has <From> element, its value overrides the e-mail ID specified by this property.

4.1.8.3 Input

SMTP component takes the input in XML format as shown below when SMTP protocol is selected.

```
<ns1:Email xmlns:ns1="http://www.fiorano.com/fesb/activity/SMTP1/smtp/in">
   <To>To</To>
   <From>From</From>
   <CC>CC</CC>
   <BCC>BCC</BCC>
   <ReplyTo>ReplyTo</ReplyTo>
   <Subject>Subject</Subject>
   <Headers>
      <Header name="name" value="value"/>
   </Headers>
   <Attachments>
      <Attachment name="name" readFromFile="no" base64Encoded="no">string</Attachment>
   </Attachments>
   <Body>
      <TextBody>TextBody</TextBody>
      <HtmlBody>HtmlBody</HtmlBody>
   </Body>
</ns1:Email>
```

Figure 3: Input in XML format

SMTP component takes the input in XML format as shown below when NNTP protocol is selected.

```
<ns1:Email xmlns:ns1="http://www.fiorano.com/fesb/activity/SMTP1/smtp/in">
  <Newsgroup>Newsgroup</Newsgroup>
  <From>From</From>
  <Subject>Subject</Subject>
  <Headers>
    <Header name="name" value="value"/>
  </Headers>
  <Attachments>
    <Attachment name="name" readFromFile="no" base64Encoded="no">string</Attachment>
  </Attachments>
  <Body charset="UTF-8">
    <TextBody>TextBody</TextBody>
    <HtmlBody>HtmlBody</HtmlBody>
  </Body>
</ns1:Email>
```

Figure: Input in XML format

- **To:** Applicable only for SMTP protocol, E-mail ID of the primary recipient(s). For multiple recipients, the e-mail IDs should be separated by comma.
- **Newsgroup**: Applicable only for NNTP protocol Newsgroup ID/Name of the primary recipient(s). For multiple recipients, the Newsgroup IDs should be separated by comma.
- **From:** E-mail ID of the sender. The E-mail ID provided here will override the value provided for property **Email ID of the Sender** in the CPS. This element is optional.

- **CC:** Applicable only for SMTP protocol, E-mail ID of the CC (carbon copy) recipient(s) to be copied in the e-mail. For multiple recipients, the e-mail IDs should be separated by comma. This element is optional.
- **BCC:** Applicable only for SMTP protocol, E-mail ID of the BCC (blind carbon copy) recipient(s) to be copied in the mail. For multiple recipients, the e-mail IDs should be separated by comma. This element is optional.
- **ReplyTo:** Applicable only for SMTP protocol, ReplyTo header field. Comma separated e-mail IDs can be used here.

  The ReplyTo field is used by some e-mail programs when the Reply address is different than the From address.

  While replying to an e-mail using Reply function, if ReplyTo header was set on the message, then the e-mail client shows the Reply-To field instead of the From field in the To address. This element is optional.
- **Subject:** Subject of the e-mail. This element is optional.
- **Headers:** The headers provided (name value pairs) are added as Headers in the message. This element is optional.

  Note: This cannot be used to replace the default e-mail headers.
- **Attachments:** This option is used to send attachments in the e-mail. This element is optional.

  The attachment name is the value of the name attribute in the Attachment element.

Note:

- If the value of **readFromFile attribute** is set to No, then a new attachment file is created with the data provided against this tag as the file contents, and added as an attachment to the email.

  Example: <Attachment name="attachment" readFromFile="no" base64Encoded="no">attachment content</Attachment>
- If the value of readFromFile attribute is set to Yes, then the path of the file which has to be added as an attachment should be provided with this tag.

  Example: <Attachment name="attachment" readFromFile="yes" base64Encoded="no">/path/of/attachment</Attachment>
- The attribute **base64encoded** attribute specifies whether the attachment is base64 encoded. This property is used only when readFromFile attribute is set to No.
- If base64encoded value is set to Yes, then the value is base64 decoded before sending as an attachment.
- **Body:** Used to specify the e-mail message body. This element is optional.
  - **TextBody:** Sets the given string as the body content with a MIME type of text/plain.
  - **HTMLBody:** Sets the given value as the body content with MIME type text/html.

4.1.8.4 Output

If the e-mail is sent successfully, then the component sends an XML output with a single element Result.

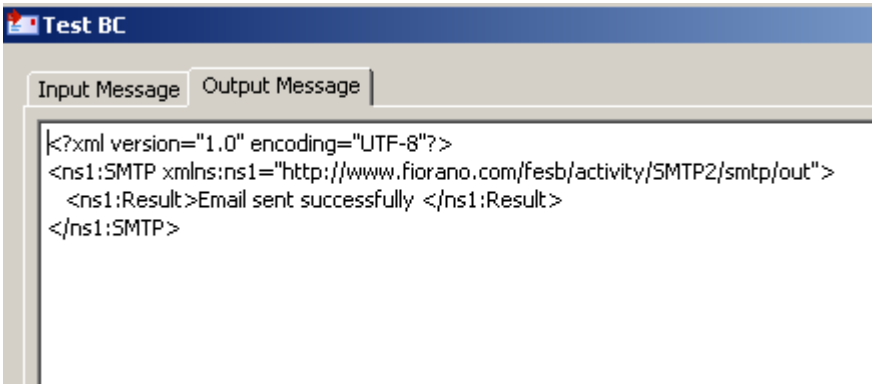


Figure 4: Sample response

4.1.8.5 Functional Demonstration

4.1.8.5.1 Scenario 1

Sending HTML mails with attachment (**Note**: choose scenario(s) that can be superset in terms of number of features it can demonstrate).

Configure the SMTP Bridge as described in Configuration and Testing section and use Feeder and Display component to send sample input and check the response respectively.

Sample Input:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:Email xmlns:ns2="http://www.fiorano.com/fesb/activity/SMTP1/smtp/in" xmlns:ns1="http://www.w3.org/2001/XMLSchema">
  <To>avrton@fiorano.com </To>
  <Subject>Confirmation for Order No.12345</Subject>
  <Attachments>
    <Attachment name="12345.txt" readFromFile="yes">c:/temp/order.txt</Attachment>
  </Attachments>
  <Body>
    <HtmlBody>Hi Ayrton,

This is a confirmation message for order no.12345Please check the attachments for more details.


Thanks,
Fiorano Support.</HtmlBody>
  </Body>
</ns2:Email>
```

Sample Output:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:SMTP xmlns:ns1="http://www.fiorano.com/fesb/activity/SMTP1/smtp/out">
   <ns1:Result>Email sent successfully </ns1:Result>
</ns1:SMTP
```

Figure 5: Demonstrating Scenario 1 with sample input and output

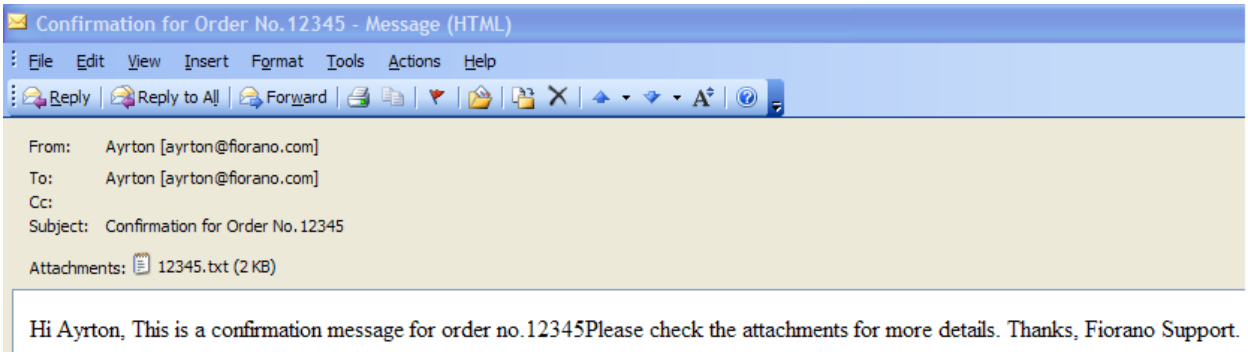


Figure 6: Mail sent by SMTP Bridge in the Inbox

4.1.8.6 Use Case Scenario

In an order entry scenario e-mails can be sent to concerned party when a PO is accepted or rejected.

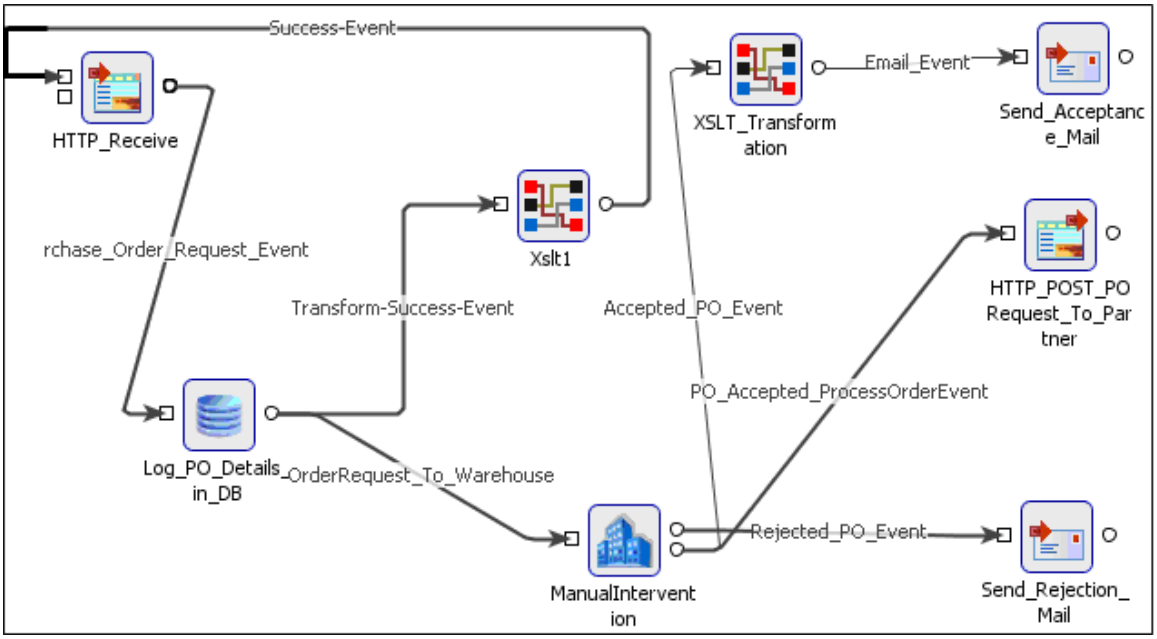


Figure 7: Event Process demonstrating

The Event Process demonstrating this scenario is bundled with the installer.

Documentation of the scenario and instructions to run the flow can be found in the Help tab of flow when open in Fiorano Studio.

4.1.8.7 Useful Tips

- SMTP component can be used as ESB Alerter when configured with ExceptionListener component to listen for exceptions from all flows.

4.1.9 SapR3Monitor

The SAPR3 Monitor adapter enables you to process IDocs (Intermediate Documents) of SAP systems and converts IDOC to XML Message. SAPR3 Monitor adapter listens for IDOC generated from SAP. SAPR3 Monitor can be used to trigger a event process.

Points to note

- The following third-party files need to be downloaded this component can be used on windows. The files can be downloaded from the SAP Service Marketplace. The following URL may be used to navigate to the third-party website from where these files can be downloaded: http://service.sap.com/. After downloading the files, you need to add them as resources to the SAPR3 component.
    - sapjco.jar
    - sapidoc.jar
    - sapidocjco.jar
    - librfc32.dll
    - sapjcorfc.dll

4.1.10 HL7Receiver

The HL7 Receiver listens on a port specified on a particular IP address to receive HL7 messages, sends the messages received on to the output port and sends the acknowledgement to the Sender.

4.1.10.1 Configuration and Testing

The component can be configured using the properties in the Custom Property Sheet (CPS) as shown in Figure 4.77.

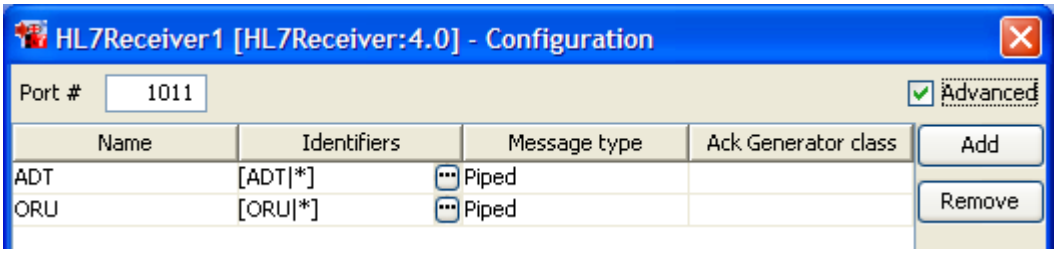


Figure 4.77: Custom Property Sheet (CPS)

Note: Check **Advanced** check box to see all fields.

- **Port #:** Port number on which HL7 Receiver is listening.
- **Name:** The name which is used in the creation of input and output ports. By default, the component has no ports. Depending on the names provided in the Custom Property Sheet, a set of input and output ports gets generated.
- **Identifiers:** It is a string of form <HL7 Message Format> | <Trigger Event> that can be configured in the Event windows as shown in Figure 4.78.

Example: The identifier ADT|A01 listens for ADT A01 messages.

Note: An asterisk* can be used as wild character for both message format and trigger event.

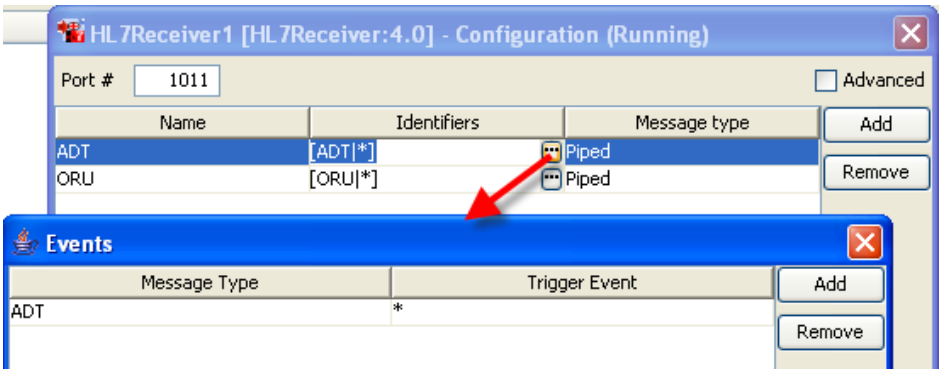


Figure 4.78: Configuring Identifiers

- **Message Type:** The type of message that is sent on to the components output port. Figure 4.79 shows the messages types that can be used.

Figure 4.79: Configuring message types

  - **Piped** – Piped message is expected on input port
  - **XML** – XML message is expected on input port

- **Ack Generator class**: A class whose instance can be delegated the responsibility of generating an ack message for HL7 message received. If value is not provided a default ack generator is used which generates AA if HL7 message is successfully converted to JMS Message and sent on output port and AE otherwise.

Ack Generator should implement com.fiorano.services.hl7receiver.engine. IAckGenerator and should have a default constructor.

```
public interface IAckGenerator {

    /**
     * Generate Acceptance Ack for <code>message</code> received
     * @param message message received by recevier
     * @return ack message which will be sent back to the sender
     * @throws HL7Exception
     * @throws IOException
     */
    Message generateAckForSuccess(Message message) throws HL7Exception, IOException;

    /**
     * Generate Error Ack for <code>message</code> received when an exception <code>e</code> happens
     * while proccessing the <code>message</code>
     * @param message message received by recevier
     * @param e excveption occured while processing the message
     * @return ack message which will be sent back to the sender
     * @throws HL7Exception
     */
    Message generateAckForException(Message message, Exception e) throws HL7Exception;
}
```

Figure 4.80: Ack Generator class

4.1.10.2 Functional Demonstration

Figure 4.80 illustrates the event process where HL7Sender accepts ADT and ORU messages and sends them to HL7Receiver. Figure 4.81 illustrates the event process where HL7Receiver listens to the messages from HL7Sender and sends them to the output port.

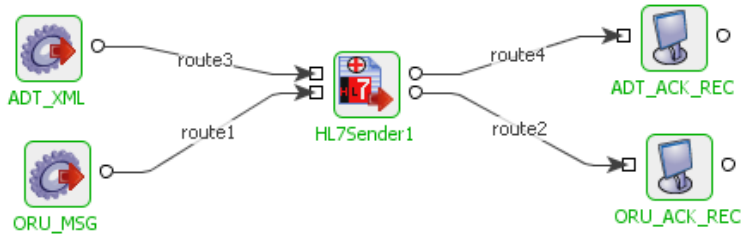


Figure 4.81: Sample Event process using HL7Sender

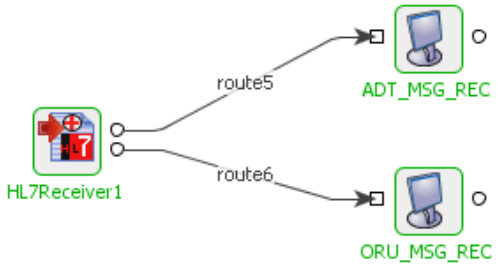


Figure 4.82: Sample Event process using HL7Receiver

4.1.10.2.1 Scenario 1

Receiving an ADT A01 message

Configure the HL7Receiver as shown in Figure 4.77.

Figure 4.82 illustrates a snapshot of the ADT A01 message received by HL7Receiver, when the message (shown in figure 4.83) is sent by HL7Sender. Figure 4.84 illustrates the acknowledgement sent by HL7Receiver.

```
MSH|^~\&|REGADT|MCM|IFENG||199112311501||ADT^A01|000001|P|2.4|||EVN|A04|199901101500|19
```

Figure 4.83: ADT A01 message received by HL7Receiver

```
<?xml version="1.0"?>
<ADT_A01 xmlns="urn:hl7-org:v2xml">
    <MSH>
        <MSH.1>|</MSH.1>
        <MSH.2>^~\&amp;</MSH.2>
        <MSH.3>
            <HD.1>REGADT</HD.1>
        </MSH.3>
        <MSH.4>
            <HD.1>MCM</HD.1>
        </MSH.4>
        <MSH.5>
            <HD.1>IFENG</HD.1>
        </MSH.5>
        <MSH.7>
            <TS.1>199112311501</TS.1>
        </MSH.7>
        <MSH.9>
            <MSG.1>ADT</MSG.1>
```

Figure 4.84: Sample ADT A01 message sent by HL7Sender

```
MSH|^~\&|||||20080327191849.565+0530||ACK|1|P|2.4MSA|AA|000001
```

Figure 4.85: Acknowledgement sent by HL7Receiver

4.1.10.2.2 Scenario 2

Receiving an ORU R01 message

Configure HL7Receiver as shown in Figure 4.77.

Figure 4.85 illustrates a snapshot of the ORU R01 message received by HL7Receiver, when the message (shown in Figure 4.86) is sent by HL7Sender. Figure 4.87 illustrates the acknowledgement sent by HL7Receiver.

```
MSH|^~\&|DEV^TRUST^ESB|PAS1|||19951010134000||ORU^R01|LABMI1199510101340007|D|2.2|||AL
```

Figure 4.86: Sample ORU R01 message received by HL7Receiver

```
MSH|^~\&|DEV^TRUST^ESB|PAS1|||19951010134000||ORU^R01|LABMI1199510101340007|D|2.2|||AL|
PID|||BB1127||LABHLVII^MICRO^A||19780625|F|||||||||001680
PV1|||CFLC|||||||MED||||||||P||||||||||||||||||||||||||||19950303150000|19950311
OBR|1||095283070860031 10|6003110^CULTURE BACTERIA UR INDWELL CA^^^C UR ICATH|||199510101319 00||||||199510101319 00|S03593^URI
OBX||TX|6000417^PRELIMINARY^^^R PRE|0001|>=100,000 COLONIES/ML ESCHERICHIA COLI||||||F|||199510101336 00||001
OBX||CE|6000417^PRELIMINARY^^^R PRE|0002|I80013^^^^GE100,||||||F|||199510101336 00||001
OBX||CE|6000417^PRELIMINARY^^^R PRE|0003|I01320^^^^E COLI||||||F|||199510101336 00||001
OBX||TX|6000417^PRELIMINARY^^^R PRE|0004|>=100,000 COLONIES/ML KLEBSIELLA PNEUMONIAE||||||F|||199510101336 00||001
OBX||CE|6000417^PRELIMINARY^^^R PRE|0005|I80013^^^^GE100,||||||F|||199510101336 00||001
OBX||CE|6000417^PRELIMINARY^^^R PRE|0006|I01570^^^^K PNEUMO||||||F|||199510101336 00||001
OBX||TX|6000417^PRELIMINARY^^^R PRE|0007|ENTEROCOCCUS FAECALIS||||||F|||199510101336 00||001
OBX||CE|6000417^PRELIMINARY^^^R PRE|0008|I40127^^^^E FAECAL||||||F|||199510101336 00||001
```

Figure 4.87: Sample ORU R01 message sent by HL7Sender

```
MSH|^~\&|||||20080327192802.033+0530||ACK|2|D|2.2MSA|AA|LABMI1199510101340007
```

Figure 4.88: Acknowledgement sent by HL7Receiver

4.1.11 HL7Sender

The HL7 Sender component is used to send the HL7 data on to a port specified on a particular IP address in a specified format. The component receives the response (acknowledgement) generated and sends it to the output port.

HL7 Sender allows sending HL7 messages onto different HL7 Receivers. A set of input and output ports is generated for each configuration.

4.1.11.1 Configuration and Testing

The component can be configured using the properties in the Custom Property Sheet (CPS) shown in Figure 4.89.

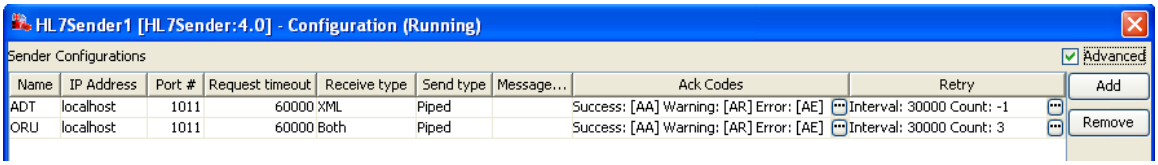


Figure 4.89: Custom Property Sheet (CPS)

Note: Select Advanced check box to see all the fields.

- **Name:** The name is used in the creation of component input and output ports. By default the component has no ports. Depending on the names provided in the property sheet, a set of input and output ports gets generated.
- **IP Address:** The IP address on which HL7 Receiver service is running.
- **Port #:** Port number on which HL7 Receiver is listening.
- **Request Time Out:** Request Time Out is the time out of the HL7 message in milliseconds. The HL7 Sender waits for the response till the timeout happens and throws an exception, if it does not receive any response.

- **Rec**eive Type: The type of message expected on component's input port. Figure 4.90 shows the receive types that can be used.

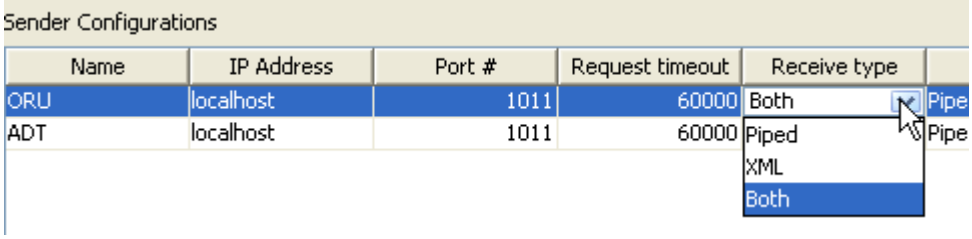


Figure 4.90: Configuring Receive type

  - **Piped** – Piped message is expected on input port.
  - **XML** – XML message is expected on input port.
  - **Both** – Message received on input port can be of any type (piped or XML).

- Send Type: Type of the Acknowledgement message which is sent on to the component's output port. Figure 4.91 shows the Send types that can be used.

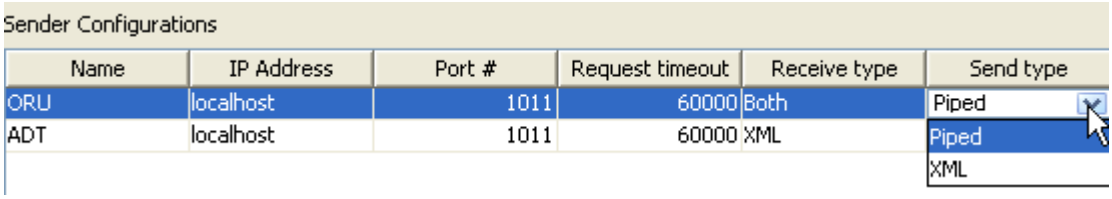


Figure 4.91: Configuring Send type

  - Piped – Piped message is expected on input port.
  - XML – XML message is expected on input port.

- Message Rectifier class: Message Rectifier class is a class whose instance is delegated the responsibility of rectifying or correcting HL7 message every time a retry is attempted. If value is not provided here then message rectification will not be done and retry is attempted with same message.

Message Rectifier should implement com.fiorano.services.hl7sender.engine. IMessageRectifier and should have a default constructor.

```
public interface IMessageRectifier {
    /**
     * This method of implementation, if exists, will be called when the <code>originalMessage</code>
     * could not be sent because of an exception.
     *
     * @param originalMessage message that has to be corrected
     * @param problem exception that was encountered when trying to send <code>originalMessage</code>
     * @return rectified message
     * @throws ServiceExecutionException any exception that might occur when rextifying
     */
    Message rectifyMessage(Message originalMessage, Exception problem) throws ServiceExecutionException;
    /**
     * This method of implementation, if exists, will be called when the <code>originalMessage</code>
     * is sent and an ack message other than AA is returned
     *
     * @param originalMessage message that has to be corrected
     * @param ackMessage ackMessage, ither than AA, that was returned in response to <code>originalMessage</code>
     * @return rectified message
     * @throws ServiceExecutionException any exception that might occur when rextifying
     */
    Message rectifyMessage(Message originalMessage, Message ackMessage) throws ServiceExecutionException;
}
```

Figure 4.92: Message Rectifier class

- Ack Codes: Each Ack Code [AA, AR, AE ….] the sender is expected to receive can be categorized as success or error or warning as shown in Figure 4.93. In case, the ack code returned to sender is configured as success or warning, the ack message is simply sent onto the output port of the component. If the received ack code is configured as warning, then a retry is attempted.

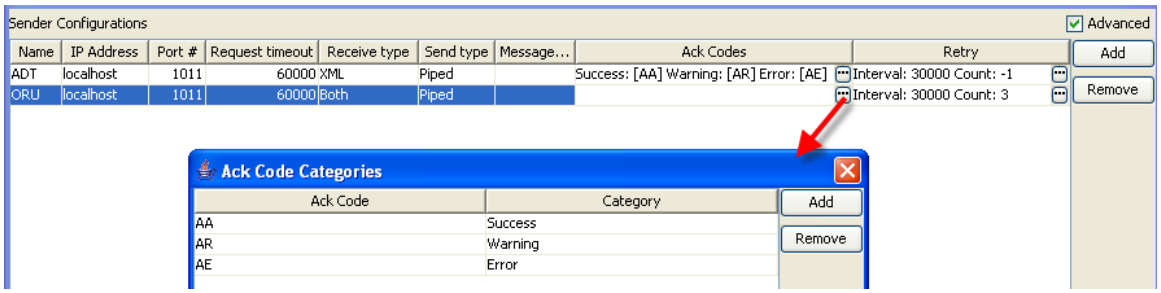


Figure 4.93: Configuring Ack Code categories

Retry Configuration: When ack code returned to sender is categorized as warning, the number of times retry is to be attempted and interval after which retry is to be attempted can be configured as shown in Figure 4.94.

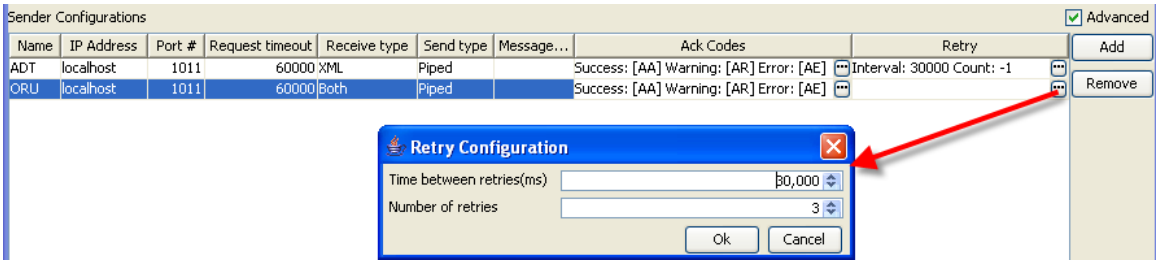


Figure 4.94: Retry configuration

4.1.11.2 Functional Demonstration

Figure 4.95 illustrates the event process where HL7Sender accepts ADT and ORU messages and sends out the corresponding acknowledgements. Figure 4.96 illustrates the event process where HL7Receiver listens to the messages from HL7Sender.

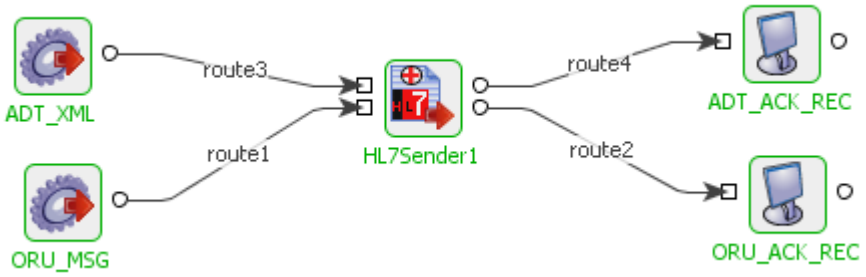


Figure 4.95: Sample Event process using HL7Sender

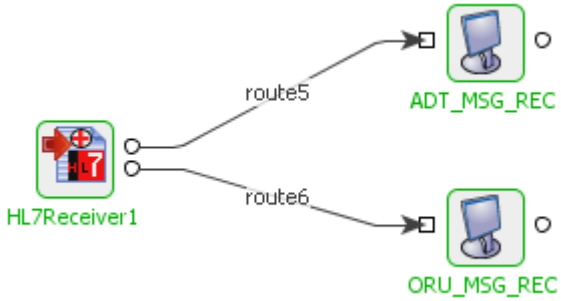


Figure 4.96: Sample Event process using HL7Receiver

4.1.11.2.1 Scenario 1

Sending an ADT A01 message

Configure the HL7Sender as shown in Figure 4.81.

When a sample ADT A01 message (shown in Figure 4.97) is sent from the Feeder ADT_XML, HL7Sender sends this message to HL7Receiver. When the message receipt is acknowledged by HL7Receiver, HL7Sender receives it (shown in Figure 4.98) and sends it to the Display ADT_ACK_REC.

```
<?xml version="1.0"?>
<ADT_A01 xmlns="urn:hl7-org:v2xml">
    <MSH>
        <MSH.1>|</MSH.1>
        <MSH.2>^~\&amp;</MSH.2>
        <MSH.3>
            <HD.1>REGADT</HD.1>
        </MSH.3>
        <MSH.4>
            <HD.1>MCM</HD.1>
        </MSH.4>
        <MSH.5>
            <HD.1>IFENG</HD.1>
        </MSH.5>
        <MSH.7>
            <TS.1>199112311501</TS.1>
        </MSH.7>
        <MSH.9>
            <MSG.1>ADT</MSG.1>
```

Figure 4.97: Sample ADT A01 message

```
MSH|^~\&|||||20080327191849.565+0530||ACK|1|P|2.4MSA|AA|000001
```

Figure 4.98: Acknowledgement received by HL7Sender

4.1.11.2.2 Scenario 2

Sending an ORU R01 message

Configure HL7Sender as shown in Figure 4.81.

When a sample ORU A01 message (shown in Figure 4.99) is sent from the Feeder ORU_MSG, HL7Sender sends this message to HL7Receiver. When the message receipt is acknowledged by HL7Receiver, HL7Sender receives it (shown in Figure 4.100) and sends it to the Display ORU_MSG_REC.

```
MSH|^~\&|DEV^TRUST^ESB|PAS1|||19951010134000||ORU^R01|LABMI1199510101340007|D|2.2|||AL|
PID|||BB1127||LABHLVII^MICRO^A||19780625|F|||||||||001680
PV1|||CFLC|||||||MED||||||||P||||||||||||||||||||||||||19950303150000|19950311
OBR|1||09528307086003110|6003110^CULTURE BACTERIA UR INDWELL CA^^^C UR ICATH|||19951010131900||||||19951010131900|S03593^URI
OBX||TX|6000417^PRELIMINARY^^^R PRE|0001|>=100,000 COLONIES/ML ESCHERICHIA COLI||||||F|||19951010133600||001
OBX||CE|6000417^PRELIMINARY^^^R PRE|0002|I80013^^^^GE100,||||||F|||19951010133600||001
OBX||CE|6000417^PRELIMINARY^^^R PRE|0003|I01320^^^^E COLI||||||F|||19951010133600||001
OBX||TX|6000417^PRELIMINARY^^^R PRE|0004|>=100,000 COLONIES/ML KLEBSIELLA PNEUMONIAE||||||F|||19951010133600||001
OBX||CE|6000417^PRELIMINARY^^^R PRE|0005|I80013^^^^GE100,||||||F|||19951010133600||001
OBX||CE|6000417^PRELIMINARY^^^R PRE|0006|I01570^^^^K PNEUMO||||||F|||19951010133600||001
OBX||TX|6000417^PRELIMINARY^^^R PRE|0007|ENTEROCOCCUS FAECALIS||||||F|||19951010133600||001
OBX||CE|6000417^PRELIMINARY^^^R PRE|0008|I40127^^^^E FAECAL||||||F|||19951010133600||001
```

Figure 4.99: Sample ORU R01 message

```
MSH|^~\&|||||20080327192802.033+0530||ACK|2|D|2.2MSA|AA|LABMI1199510101340007
```

Figure 4.100: Acknowledgement received by HL7Sender

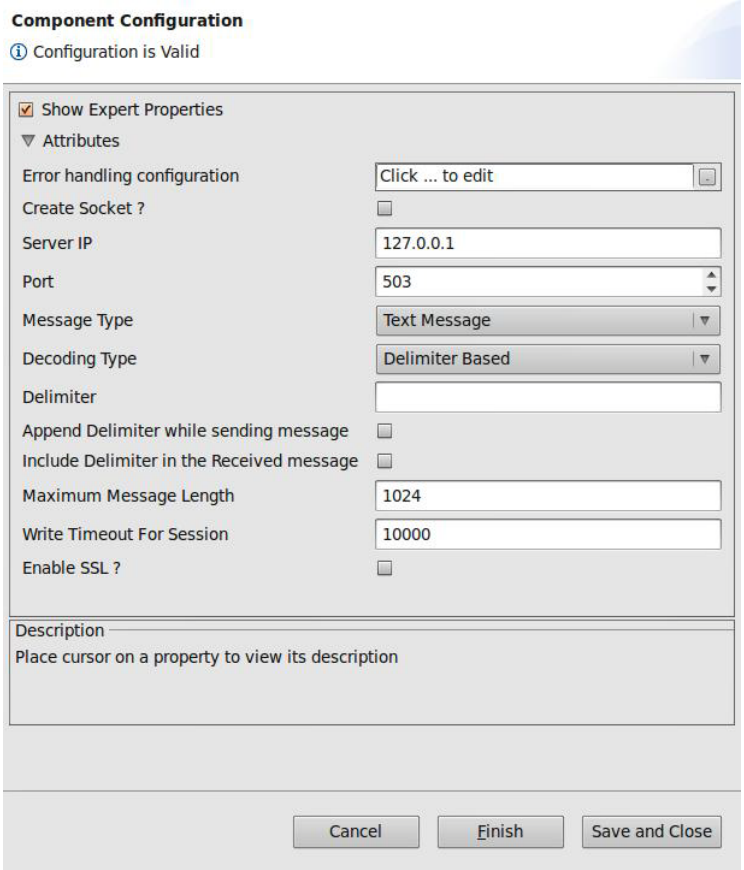
4.1.12 SocketAdapter

Socket Adapter can be used in sending data to sockets and receiving data from sockets. This adapter will have two modes of operation.

1. **Server:** Acts as a server if 'Create Socket?' property is set to true. In server mode adapter will create a socket on the configured port provided in the component property sheet and listens for connections. Once a connection is created, messages coming on the input port will be sent to the connected client/clients and the messages received from the client will be sent to output port.
2. **Client:** Acts as a client if 'Create Socket ?' property is set to false. In client mode adapter will connect to the socket configured. If a successful connection is made then the message coming on the input port will be sent to the server and the received messages will be sent to the output port.

4.1.12.1 Configuration

The configuration of Socket Adapter is defined as shown in the following figure.



Attributes:

- **Create Socket ?**:
  - Yes - Acts as a server by creating a socket on the specified port and listens for client connections.
  - No - Acts as client assuming that that server is already running on the specified port and connects to it.
- **IP**: IP address of the socket
- **Port**: Port of the socket
- **Message Type**: Mode of the message transfer.
  - **Text Message**: Messages will be encoded and decoded as Strings.
  - **Byte Message**: Messages will be encoded and decoded as bytes.
- **Decoding Type**: Messages received in the communication will be decoded in the following two ways.
  - **Delimiter Based**: Messages are decoded based on the delimiter configured.
  - **Length Based**: Messages are decode based on the length of the message.
- **Delimiter**: Delimiter to be used as an end of message.
- **Append Delimiter while sending message**: If yes, delimiter provided in the configuration will be appended to the input message before sending it to the socket.
- **Include Delimiter in the received message**: This property is visible only when Decoding type is Delimiter Based. Upon decoding the received data from socket, message will be sent to the output port with the delimiter included in it.
- **Maximum Message Length**: Length of the largest message in the messages that are expected to receive. This property is visible only when Decoding type is Delimiter Based. While decoding messages based on delimiter if a delimiter is not received till this length then a BufferDataException will occur.

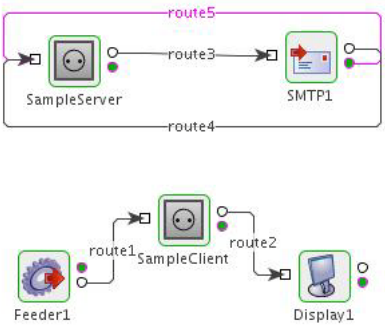
  Instead of waiting forever to match a delimiter, this property helps if the stream receiving is invalid or corrupted. Specify a larger value if the length of the receiving messages is unknown.

  This property specifies the length of the message in case of Length Based decoding. This property is visible only when the property Decoding Type is set to Length Based.
- **Write Timeout for Session**: A Session write time out will happen if a write operation (while sending message to socket) takes more time than this value.
- **Enable SSL?**: By enabling SSL and providing the required key store/trust store with respective passwords, secure message transfer can be achieved.

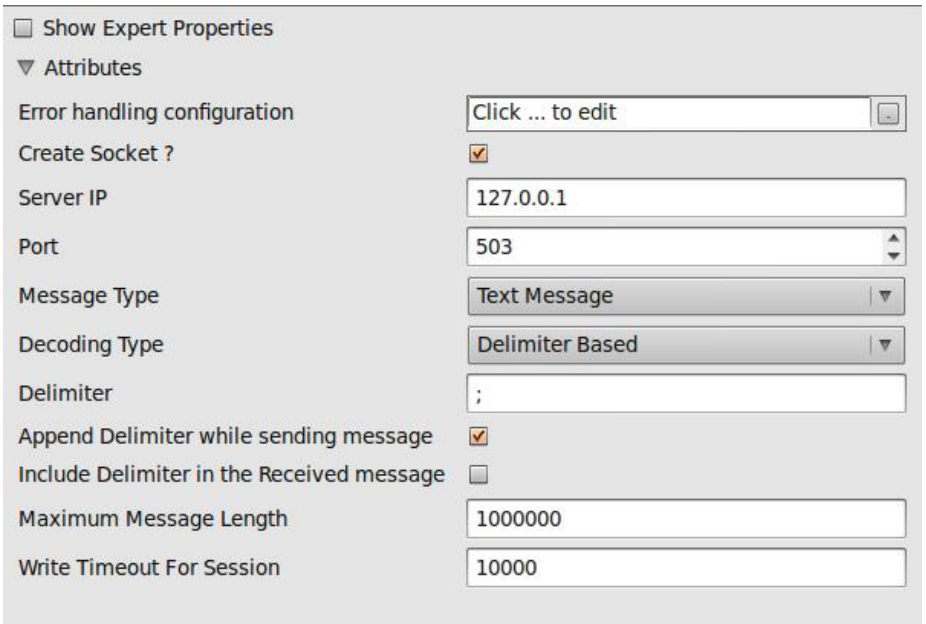
4.1.12.2 Functional Demonstration

The following Sample flow has two socket adapter, one acting as a server and the other as a client. After receiving requests from the client, server will send it to the SMTP component connected to its output port and the response from the SMTP component is sent back to the client.

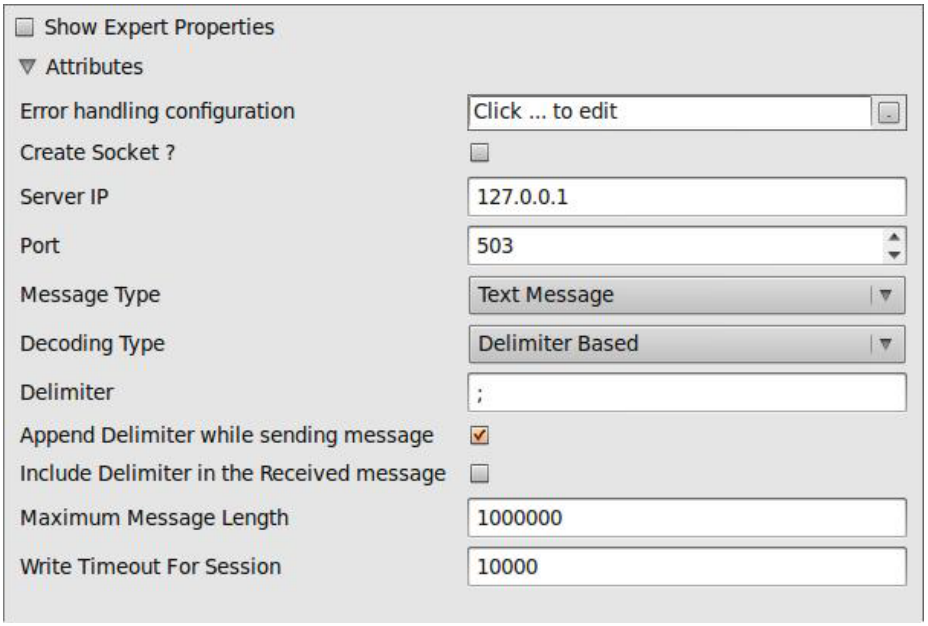
The Event Process is shown in the following figure.



'SampleServer' Component Configuration is shown in the following figure.



'SampleClient' Component Configuration is shown in the following figure.



4.1.12.3 Useful Tips

In 'Delimiter Based' decoding, Instead of waiting forever to match a delimiter, 'Maximum Message Length' property helps if the stream receiving is invalid or corrupted. Specify a larger value if the length of the receiving messages is unknown.

4.1.13 SAPB1

The SAPB1 component can be used to perform operations on SAP Business One ERP solution. Using these four versions of components, operations can be performed on the corresponding version of SAP Business One. Currently SAPB1 component supports simple select, insert and delete operations for the following business objects.

- Business Partners
- Orders
- Items
- Price Lists
- Credit notes
- Delivery notes
- Invoices
- Purchase Credit Notes
- Purchase Delivery Notes
- Purchase Invoice
- Purchase Orders
- Purchase Returns
- Quotations

- Delivery Notes Return
- Receipts Incoming Payments
- Journal Entries
- Payments to Vendors
- Stock Transfer
- Product Trees
- Production Order
- Chart of Accounts
- Inventory General Entry
- Inventory General Exit
- Down Payments
- Draft
- Payments Draft
- Purchase Down Payments
- Sales Tax Invoice
- Purchase Tax Invoice

SAPB1 component requires credentials of company to be connected as part of the configuration. These details along with chosen business object type and operation are used to connect to the company and fetch the schema which is required to perform the operation.

Points to note:

- This component cannot be launched in-memory of the peer server.
- This component requires SAP Business One client environment installed on the machine where peer server is running.
- This component requires Microsoft .NET Framework 3.5 or above installed on the machine where peer server is running.

4.1.13.1 Configuration

The component can be configured using the properties in the Custom Property Sheet (CPS) shown in figure 1.

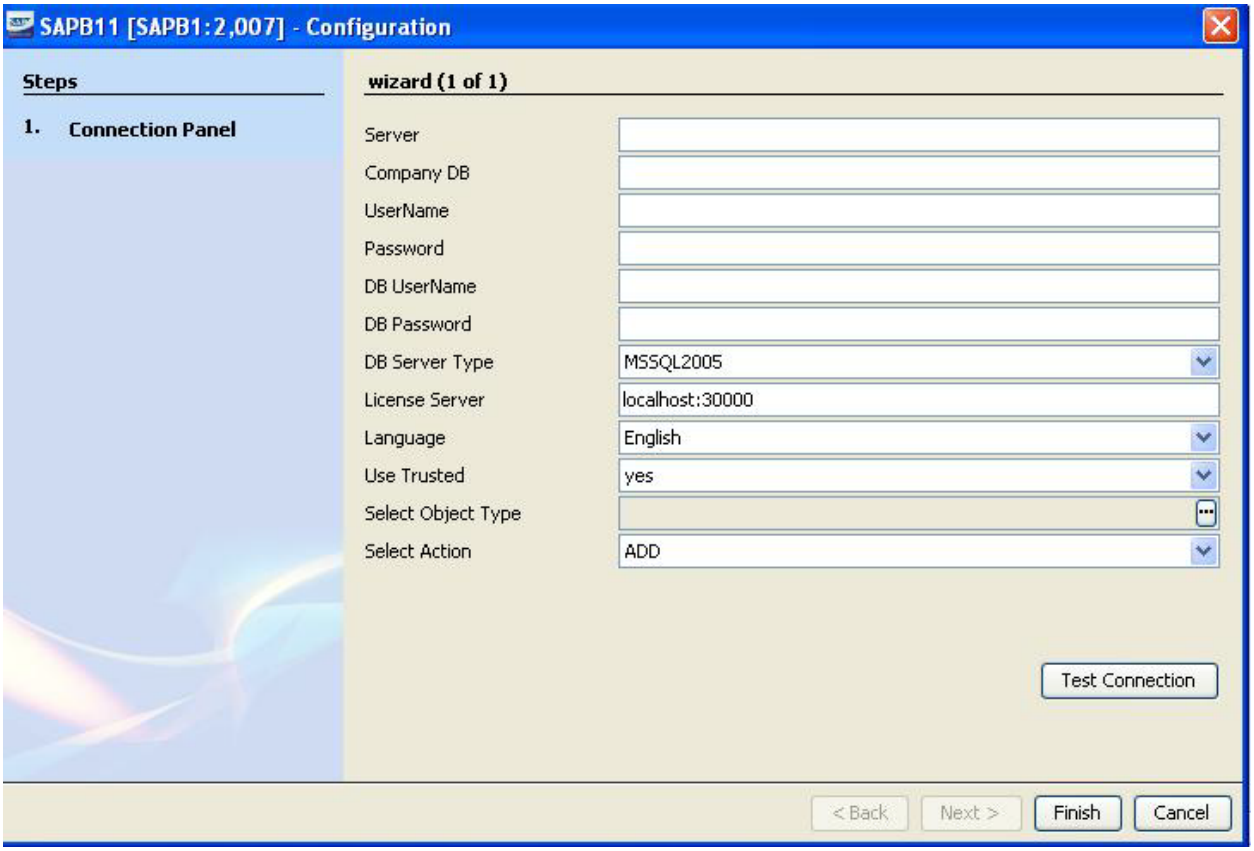


Figure 1: Custom Property Sheet (CPS)

- **Server:** Name of the machine where SAP Business One Company server is present.
- **Company DB:** Name of the SAP Business One company database.
- **UserName**: SAP Business One user login name.
- **Password**: SAP Business One user password.
- **DB UserName**: SAP Business One company database's user name required as part of SQL server authentication.
- **DB Password**: SAP Business One company database's password required as part of SQL server authentication.
- **DB Server Type**: Server type of SAP Business One company database.

Figure 2: DB Server Type drop-down box

- **License Server**: Machine name of SAP Business One License server along with port number of sub license server (for example, localhost:30000).

- **Language**: Language of SAP Business One data.

Figure 3: Language drop-down box

- **Use Trusted**:
  - **yes**: To use windows authentication in SQL server.
  - **no**: To use SQL server's own login credentials for authentication.
- **Select Object Type**: Business object type on which the operation has to be performed.
- **Select Action**: The operation which has to be performed on the selected business object type.

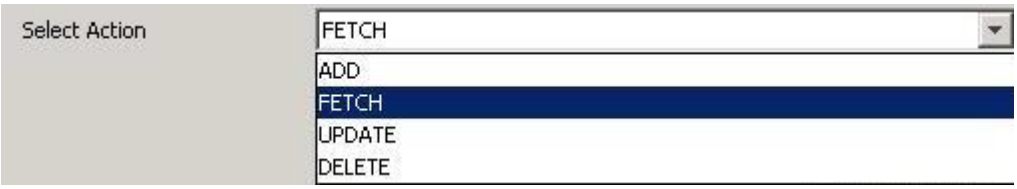


Figure 4: Action drop-down box

4.1.13.2 Testing

After SAPB1 component is configured, the connection to the SAP Business One company database can be tested using the Test Connection button.

4.1.13.3 Functional Demonstration

4.1.13.3.1 Scenario 1

Add a new Price List to SAP Business One company database and displaying the response.

Configure the SAPB1 component as described in section 4.1.13.1 Configuration and use feeder component configured with the schema fetched from the connected port of SAPB1 to send input request and use display component to check the response.

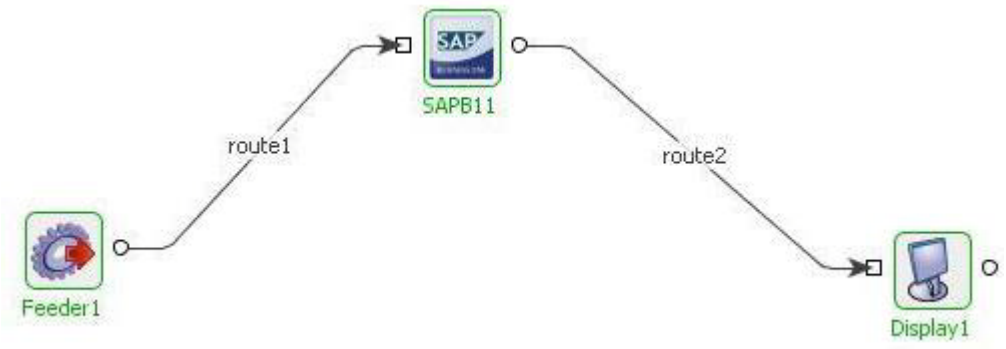


Figure 5: Demonstrating Scenario 1 with sample input and output

4.1.13.4 Sample Input

```
<?xml version="1.0" encoding="UTF-16"?>
<BOM>
  <BO>
    <AdmInfo>
      <Object>6</Object>
    </AdmInfo>
    <OPLN>
      <row>
        <ListName>MARGIN Price</ListName>
        <BASE_NUM>1</BASE_NUM>
        <Factor>1.000000</Factor>
        <RoundSys>0</RoundSys>
        <GroupCode>1</GroupCode>
      </row>
    </OPLN>
  </BO>
</BOM>
```

4.1.13.5 Sample Output

```
<ns1:Return xmlns:ns1="http://www.fiorano.com/SAPB1">
  <Key>9</Key>
  <Type>S</Type>
  <MessageCode></MessageCode>
  <Description></Description>
</ns1:Return>
```

Returned response includes key of the newly added business object, type S (Success) or E (Error) according to whether operation is successful or not. If the operation fails, corresponding Message code and description is returned.

4.1.14 SalesForce

This component simplifies the process of connecting to salesforce.com's Web service (SOAP) API.

The component takes the salesforce credentials as part of its configuration. This is used to get the session ID which is required to perform operations on SalesForce API. If the session expires, the component fetches a new session ID. Most of the properties used by the component like the ones for Basic authentication, SOAP compression are similar to WebService Consumer component. For detailed description of those properties, please refer to help WebServiceConsumer 4.0 component.

Configuration:

Connection	
Load WSDL From	**File**
SalesForce WSDL File	/home/ramesh/wsdl/FlightpriceS...
Error handling configuration	com.fiorano.edbc.framework.ser...
Store imported schemas	yes

Figure 1: Connection Configuration

4.1.14.1 Connection Configuration

- **Load WSDL From**: Source for WSDL can be configured here. It can be a File or a valid WSDL URL. When File is selected, the WSDL file location can be configured using the property SalesForce WSDL File. When URL is selected, the WSDL URL can be configured using the property SalesForce WSDL URL.
- **SalesForce WSDL File**: This property is used to specify the SalesForce WSDL File location. This property gets enabled when the Load WSDL From is selected as File.
- **SalesForce WSDL URL**: This property is used to specify SalesForce WSDL URL location. This property gets enabled when the Load WSDL From is selected as URL.
- **Store Imported Schemas**: If selected yes, saves imported schemas to Schema repository

Call and Addressing	
WSDL Service	SforceService
WSDL Port	Soap
Login Operation	login
Login Parameters	click ... to see the details.
Logout Operation	logout
Operations	click ... to see the details.
Call Properties	click ... to see the details.
Endpoint Address	https://login.salesforce.com/services/Soa...
SOAPBody Namespace	
Enable WS-Addressing	no

Figure 2: Call and Addressing Properties

4.1.14.1.1 Call and Addressing Properties

- **WSDL Service**: This property specifies the WSDL Service name. This property will be automatically populated when one or more operations are selected while configuring the property Operations.

- **WSDL Port**: This property specifies the WSDL Port. This property will be automatically populated when one or more operations are selected while configuring the property Operations.
- **Endpoint Address**: This property specifies the Endpoint Address for the Web service operation. This Property will be automatically populated when one or more operations are selected while configuring the Property Operations.
- **Login Operation**: This property specifies the web service operation which is used to login to salesforce.com.
- **Login Params**: This property specifies the parameters which are needed to be passed to salesforce.com while logging. Parameters like username, password etc. can be specified here. While entering password details, the security token should be concatenated with it to make authentication successful.

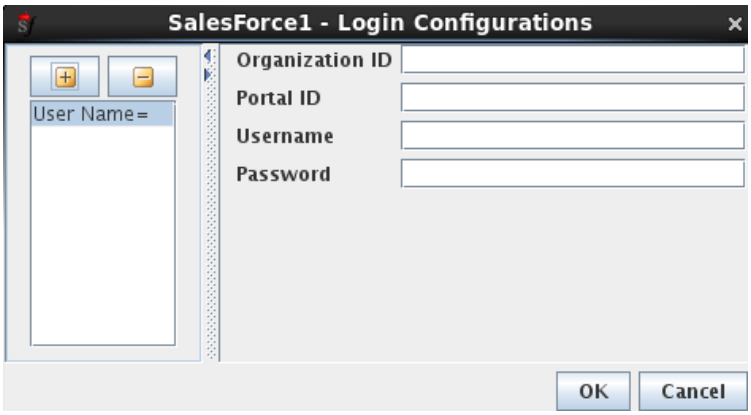


Figure 3: Login Parameters

- **Logout Operation**: This property specifies the web service operation which is used to logout from salesforce.com.
- **Operations**: Operations which need to be invoked on Salesforce API can be configured here. The operations supported by Salesforce will be shown when add button is clicked in the panel. In the panel beside, Operation can selected. Once the operation gets selected, the properties WSDL Service, WSDL Port, Endpoint Address, Input parameter and Output parameter will be populated.

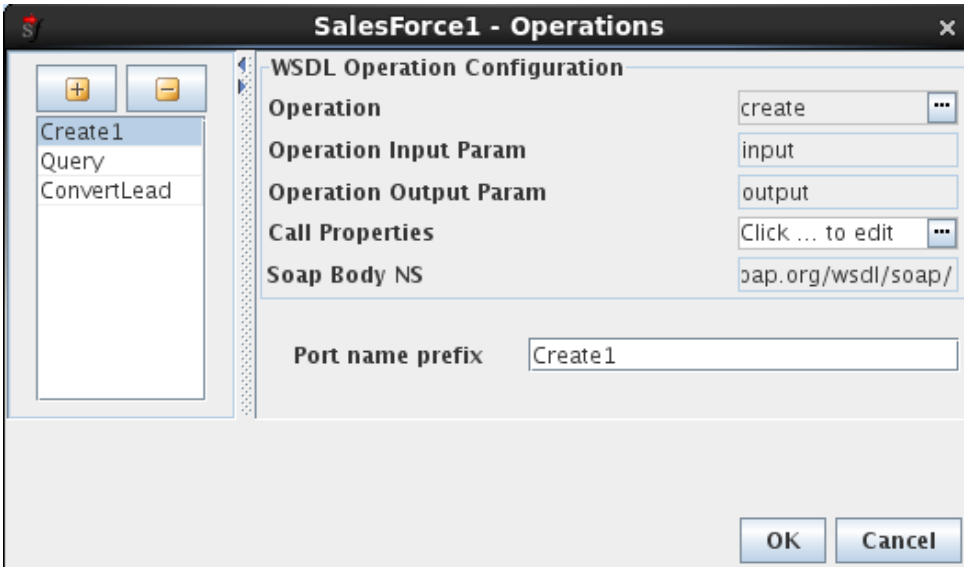


Figure 4: WSDL Operation Configuration

- **Port Prefix**: Displays param name of the operation to be invoked which is used as prefix to generate port names based on given param name. Input port name will be generated with "_Request" as suffix and Output port name will be generated with "_Response" as suffix. For param with name as "Create1", input and output port names will be Create1_Request and Create1_Response.
- **Input Parameter**: This property specifies the input message name for the selected WSDL Operation. Once the WSDL Operation is selected, this property will be automatically populated.
- **Output Parameter**: This property specifies the output message name for the selected WSDL Operation. Once the WSDL Operation is selected, this property will be automatically populated.
- **SOAPBody Namespace**: This property specifies the SOAP Body namespace. Once the WSDL Operation is selected, this property will be automatically populated.
- **Call Properties**: Advanced properties which can be used to optimize and change the behavior SOAP Invocation Call. The description for the axis call properties can be found at http://ws.apache.org/axis/java/apiDocs/org/apache/axis/client/Call.html

Useful Points to Note:

- If the salesforce web service is secured using basic authentication, then the details of the basic authentication should also be provided in the Call Properties property during execution time.
- While entering password details, the security token should be concatenated with it to make authentication successful.
- To pass http headers to the salesforce web service, the input message should contain properties with the header name prefixed with http_. Example http_Content-Type.
- Please refer the help of WebServiceConsumer 4.0 for other properties.
- In case of 'In Memory' launch, enable SSL settings in CPS and set 'Trust Store Location' to jre/lib/security/cacerts.

4.1.15 AS2Connector

AS2connector component serves as a connector to send/receive messages as per AS2 protocol.

AS2Connector can be configured in two modes **SEND** and **RECEIVE**. SEND mode to send encrypted/signed message to receiver. RECEIVE mode to decrypt and verify the signature of message received.

4.1.15.1 Configuration and Testing

4.1.15.1.1 Connection Configuration

Figure 1: Connection configuration details

Attributes

Send connection details from input: Parameters to create the connection can be specified in the input message when this property is set to true.

Connector Mode:

- **SEND**: When SEND mode is chosen, Send Configuration panel will appear as below.

Figure 2: Send configuration details

- **Send URI**: URI where the HTTP server is present.
- **RECEIVE**: When RECEIVE mode is chosen, RECEIVE Configuration panel will appear as below.

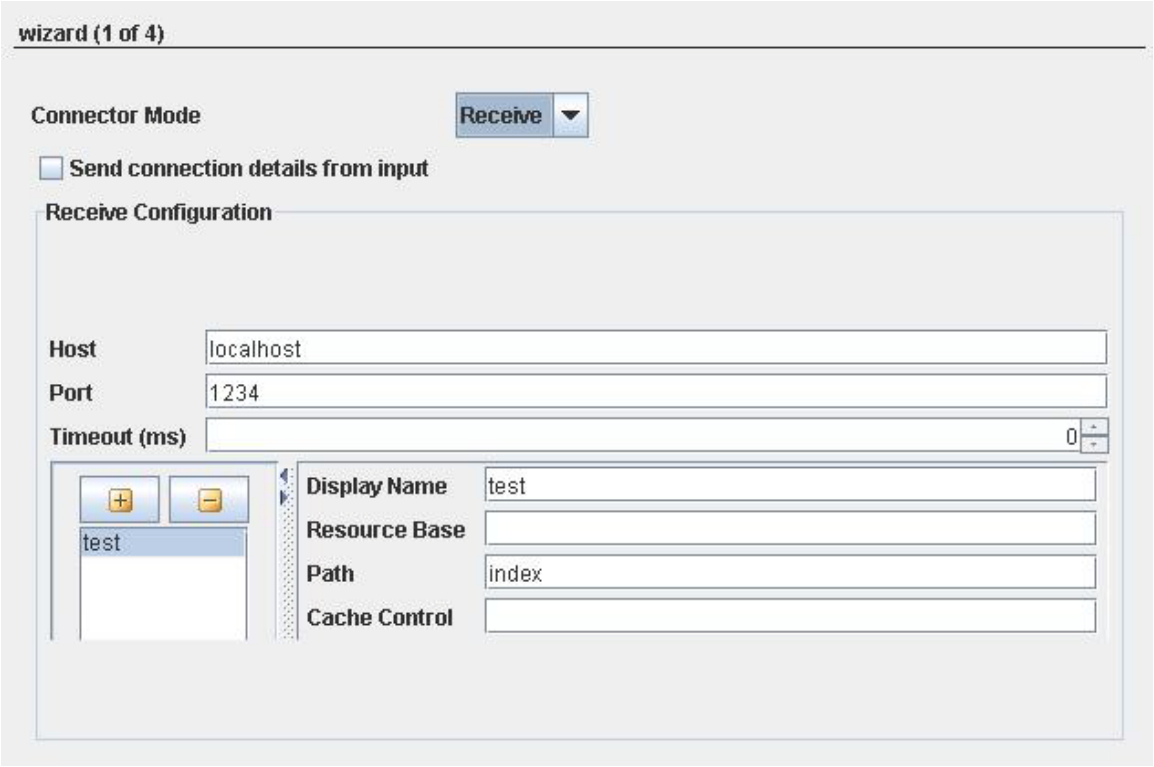


Figure 3: Receive configuration details

Host: The host name or the IP address where the server will be started.

Port: The port number on which server will run.

Timeout (ms): This Specifies the time for which this component will wait for acknowledgment.

Default value is 0 which specifies infinite timeout.

New server instance can be created by clicking the button in the receiver configuration.

Display Name: The unique display name for the context. The names of the ports that are generated corresponding to a context are suffixed by **__<Display Name>**.

Resource Base: Location of the resource that contains static content like images, css and so on. This can be any valid path in the file system (may also be relative to %FIORANO_HOME%\runtimedata\PeerServers\<profile-name>\FPS\run\components\AS2Connector\4.0) or a web URL. The resources present in resource base will then be hosted in URLs of the form http://<hostname>:<port>/<contextpath>/<resource>

Path: The unique context path of the AS2 request corresponding to a context. The path of the context will be computed relative to the server details provided by properties Host Name and Port.

The URI of a context will be http://<Host Name>:<Port>/<Path>.

Cache Control: The cache control mechanism that is to be used to cache the static content. For example, cache control value "max-age=3600,public" states that static content would be cached for up to an hour and shared between all users, without checking the server.

For valid Cache-Control values, please refer to http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9

4.1.15.1.2 AS2 Configuration

Encryption Configuration

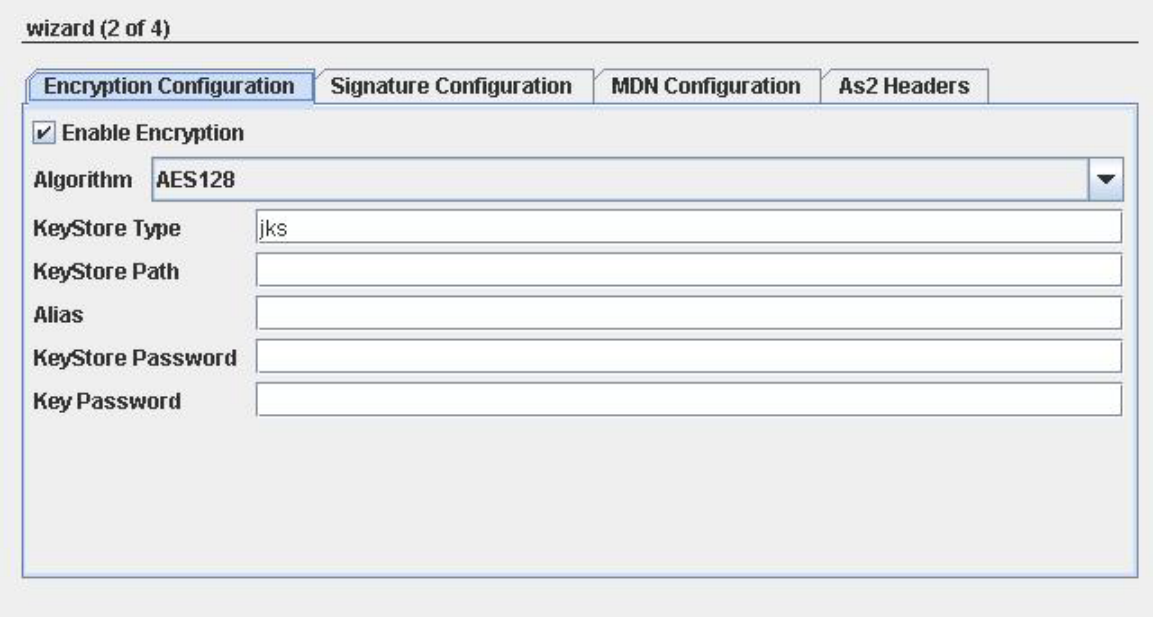


Figure 4: Encryption configuration details

Enable Encryption: When this option is selected, message will be encrypted in case of SEND and decrypted in case of RECEIVE with the provided algorithm and Keystore details.

Signature Configuration

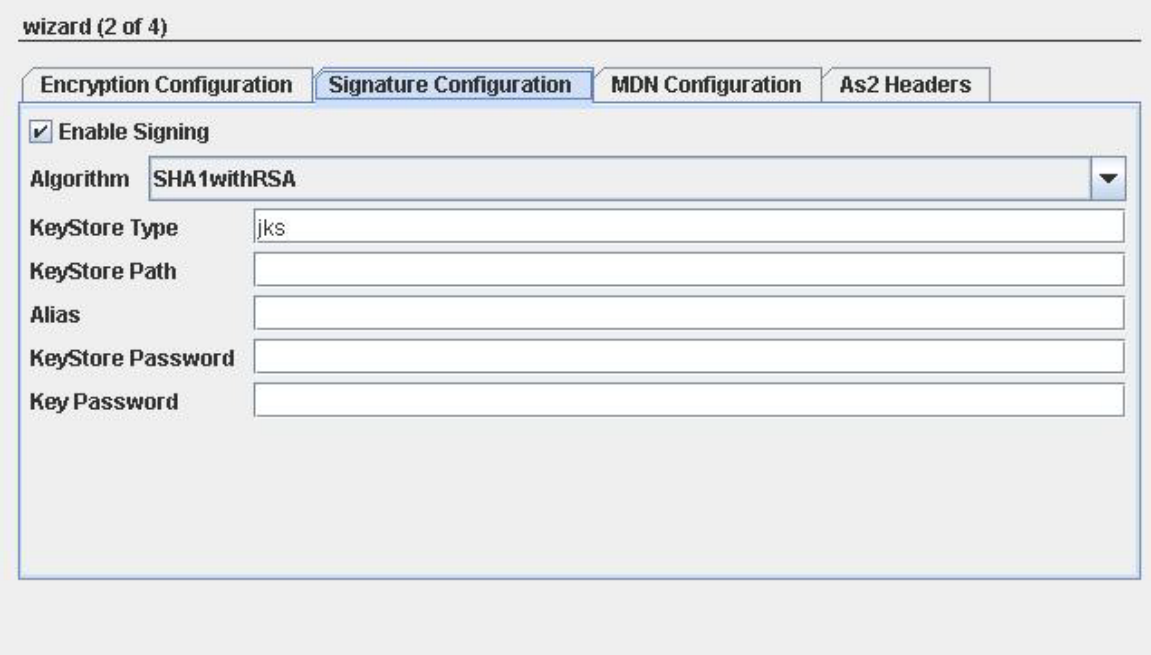


Figure 5: Signature configuration details

Enable Signing: Select to sign or verify signature of a message.

Algorithm: Algorithm to be used for signing.

MDN configuration

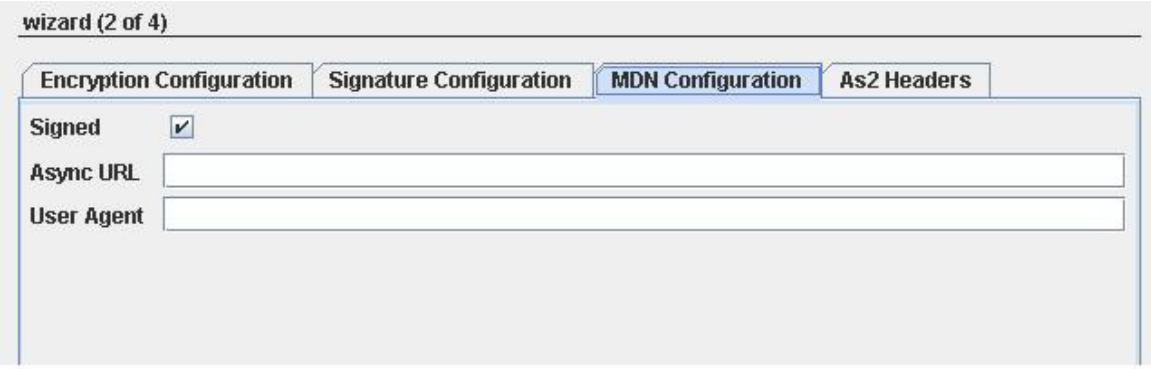


Figure 6: MDN configuration details

MDN Configuration – In RECEIVE mode, this configuration specifies the details of Message Disposition Notification sent back to the sender.

Signed: Select this for signing MDN, which will set the **disposition-notification-options** AS2 Header for default value. signed-receipt-protocol=required,pkcs7-signature; signed-receipt-micalg=required,sha1,md5.

Async URL: Url where the MDN should be sent to for Asynchronous MDN. The provided URL will be set as Receipt-Delivery-Option AS2 header.

User Agent: User which processes the AS2 request.

AS2 Headers

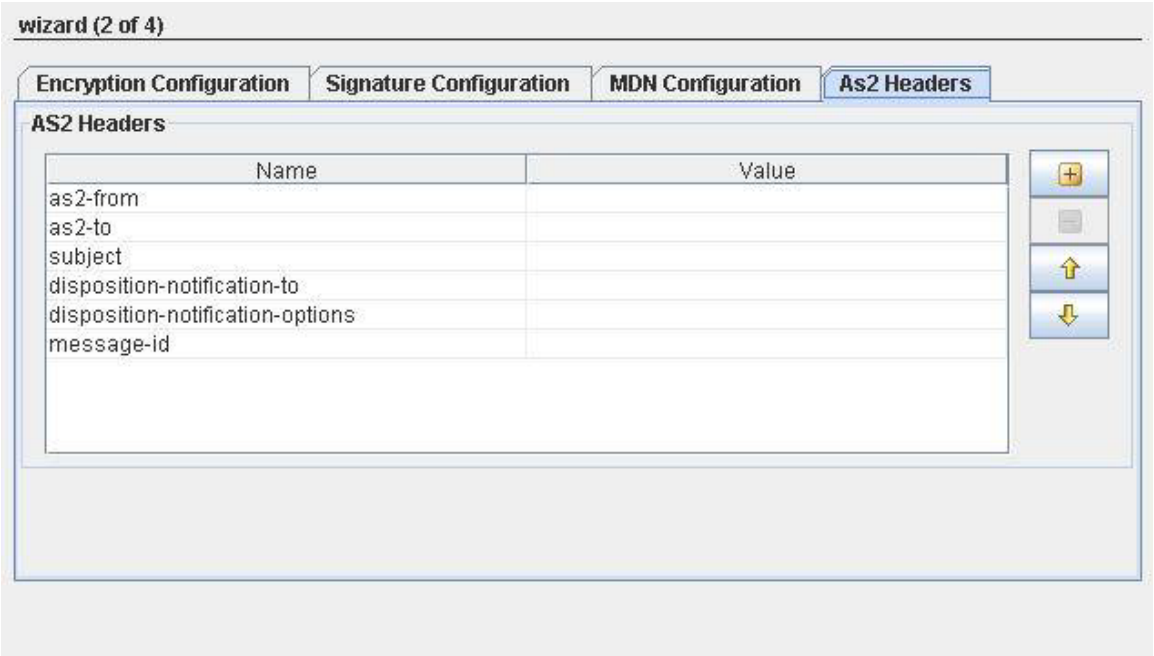


Figure 7: AS2 Headers details

Some of the AS2 headers are added by default. User can add/remove headers as necessary.

- **as2-from**: Sender of AS2 message.
- **as2-to**: Receiver to whom the AS2 message is sent to.
- **subject**: Specifies the subject of the message.
- **Disposition-notification-to**: If present MDN will be returned..
- **disposition-notification-options**: MDN options such as signing.
- **message-id**: Identifier of message.

4.1.15.2 Input and Output

4.1.15.2.1 Input

The input schema is auto generated for SEND mode based on whether Send connection details from input option is selected or not.

When it is selected, input schema is generated as shown in figure and sample input XML is shown in figure 8 and figure 9.

XML Item	Value Type
⊟ <ns1:AS2ConnectorInput>	
⊟ <AS2ConnectorConfiguration>	
⊟ <ConnectionConfiguration>	
<URL>	#string
⊟ <AS2Configuration>	
⊟ <AS2Headers>	
<as2-from>	#string
<as2-to>	#string
<subject>	#string
<disposition-notification-to>	#string
<disposition-notification-options>	#string
<message-id>	#string
⊞ <Header>*	
⊟ <EncryptionConfiguration>	
⊟ <EncryptionSource>	
⊟ [Choice]	
⊟ <KeyStore>	
<KeyStoreType>	#string
<KeyStorePath>	#string
<Alias>	#string
<KeyStorePassword>	#string
<KeyPassword>	#string
@SourceType	#string
<Algorithm>	#string
⊟ <SignatureConfiguration>	
⊟ <EncryptionSource>	
⊟ [Choice]	
⊟ <KeyStore>	
<KeyStoreType>	#string
<KeyStorePath>	#string
<Alias>	#string
<KeyStorePassword>	#string
<KeyPassword>	#string
@SourceType	#string
<Algorithm>	#string
⊟ <MDNConfiguration>	
<Signed>	#boolean
<AsyncURL>	#string
<UserAgent>	#string
⊞ <Payload>?	

Figure 8: Input Schema when Send connection details from input option is selected

```
<ns1:AS2ConnectorInput xmlns:ns1="http://www.fiorano.com/as2connector/AS2Connector2">
   <AS2ConnectorConfiguration>
      <ConnectionConfiguration>
         <URL>http://localhost:1234/index</URL>
      </ConnectionConfiguration>
      <AS2Configuration>
         <EncryptionConfiguration>
            <Algorithm>AES128</Algorithm>
         </EncryptionConfiguration>
         <SignatureConfiguration>
            <Algorithm>SHA1withRSA</Algorithm>
         </SignatureConfiguration>
         <MDNConfiguration>
            <Signed>false</Signed>
         </MDNConfiguration>
      </AS2Configuration>
   </AS2ConnectorConfiguration>
   <Payload>
      <Part>string</Part>
   </Payload>
</ns1:AS2ConnectorInput>
```

Figure 9: Sample Input XML when Send connection details from input option is selected

When it is not selected, input schema is generated as shown in figure and sample input XML is shown in figure 10 and figure 11.

XML Item	Value Type
⊟ <ns1:AS2ConnectorInput>	
⊟ <Payload>?	
⊟ <Part>*	
@file	#string
@ContentType	#string
#string	

Figure 10: Input Schema when Send connection details from input option is not selected

```
<ns1:AS2ConnectorInput xmlns:ns1="http://www.fiorano.com/as2connector/AS2Connector2">
   <Payload>
      <Part file="file">string</Part>
   </Payload>
</ns1:AS2ConnectorInput>
```

Figure 11: Sample Input XML when Send connection details from input option is not selected

For all other cases no schema is set on input/output ports.

4.1.15.3 Functional Demonstration

4.1.15.3.1 Scenario 1

Sending AS2 message request and receiving AS2 message.

Configure one AS2connector in **SEND** mode and another in **RECEIVE** mode as described in *Configuration* section. Use a Feeder component to send the message part to AS2Connector configured in send mode and use a Display component to check the response as shown in figure 12.

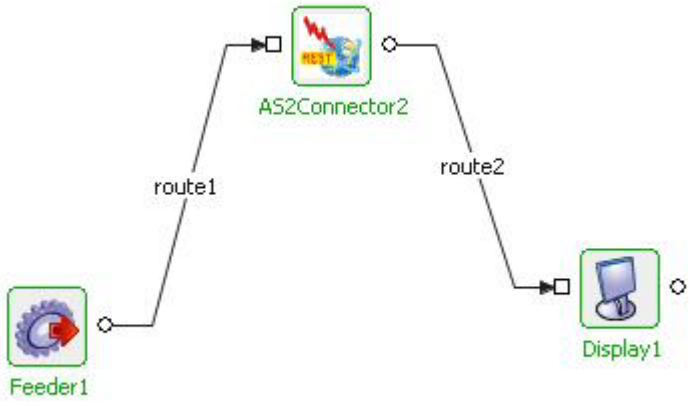


Figure 12: Sample Event Process demonstrating SEND mode.

Connect a Display component to the output port of AS2Connector configured in **RECEIVE** mode and also the output port of Display to the response port of AS2Connector as shown in Figure 10.

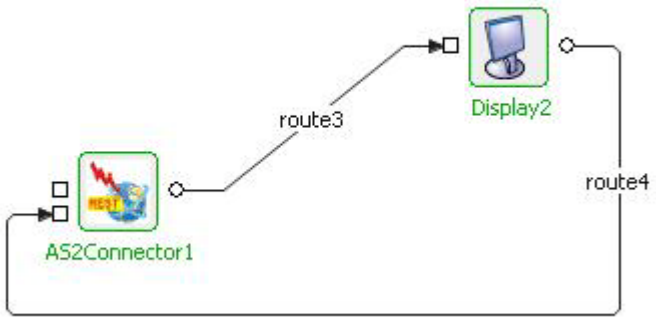


Figure 13: Sample Event Process demonstrating RECEIVE mode

4.1.16 EBMSAdapter

The EBMSAdapter is used for the EB Message Service protocol. Using EBMSAdapter business information can be exchanged in a secure and reliable way between the host and the partner. Messages can be digitally signed and sent encrypted. Additionally successful receipt of messages can be confirmed by returning an acknowledgment.

4.1.16.1 Configuration

The component can be configured using the custom property sheet as shown in the following sections.

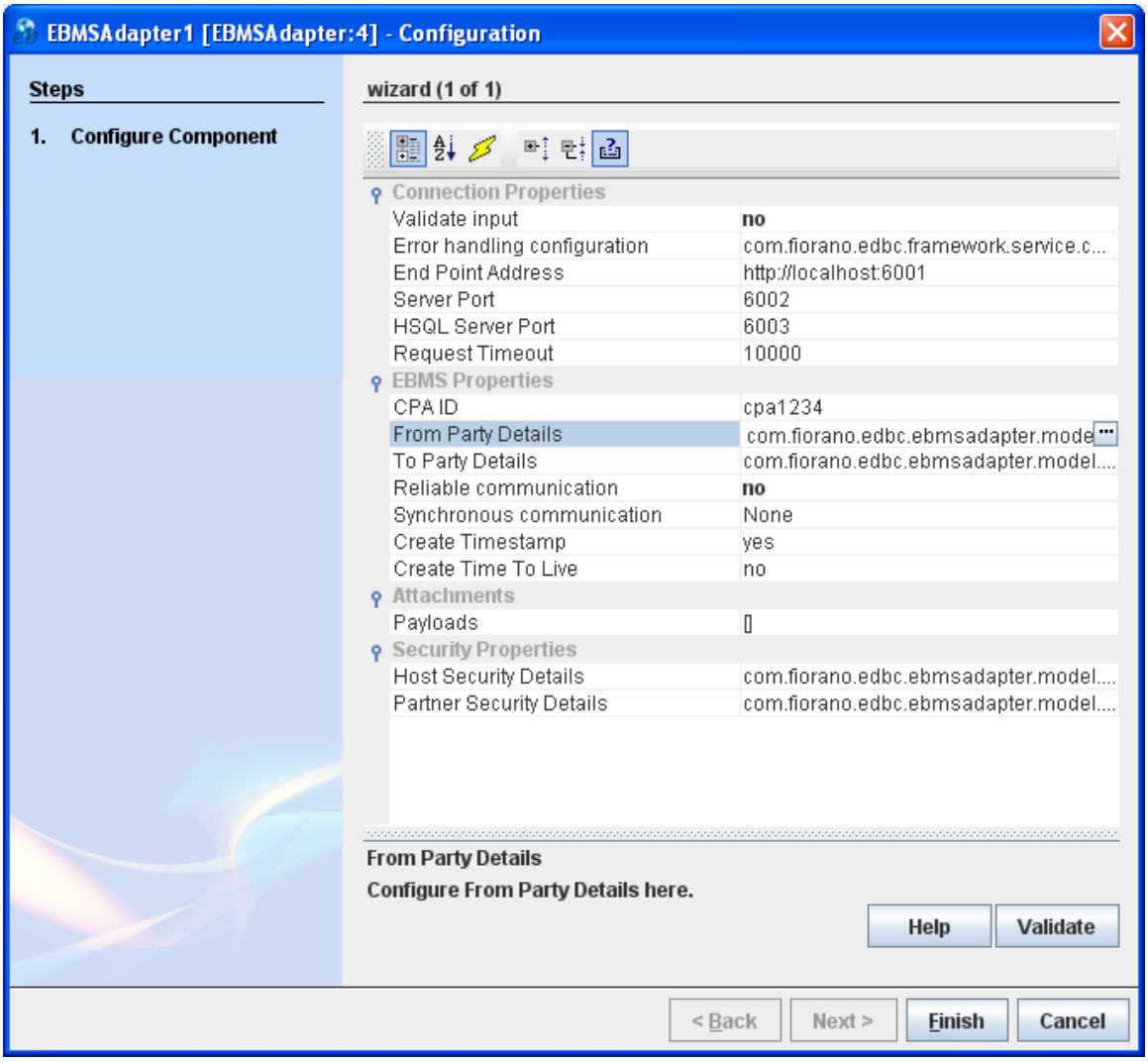


Figure 1: Configuration Property Sheet

This page is used to configure all the connection details,security details, agreement details and attachments used by the host and partner, to be sent using this EB Message Service protocol.

Configuration Property Sheet consists of following properties

4.1.16.1.1 Connection Properties

End Point Address: Partner's URL on which EB MSH (message service handler) is listening, URL to which the request should be sent.

Server Port: Details of the port on which HOST EB MSH runs,the request processing server listens

HSQL Server Port: The details of the port on which the HSQL server should run.

Request Timeout(in ms): TimeOut for HTTPConnection. If no response is received, on timeout Exception message is sent.

Authentication Mode: It supports BASIC authentication for both host and partner. When BASIC authentication is selected, username and password is required.

4.1.16.1.2 Expert Properties

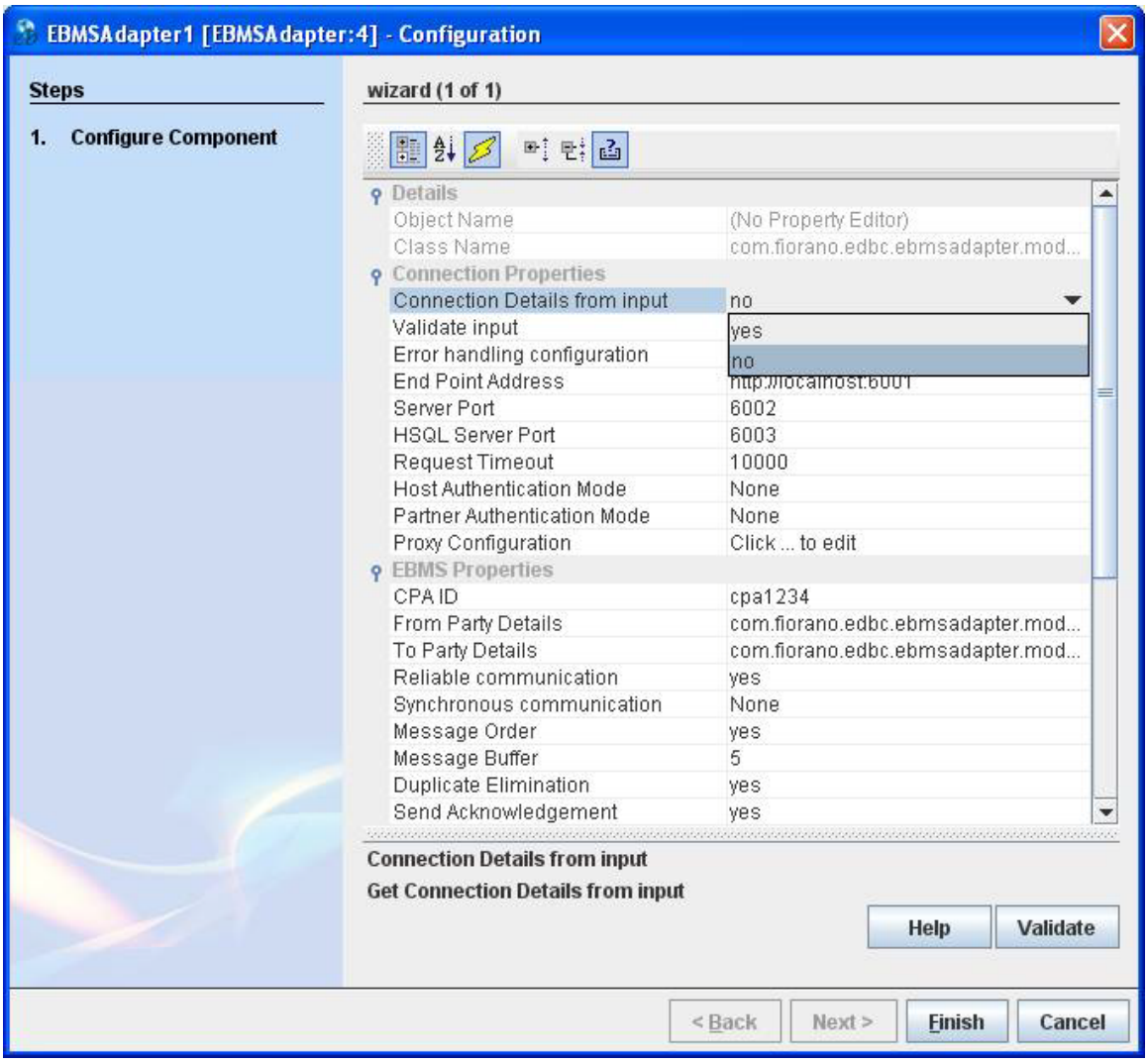


Figure 2: Configuration Property Sheet with expert properties

Connection Details From Input: Connection Details along with agreement details are taken from input port/message, cps configuration values are overridden.

Proxy Configuration: Proxy configuration values should be provided

4.1.16.1.3 EBMS Properties

CPA ID: CPA ID describes the formal relationship between the host and partner.It identifies the parameters governing the exchange of messages between the parties.The recipient of a message must be able to resolve the CPA ID to an individual set of parameters, taking into account the sender of the message.Simply said, it is just an arbitary and mandatory string that can be used to identify sender and recipient.

From and To Party Details: The 'From Party Details' property identifies the Party that originated the message. The 'To Party Details' property identifies the Party that is the intended recipient of the message. The PartyDetails property has PartyID, Type and Role. The type indicates the domain of names to which the string in the content of the PartyID element belongs. The value of the type MUST be mutually agreed and understood by each of the parties. The Role property identifies the authorized role (from or to) of the Party sending (when present as a child of the From element) and/or receiving (when present as a child of the To element) the message.

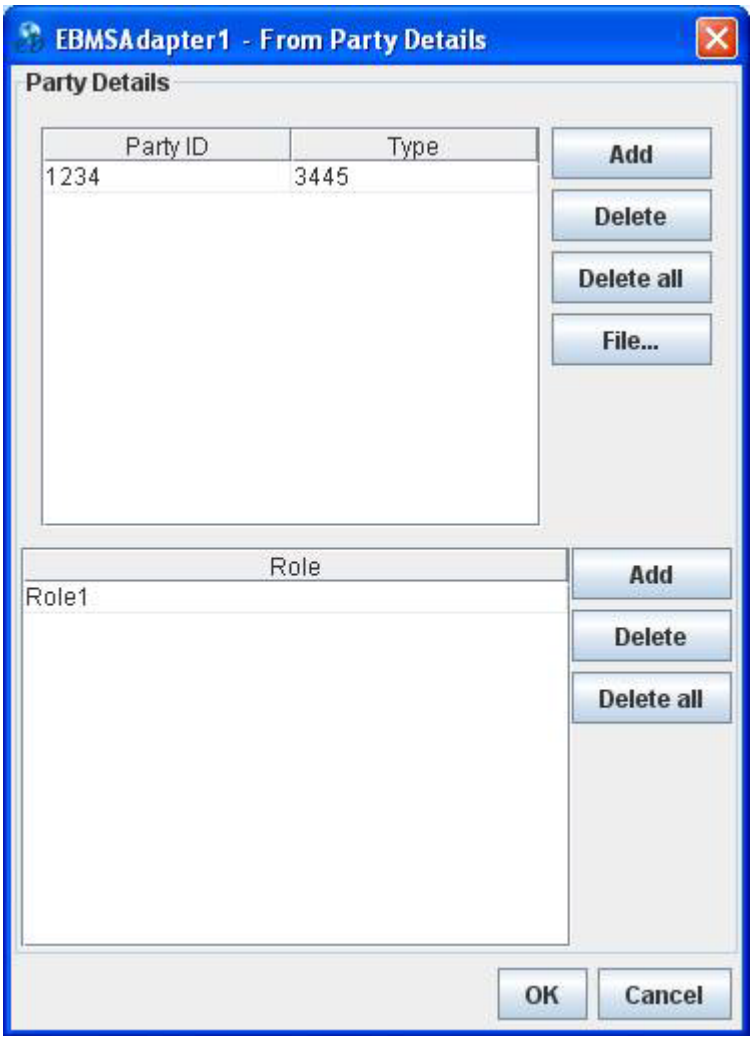


Figure 3: From Party Details

Reliable Communication:

(yes/no) - If Reliable communication is true, it enables following properties:

Duplicate Elimination: If Duplicate Elimination is enabled to true, it will eliminate the duplicate responses. When Message Order is enabled, Duplication Elimination is mandatory.

Send Acknowledgement: If true, then Acknowledgement will be sent for synchronous Request.
Retry Count: the maximum number of times request is retried if ack is not received
Retry Interval(in ms): The interval for which request will be retired.

Send Acknowledgement to outport: If true, Received acknowledgements from partner are sent to output port.

Persist Duration(in ms): Specifies the minimum length of time to keep the EB message in persistent storage(HSQLDB).

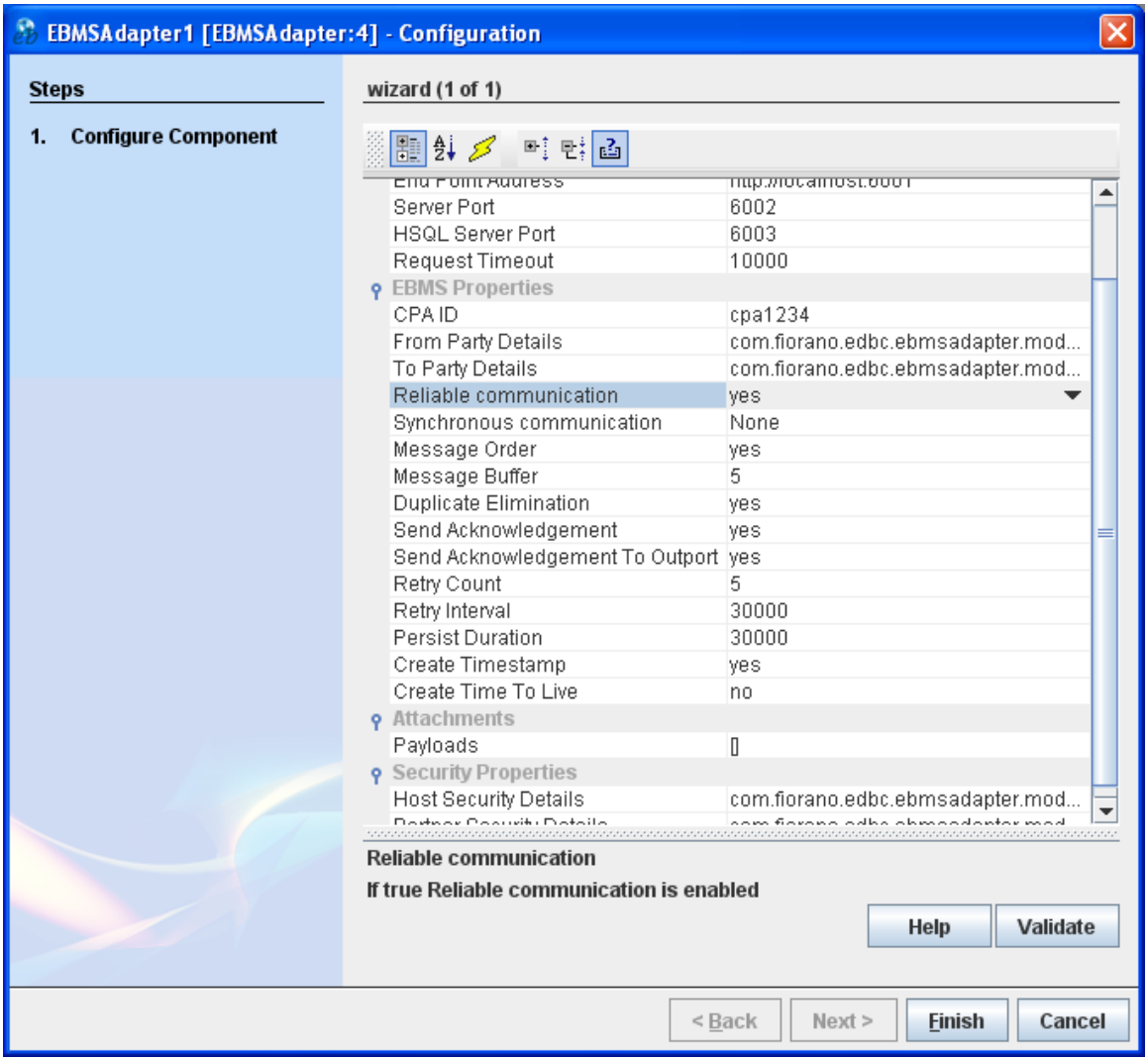


Figure 4: With Reliable Communication enabled

Synchronous communication: The sender of an EB message can use a synchronous communications protocol to receive the associated response message over the same connection request.For synchronous reply,use following options

9. **responseOnly** - any mshSignals along with business responses are received on same connection request.
10. **None(asynchronous)** - business responses are received asynchronously.

If Message Order will be available only in Asynchronous Communication.

Message Order: If true, the recipient receives EB messages in the order of sending.
Message Buffer: It is used in case of Message Order.
If Messages are received out of sequence, messages are buffered (not sent to output port). If the number of messages are more than the buffer size,BUFFER_OVERFLOW message is sent to partner.

Create Timestamp: Add Timestamp in request.

Create Time To Live(in ms): Specify the interval in milliseconds for the request sent.
It is used to indicate the time,by which a message should be delivered to the To partner.If the time taken for message to deliver is more than the value of Time To Live,message expires.

Attachments

Payloads: Configure the attachments.

Security Details

Security Details for both host and counterpart can be configured here.
Set 'Sign Message' to indicate whether to require inbound messages to be signed.
Set Message Encrypted to indicate whether to require inbound messages to be encrypted. If requiring encryption:From the EncryptionMethod, select the required method.Present encryption methods:

11. DESede
12. AES-128
13. AES-192
14. AES-256

The acknowledgement is signed at the request of a partner and depends on the configuration of an identification credentials set.
From the 'Signature Algorithm' list, select the algorithm used to sign the outbound EB message. From the 'Canonicalization Method' list, select the algorithm to canonicalize the SOAP Envelope XML and exclude comments before signing outbound EB messages.

Figure 5: Host Security Details

Similarly, Counterparty's Security Details can be configured.

4.2 Collaboration

The Collaboration category consists of components like chat, csChat, vbChat, and vcChat. The following section describes each component.

4.2.1 Chat

Chat component is a simple JMS application used to send messages from one chat component to another through their input and output ports. The font and color of the messages can be configured in the CPS of the Java chat component.

Note: This component cannot be launched in-memory of the peer server.

4.2.2 C# Chat

CS Chat is a native Csharp based component. CS Chat uses the .Net framework for its GUI (chat window). The CS Chat component uses the Csharp runtime which is a wrapper on C++ native runtime for running the application.

Points to note

15. This component cannot be launched in-memory of the peer server.
16. This component doesn't support Configuration Property Sheet (CPS).

Note: To run csChat component, make sure DOT NET 2003 or above is installed on the machine where peer server is running.

4.2.3 VB Chat

Vbchat is built upon JNI based cpp client library which uses fmq-cpp-client-msg-adapter.lib for running the component. Vbchat uses Microsoft ActiveX controls for its GUI (chat window).

Points to note

- This component cannot be launched in-memory of the peer server.
- This component doesn't support Configuration Property Sheet (CPS).

Note: To run vbChat component, make sure DOT NET 2003 or above is installed on the machine where peer server is running.

4.2.4 VC Chat

VC Chat is a native C++ based component. VC Chat uses the MFC library for its GUI (chat window). VC Chat uses the c++ native runtime library for running the application.

Points to note

12. This component cannot be launched in-memory of the peer server.
13. This component doesn't support Configuration Property Sheet (CPS).

Note: To run vcChat component, make sure Microsoft Visual Studio 6.0 or above is installed on the machine where peer server is running.

4.3 DB

The DB component is an all encompassing powerful component which can be used to configure simple and nested queries like insert, update, delete and select. It can also be used to monitor tables by value, by reference, by using alter tables and by using stored procedures. Monitoring can also be used for loop detection in replicating databases. The graphic user interface of this component allows designing queries with the application of zero coding effort using the Design mode. However, the SQL mode can also be used to write queries. Syntactical validity of the SQL can be ensured by using the Check Syntax SQL button provided on the SQL configuration panel in the SQL mode.

DB component is capable of handling:

14. Query execution
15. Nested Query execution
16. Sequencing - Execution of a set of queries in a pre-defined order.
17. Stored Procedures execution
18. Failover Queries - Executed when an SQL Statement fails to execute due to an error.
19. Post Processing - SQL statements to be executed after every query or after a sequence of queries.
20. Table Monitoring – Data changes due to updates, inserts or deletes.
21. Customized Transactions
22. Customized Response Size
23. Advanced and Complex Data types including User Defined Data types.
24. DB component uses the JDBC 1.0 API.

4.3.1 DB

The DB component is an all encompassing powerful component which can be used to configure simple and nested queries like insert, update, delete and select. It can also be used to monitor tables by value, by reference, by using alter tables and by using stored procedures. Monitoring can also be used for loop detection in replicating databases. The graphic user interface of this component allows designing queries with the application of zero coding effort using the Design mode. However, the SQL mode can also be used to write queries. Syntactical validity of the SQL can be ensured by using the Check Syntax SQL button provided on the SQL configuration panel in the SQL mode.

The following are some silent features of the DB component:

- **Query execution** - Using this component, simple select, update, insert and delete queries can be executed.
- **Nested Query Execution** - The DB component provides one level of nesting for insert, update, delete, and stored procedures.
- **Grouping** - DB components support query grouping which is the execution of a set of queries in a pre-defined order.
- **Stored Procedures** - The DB component supports execution of Stored Procedures.

- **Failover Queries** - These queries are executed when an SQL Statement fails to execute due to an error. Failover queries maintain data consistency even in a case when an unexpected error while executing an SQL Statement is experienced by the component flow.
- **Post Processing** - SQL statements for post processing can be defined after a single query execution or after an execution of multiple queries.
- **Table Monitoring** - The DB component supports monitoring of simple and nested tables for data insertion, updation, and deletion. This component also supports monitoring for updation of selected columns. Multiple tables can be monitored using this component.
- **Customized Transactions** - The component can be configured to commit the entire transaction after a row, document, batch, or can be committed automatically.
- **Customized Response Size** - The response size for the output of the component can be configured. This allows the processing of multiple records in a single transaction. For example, if only 100 records should be processed in a transaction, it can be set using the Response Size field. This ensures that only 100 records are sent as part of one message. If there are 500 records, 5 responses are sent with 100 records in each.
- **Support for Advanced and Complex Data types** - The DB component supports BLOB, CLOB, User Defined Data types (UDTs) and different date-time formats.

Points to note

- It is recommended **NOT** to use JDBC-ODBC Bridge driver to connect to any RDBMS in your production environment. Please use a commercial JDBC driver instead.
- The JDBC drivers or the resources must be directly added onto the JDBC system lib and not as resource to the DB component itself. To add JDBC drivers to DB component, please refer to section 3.3.3.3.

4.3.1.1 Database Connection Configuration

Connection details are configured in the first panel, which is **Managed Connection Factory - MCF** of **Configuration Property Sheet** - CPS. Figure 4.101 illustrates the panel with expert properties view enabled.

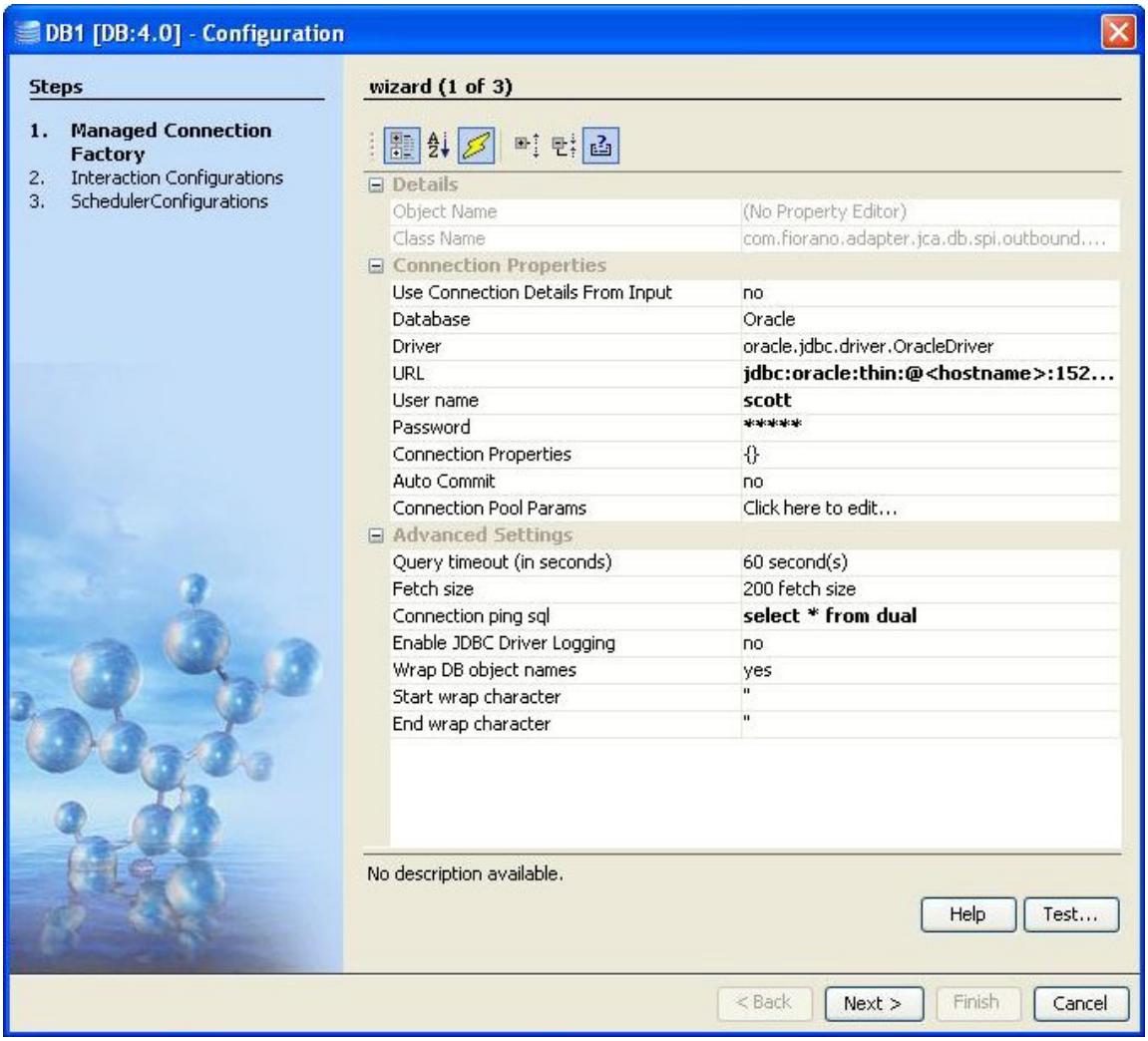


Figure 4.101: Managed Connection Factory Panel

- **Use Connection Details From Input**: Parameters to create the connection can be specified in the input message when this property is set to true. If this property is selected the validation errors in the managed connection factory panel of the CPS are treated as warnings. So user can bypass this step without giving valid configuration and complete the configuration of the component. If valid properties are not provided even in the input message exception will be thrown at runtime.
- **Database**: Select the appropriate database in the **Database** property, the drop-down list all the supported databases as shown in Figure 4.102. If the required database is not listed, select **Other** as the database option.

  Note: If a database is not listed in the drop-down list, this does not mean that the component will not work with the database. This only means:

- Monitor table feature will not be supported for the database
- Appropriate **Driver** and **URL** should be specified

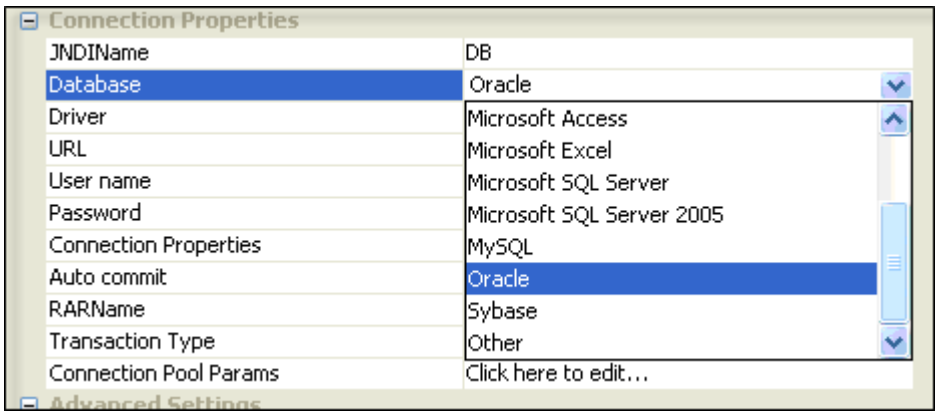


Figure 4.102: Database Drop-Down List

- **Driver**: Driver class name that should be used to connect to the database. On selecting required database, **Driver** value is populated with standard value (This can be changed to required values based on driver being used).

  Note: jar/zip file containing the driver class should be added as resource to **JDBC** System Library

- **URL:** URL at which the database is running. On selecting required database, **URL** value is populated with standard value (This can be changed to required values based on driver being used). The populated value will have place holders which have to be replaced to point to correct database location, for example, replace **<hostname>**, shown in Figure4.103, with appropriate IP address

- **User name:** Name of the user to connect to database as

- **Password:** Password for user

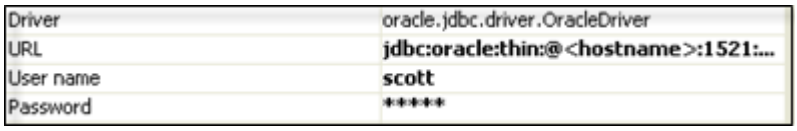


Figure 4.103: Driver, URL, User name, and Password Properties

- **Connection Properties**: Any driver specific connection properties which may have to be passed while creating a JDBC connection should be provided against **Connection Properties** (shown in Figure 4.104). For example, fixedString=true uses FIXED CHAR semantics for string values in oracle.

  Note: Please refer to documentation of driver that is being used for valid name-values for connection properties.

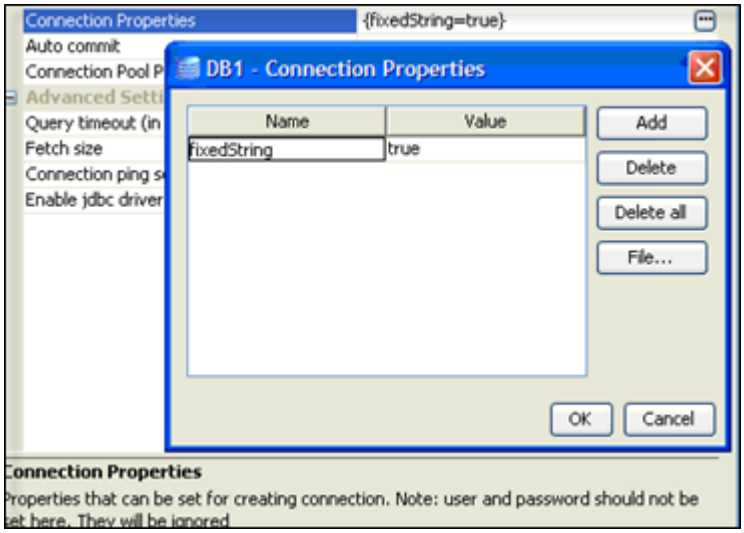


Figure 4.104: DB Connection Properties

- **Auto commit:** Commit mode that should be used by the JDBC connection.

	yes	**no**
Commit behavior	Transactions are committed implicitly	Transactions are committed explicitly
Database update	Happens instantaneously	Happens when the commit is called explicitly
Performance	Low	High (comparatively)
Granularity of transaction	Fixed. Every transaction is atomically committed	User defined. Granularity can be defined by specifying appropriate value for Commit Mode in Advanced Properties panel in the SQL configuration wizard (explained later)

- **Query timeout:** Time, in seconds (>= 0), after which an exception is thrown if the query execution is not complete. For example, if this value is set to 60 and a query to database does not return within 60 seconds, then an exception is thrown and query execution is stopped.
- **Fetch size:** Number of rows (>=0), which should be fetched from database into the component when iterating through result sets. This value provides a tradeoff between number of trips on networks and memory requirement.

  For example, a query results in 1000 rows and fetch size is set to 500, then result set gets all rows from database in two sets of 500 rows each.

  Note: If this value is set to 0, all the rows are returned in one turn.
- **Connection ping sql:** A SQL statement which is guaranteed to execute without exception, except when connection to database is lost. When a SQL exception occurs on executing a configured query, this SQL statement is executed. If execution of this SQL statement fails as well, then it is assumed that connection to database is lost and appropriate configured action (say, reconnect) is taken.

Example: select * from dual for **oracle, select 1** for **MS SQL**

- **Enable jdbc driver logging:** Value **yes** implies that logging at the driver level should be enabled. This is used as a debugging option.
- **Wrap DB object names:** When database object names (viz. table names, column names, schema names…) contain spaces, they should be wrapped in database dependent special characters. For example, **" "** for oracle and **[ ]** for excel.

  Database object names are wrapped as shown below:

  - **Start wrap character** + object name + **End wrap character**
  - **Start wrap character:** Character which should be used before the object name
  - **End wrap character:** Character which should be used after the object name

4.3.1.2 Interaction Configurations

SQL configuration details and advanced configurations are configured in the second panel (**Interaction configurations**) of CPS.

4.3.1.2.1 SQL Configuration

Click on next to **SQL configuration** property (shown in Figure 4.105) to launch the wizard (Figure 4.106) which allows configuring queries that have to be executed.

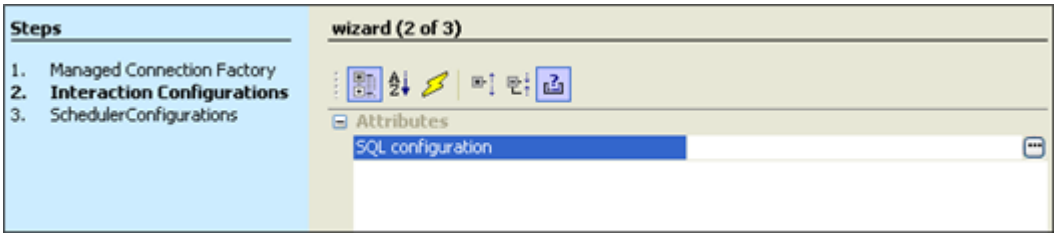


Figure 4.105: Interaction Configurations

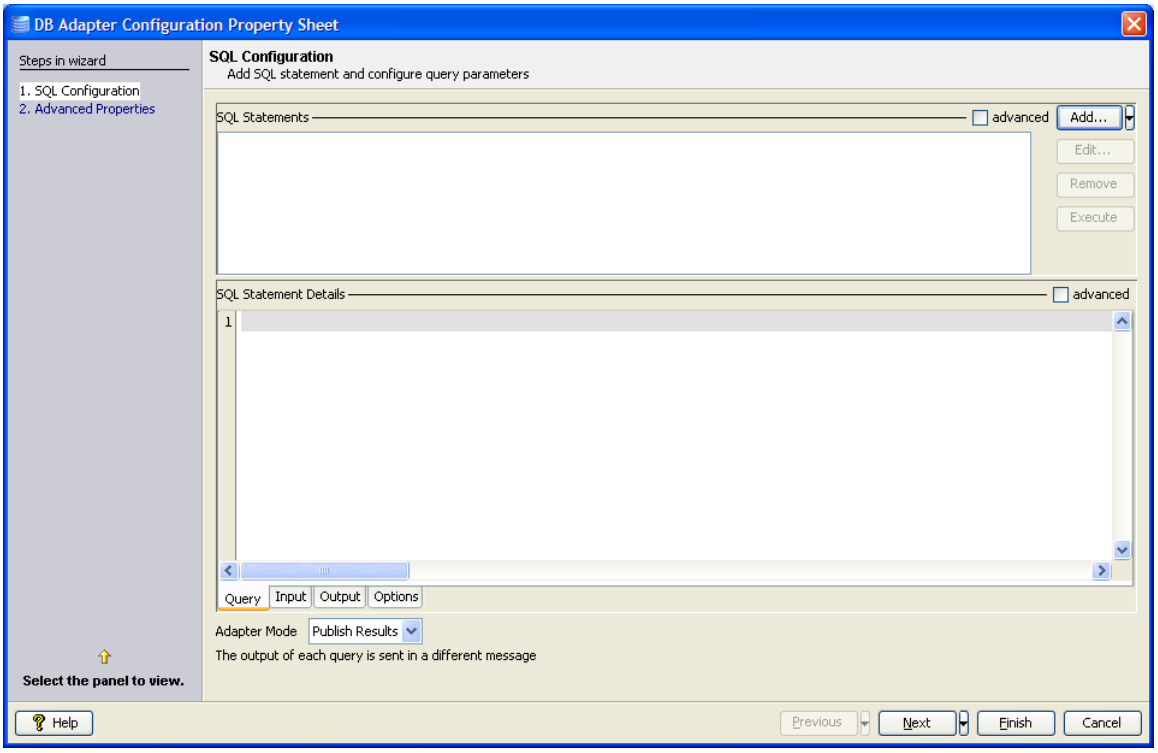


Figure 4.106: SQL Configurations Panel

SQL Configuration panel allows configuring multiple queries. To configure a query, click **button** and select required type of query from the pop-up menu.

4.3.1.2.1.1 Adding Query Configuration

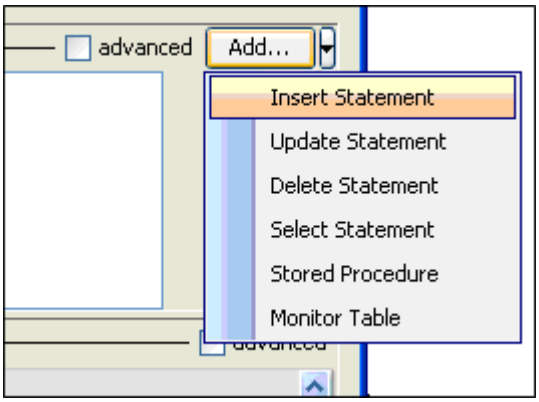


Figure 4.107: Adding a Query

Explanation for different types of queries is given in the following table:

Type of query	Explanation
Insert Statement	Inserts / adds data into database table
Update Statement	Modifies existing data in database table. This option also allows to configure upsert queries (explained later)

Type of query	Explanation
Delete Statement	Deletes data from database table
Select Statement	Retrieves data from database table
Stored Procedure	Executes stored procedure in database
Monitor Table	Checks for inserts / updates/ deletes on a table and reports them

4.3.1.2.1.2 Object Selection

Configuring queries requires selecting database objects on which actions have to be taken. Three kinds of objects are dealt within SQL configuration – tables, stored procedures, and user defined data types. UI for selecting objects are very similar in appearance and functionality. Table selection UI is shown in the Figure 4.108:

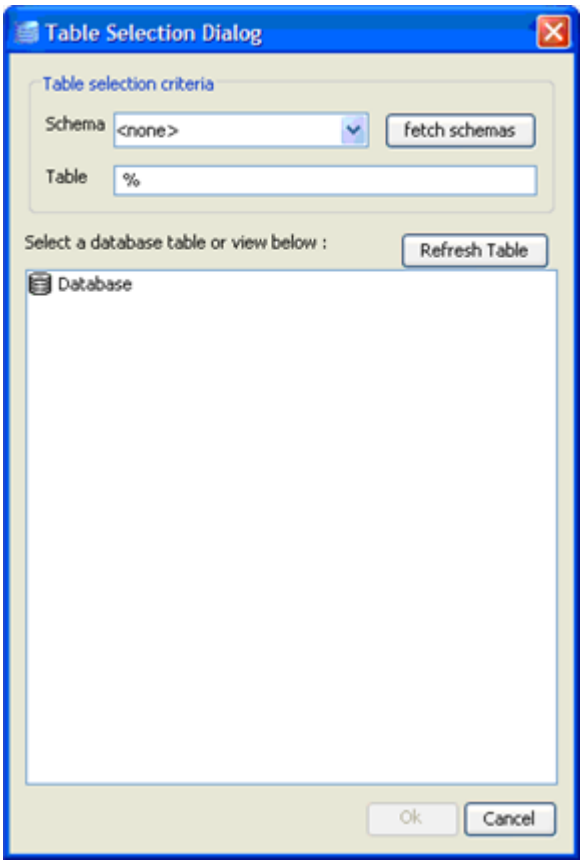


Figure 4.108: Table Selection Dialog

To select a database object, provide selection criteria (schema name pattern and object name pattern) in corresponding database object selection UI as shown in the Figure 4.109:

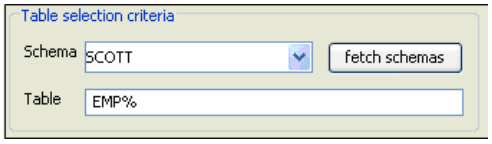


Figure 4.109: Table Selection Criteria

Schema/object name pattern may contain SQL wild cards:

- **%** - matches an number of characters
- **_** - matches one character

Example:S% - means all object names starting with S, **%S%** - means all object names containing S, and **_S%** – means all object names whose second character is S.

Schema name can either be typed or selected from drop-down list after clicking on **fetch schemas** button.

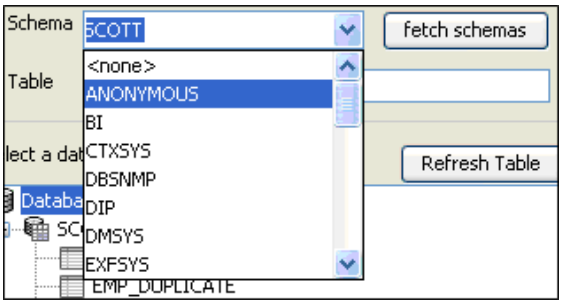


Figure 4.110: Selecting a Schema

Note: Select **<none>** to ignore schema while searching. Provide empty value to get objects without a schema.

Click on **Refresh <object>** (**Refresh Table**, for table selection) to fetch the list of objects matching the criteria specified. Result can be incrementally searched for appropriate value by typing in first few characters when the result tree is focus as shown in Figure 4.11.

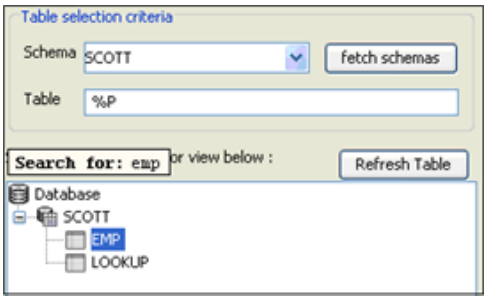


Figure 4.111: Searching a Table

Note: Response time for fetching required objects depends on search criteria, narrower the search criteria faster the response.

4.3.1.2.1.3 Insert Statement Configuration

Click on **Add...** → **Insert Statement** to launch **Insert Query Builder** (shown in Figure 4.112)

Note:

- Do not type in the text area before configuring the query.
- Do any modifications only after all other configurations are done.

- For user modified values, required input/output details (for example, data types) will not populated and these have to be configured manually.

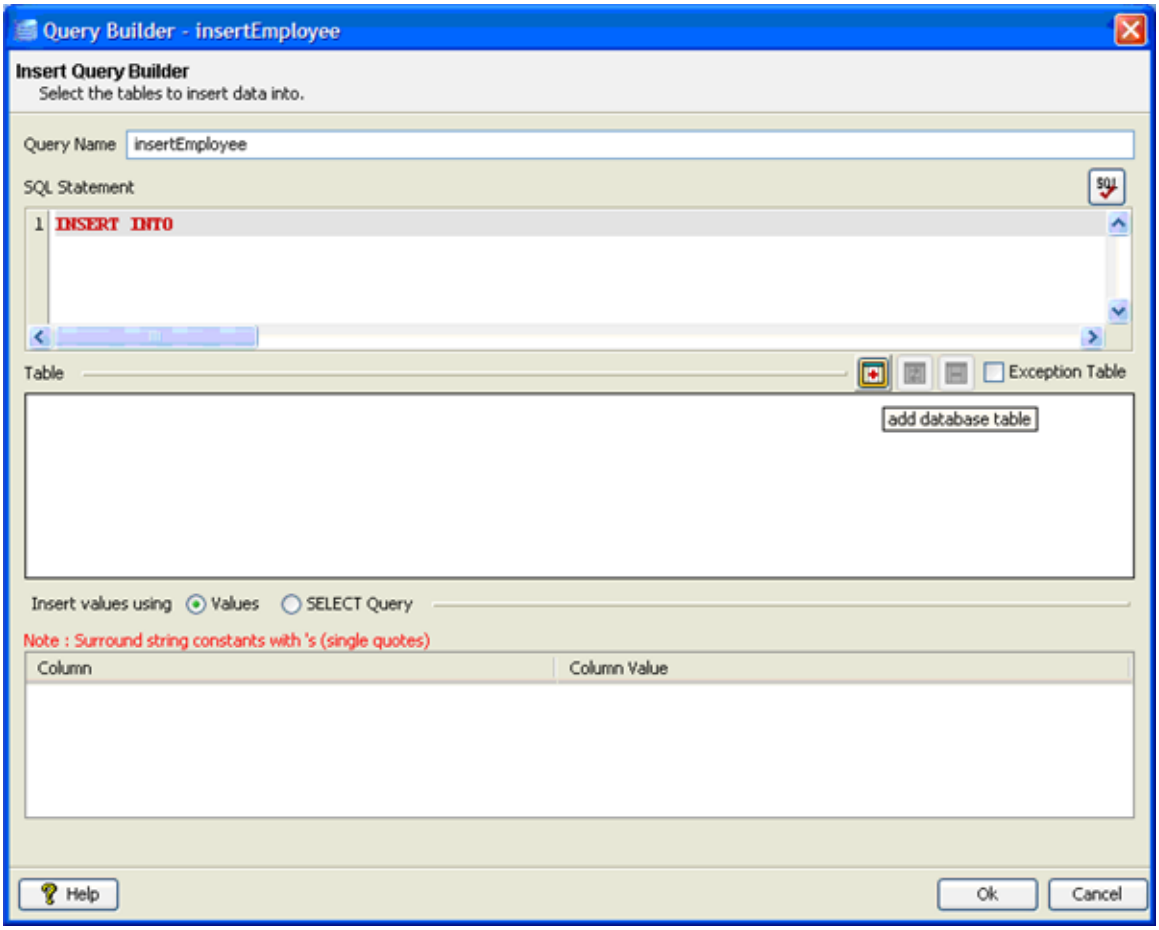


Figure 4.112: Insert Query Builder

Simple Insert Statement

Behavior: Inserts a row in configured table with column values taken from input XML or with constant column values.

1. Provide a name for the query against Query Name.
2. Click on (**add database table**) button to launch **Table Selection** dialog.
3. Select required table as explained in **Object Selection** section. Selected table is added to the easel under **Table**. Primary key column, if exists, is marked with adjacent to column name.

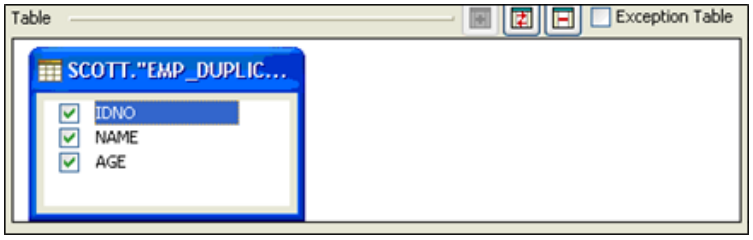


Figure 4.113: Selected table added to easel with all columns

4. Table can be changed by clicking (**replace selected table**) button and removed by clicking (**remove database table**) button.
5. If values are never to be inserted into a particular column, then that column can be unchecked (this requires column has a default value or supports null values) as shown in Figure 4.114.

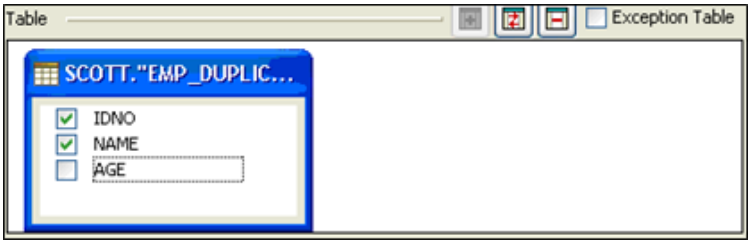


Figure 4.114: Ignoring column for insertion

6. To insert a constant value for a particular column, specify the required value in the **Column Value** column against the required column name.

Note:

- If the value is a string value it should be wrapped in single quotes (' ')
- ? indicates value is taken from input or from the output of another query where possible

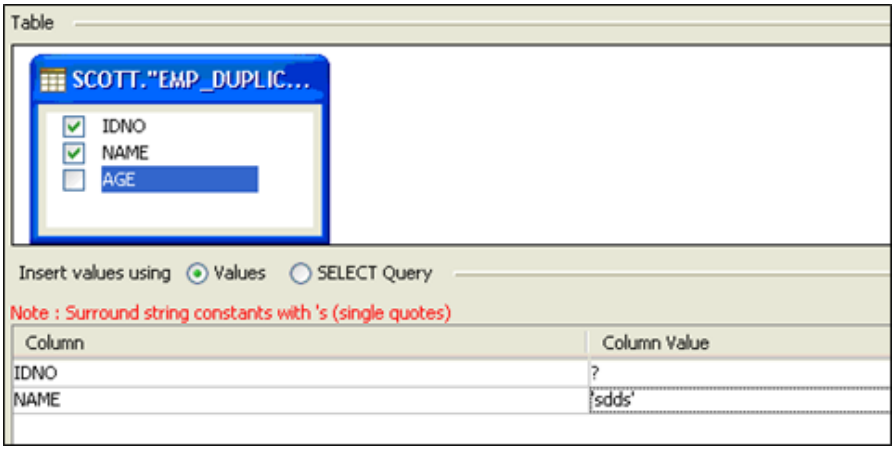


Figure 4.115: Inserting constant value into a table

Insert statement is automatically generated and shown in the text editor under SQL Statement Figure 4.116. The generated SQL can be validated by pressing (**check syntax**) button.

Note: This feature only checks for invalid tokens, it does not perform a complete syntax check

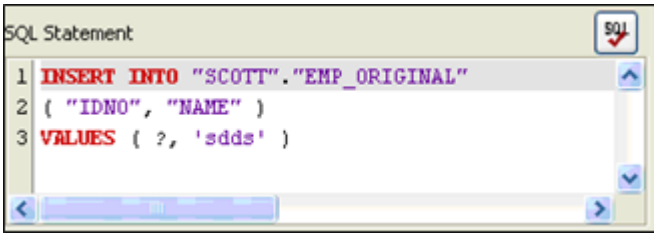


Figure 4.116: Generated insert query

7. Click **Ok** to close the dialog.

Insert Statement with Select

Behavior: Insert rows in configured table by selecting rows from another table.

1. Follow the steps from 1 to 6 as described in section Simple Insert Statement.
2. Select option **SELECT Query** against **Insert values using** as shown in Figure 4.117.

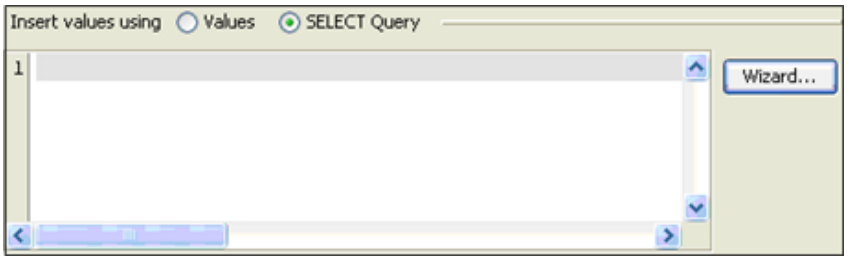


Figure 4.117: Option to insert values into a table using select query

3. Click on **Wizard...** button to launch **SELECT Query Builder**.
4. Follow the steps as described in section Select Statement Configuration.
5. Insert statement is automatically generated and shown in the text editor under **SQL Statement**. The generated SQL can be validated by clicking the (**check syntax**) button.

   Note: This feature only checks for invalid tokens, it does not perform a complete syntax check

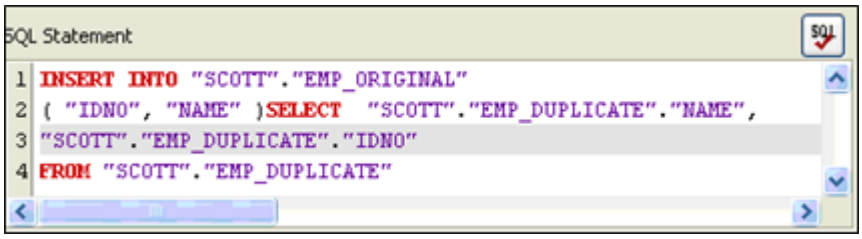


Figure 4.118: Generated query to insert values using select

6. Click **Ok** to close the dialog.

Insert Statement with failover

Behavior: Insert rows in configured table. If an exception occurs, insert in the exception table.

1. Follow the steps from 1 to 8 as described in section Simple Insert Statement.
2. Click on the check box **Exception Table**. Another **Insert Query Builder** launches, this is the table in which the values are stored which raised exceptions.

Figure 4.119: Exception Table Selection

3. Configure inserting into exception table following steps from one of the previous **Insert Statement** sections based on the requirement.
4. Click **Ok** to close the dialog.

4.3.1.2.1.4 Update Statement Configuration

Click on **Add...** → **Update Statement** to launch **UPDATE Query Builder** (shown in Figure 4.120)

Note:

- Do not type in the text area before configuring the query.
- Do any modifications only after all other configurations are done.
- For user modified values, required input / output details (for example, data types) are not populated and have to be configured manually.

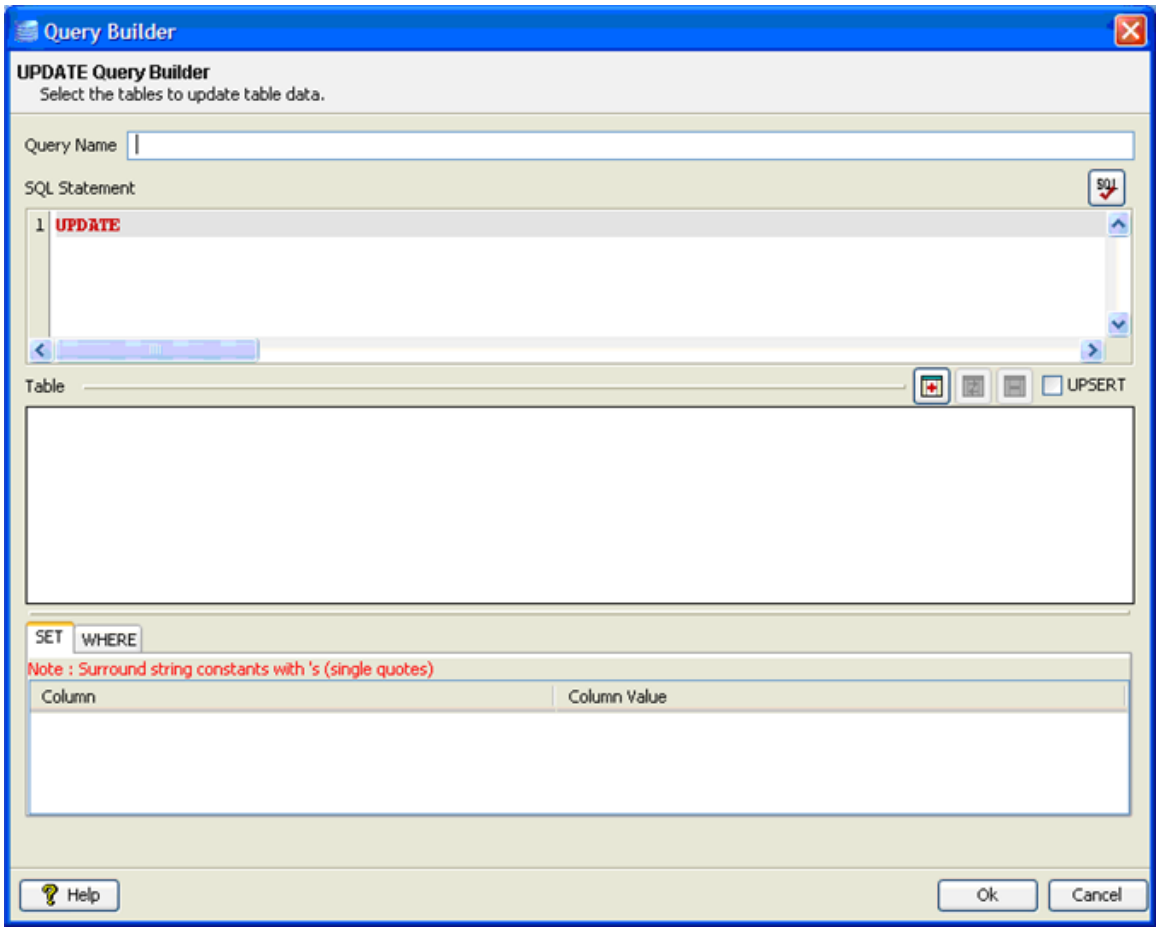


Figure 4.120: Update Query Builder

Simple Update Statement

Behavior: Update rows satisfying defined condition in configured table, with column values taken from input XML or with constant values. Condition values can also be taken from input XML or defined as constant values.

1. Provide a name for the query against Query Name.
2. Click on (**add database table**) button to launch **Table Selection** dialog.
3. Select required table as explained in Object Selection section.
4. Selected table is added to the easel under **Table**. Primary key column, if exists, is marked with adjacent to column name.

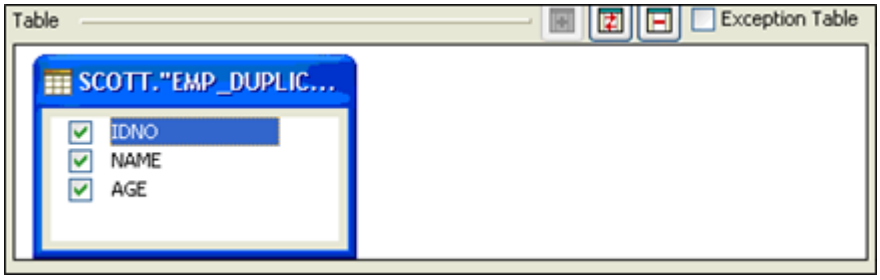


Figure 4.121: Selected table added to easel with all columns

5. Table can be changed by clicking on (**replace selected table**) button and removed by clicking on (**remove database table**) button.
6. Select the columns whose values have to be set (Figure 4.122 shows that NAME and AGE are selected for update)

Figure 4.122: Ignoring column for update

7. Selected columns are added under the **SET** tab.

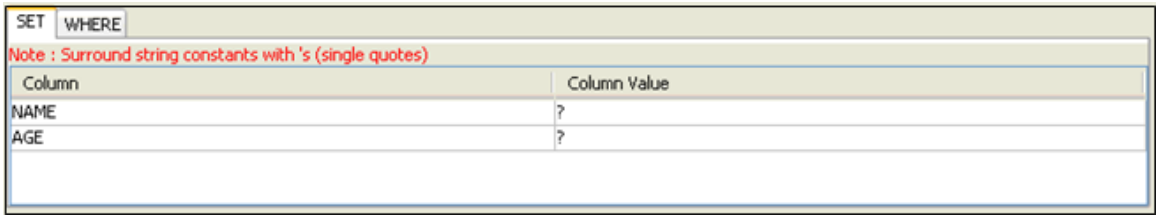


Figure 4.123: Columns added to SET clause

8. Click on **WHERE** tab and select a column name on which **where** condition has to be applied.

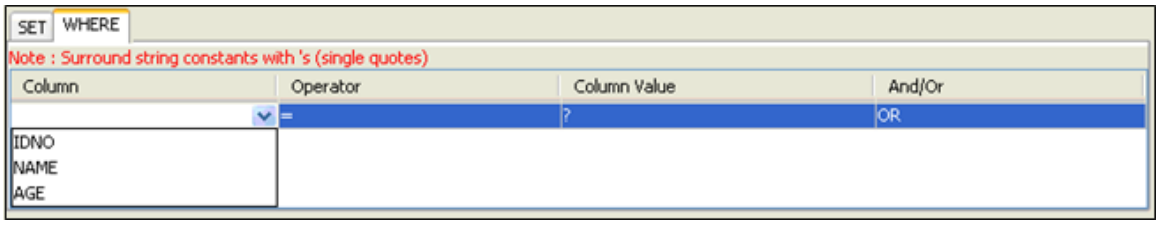


Figure 4.124: Adding condition on column to WHERE clause

9. When selecting multiple columns for **where** condition, conditions can be combined using **AND** or **OR** under **And/Or** column.

SET WHERE

Note : Surround string constants with 's (single quotes)

Column	Operator	Column Value	And/Or
IDNO	=	?	AND
NAME	=	?	OR
	=	?	OR

Figure 4.125: Specifying multiple conditions for WHERE clause

10. Operator of choice can be selected from the drop-down list under **Operator** column.

SET WHERE

Note : Surround string constants with 's (single quotes)

Column	Operator	Column Value	And/Or
IDNO	=	?	OR
	=	?	OR
	<		
	<=		
	>		
	>=		
	<>		
	BETWEEN		
	NOT BETWEEN		

Help Ok Cancel

Figure 4.126: Selecting operator for a condition

SET WHERE

Note : Surround string constants with 's (single quotes)

Column	Operator	Column Value	And/Or
IDNO	BETWEEN	? AND ?	OR
	=	?	OR

Figure 4.127: WHERE tab with conditions and operators selected

11. Constant values can also be set to columns that have to be updated (under **SET** tab) or for values in where condition (under **WHERE** tab).
12. To update a column with a constant value, specify the required value in the **Column Value** column against the required column name in **SET** tab.

    Note:

    - If the value is a string value it should be wrapped in single quotes (**' '**)
    - **?** indicates value is taken from input or from the output of another query where possible

SET WHERE

Note : Surround string constants with 's (single quotes)

Column	Column Value
NAME	?
AGE	20

Figure 4.128: Specifying constant value for a column in SET clause

13. To specify a constant value for **where** condition on a column, specify the required value in the **Column Value** column against the required column name in **WHERE** tab.

Note:

- If the value is a string value it should be wrapped in single quotes ('').
- **?** indicates value is taken from input or from the output of another query where possible.

SET | WHERE

Note : Surround string constants with 's (single quotes)

Column	Operator	Column Value	And/Or
IDNO	=	?	AND
NAME	=	'testname'	OR
	=	?	OR

Figure 4.129: Specifying constant value for a column in condition for WHERE clause

14. To specify **where** condition on a column whose value is equal to value defined in another column, select the required column from the drop-down list in the **Column Value** column against the required column name in **WHERE** tab.

SET | WHERE

Note : Surround string constants with 's (single quotes)

Column	Operator	Column Value	And/Or
IDNO	=	?	OR
	=	IDNO	OR
		NAME	
		AGE	

Figure 4.130: Specifying comparison between columns in condition for WHERE clause

15. Update statement is automatically generated and shown in the text editor under SQL Statement. The generated SQL can be validated by pressing (**check syntax**) button.

    Note: This feature only checks for invalid tokens, it does not perform a complete syntax check

SQL Statement

```
1 UPDATE "SCOTT"."EMP_ORIGINAL"
2 SET  "NAME" = ?, "AGE" = 20
3 WHERE "IDNO" = ? AND "NAME" = 'testname'
```

Figure 4.131: Generated update query

16. Click **Ok** to close the dialog.

Update Statement with Failover Insert (aka upsert)

Behavior: Update a rows satisfying defined condition in configured table with column values taken from input XML. Condition values can also be taken from input XML. If the update fails to update any rows (update count = 0), then insert a row with provided values.

Note:

- Values are inserted only for columns which are either selected for **set** or **where** clause
- UPSERT fails if a column which has NOT NULL condition is not a part of either **set** or **where** clause.
- UPSERT when placed under SELECT will not work as we do no support multi-level processing.

1. Configuring update statement following steps mentioned in Simple Update Statement.
2. Check **UPSERT** check box.

Figure 4.132: UPSERT Check Box

4.3.1.2.1.5 Delete Statement Configuration

Click on **Add...** → **Delete Statement** to launch **DELETE Query Builder** (shown in Figure 4.133)

Note:

- Do not type in the text area under before configuring the query.
- Do any modifications after all other configuration is done.
- For user modified values required input / output details (e.g. data types) will not be populated and have to configured manually

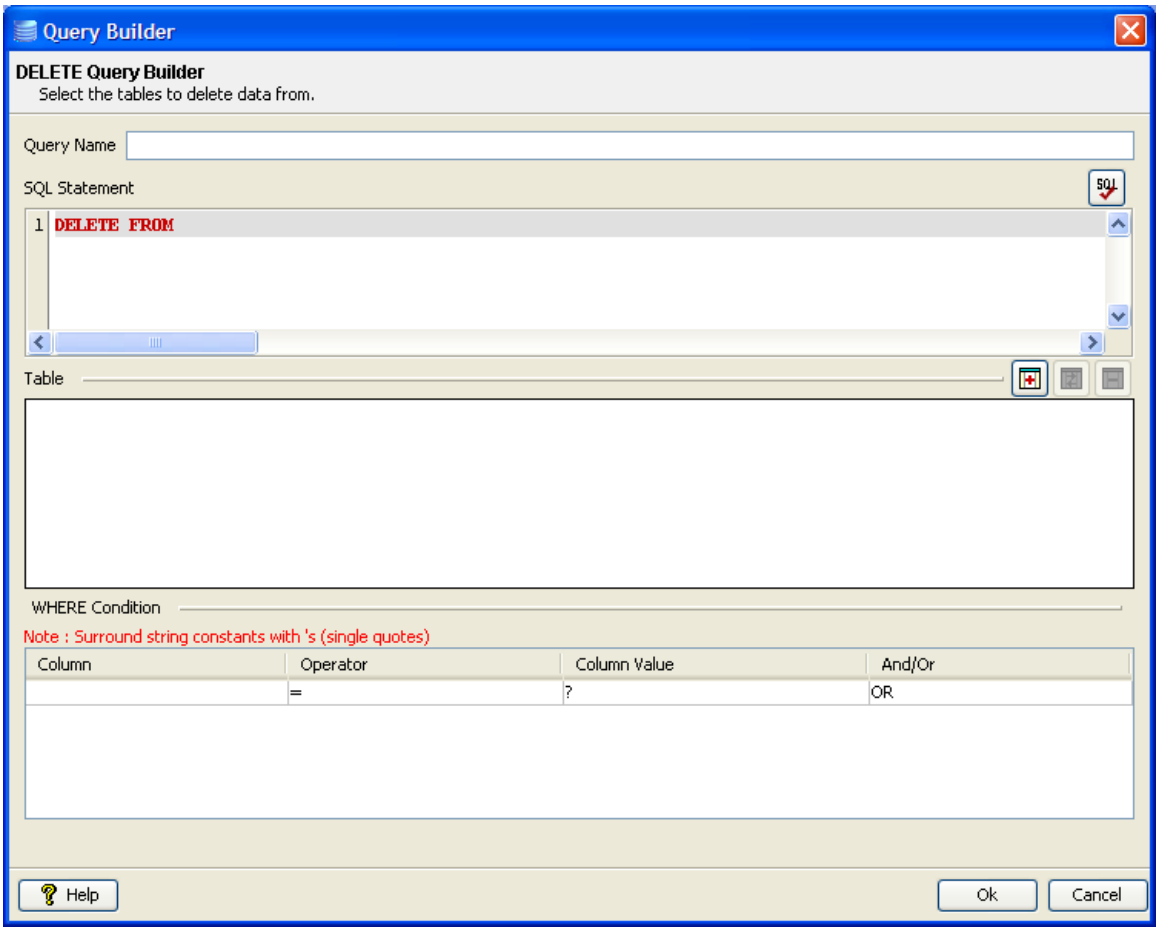


Figure 4.133: Delete Query Builder

Simple Delete Statement

Behavior: Delete rows satisfying defined condition in configured table, with column values taken from input XML or with constant values

1. Provide a name for the query against Query Name.
2. Click on (**add database table**) button to launch **Table Selection Dialog**.
3. Select required table as explained in **Object Selection** section.
4. Selected table is added to the easel under **Table**.

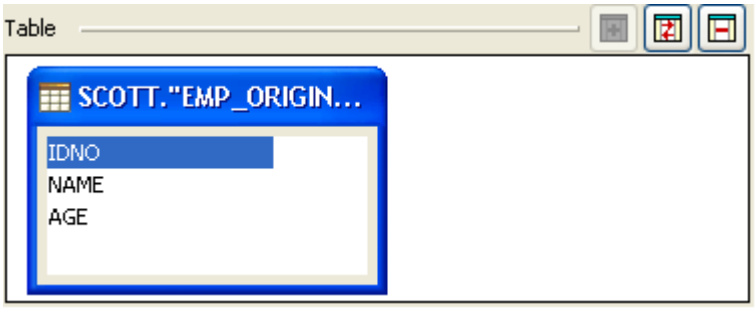


Figure 4.134: Selected table added to easel with all columns

5. Table can be changed by clicking (**replace selected table**) button and removed by clicking (**remove database table**) button.
6. Specify condition which should be satisfied for deleting row under **WHERE condition**. Select a column name on which where condition has to be applied.

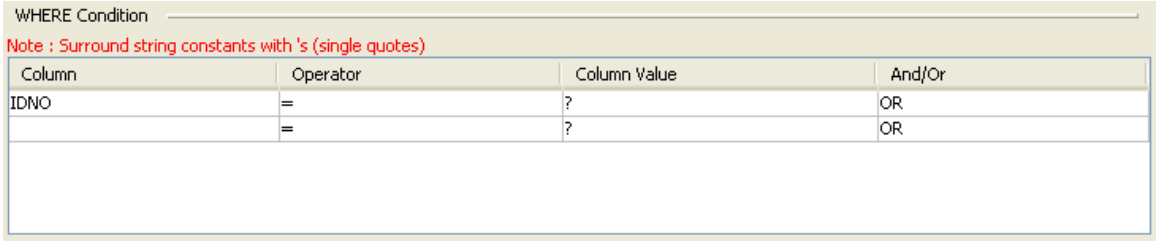


Figure 4.135: Adding condition on column to WHERE clause

7. When selecting multiple columns for where condition, conditions can be combined using **AND** or **OR** under **And/Or** column.

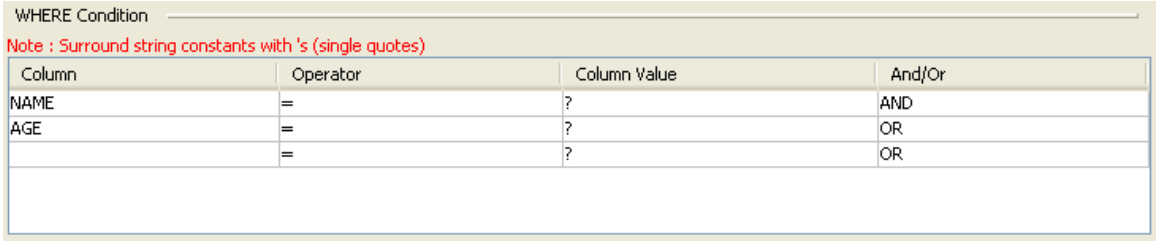


Figure 4.136: Specifying multiple conditions for WHERE clause

8. Operator of choice can be chosen from the drop down under **Operator** column.

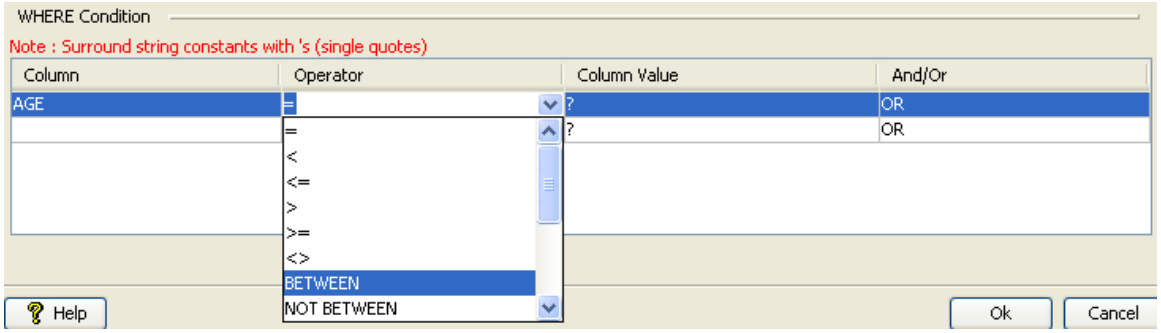


Figure 4.137: Selecting operator for a condition

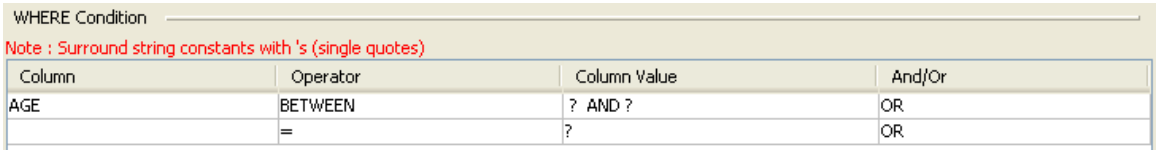


Figure 4.138: WHERE Tab with conditions and operators selected

9. To specify a constant value for where condition on a column, specify the required value in the **Column Value** column against the required column name in **where** tab.

   Note:

   - If the value is a string value it should be wrapped in single quotes (' ').
   - **?** indicates value is taken from input or from the output of another query where possible.

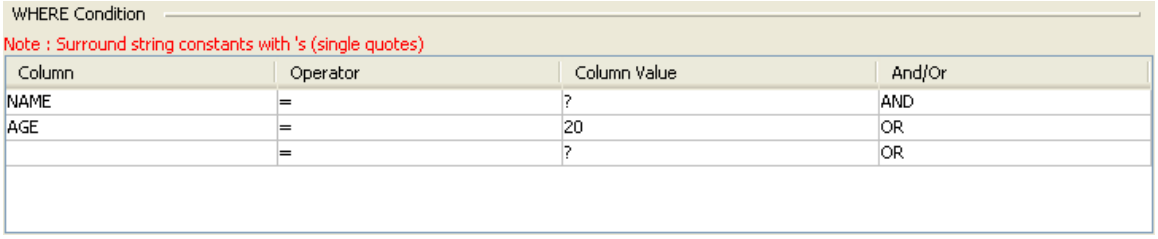


Figure 4.139: Specifying constant value for a column in condition for WHERE clause

10. To specify where condition on a column whose value is equal to value defined in another column, select the required column from drop down in the **Column Value** column against the required column name in **where** tab.

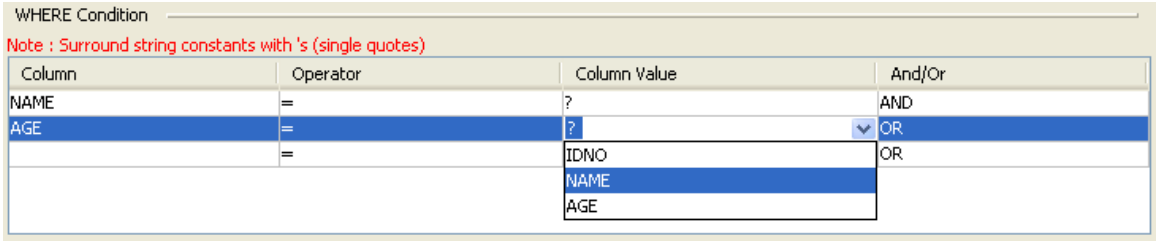


Figure 4.140: Specifying comparison between columns in condition for WHERE clause

11. Delete statement is automatically generated and shown in the text editor under SQL Statement. The generated SQL can be validated by pressing (**check syntax**) button.

    Note: This feature only checks for invalid tokens, it does not perform a complete syntax check.

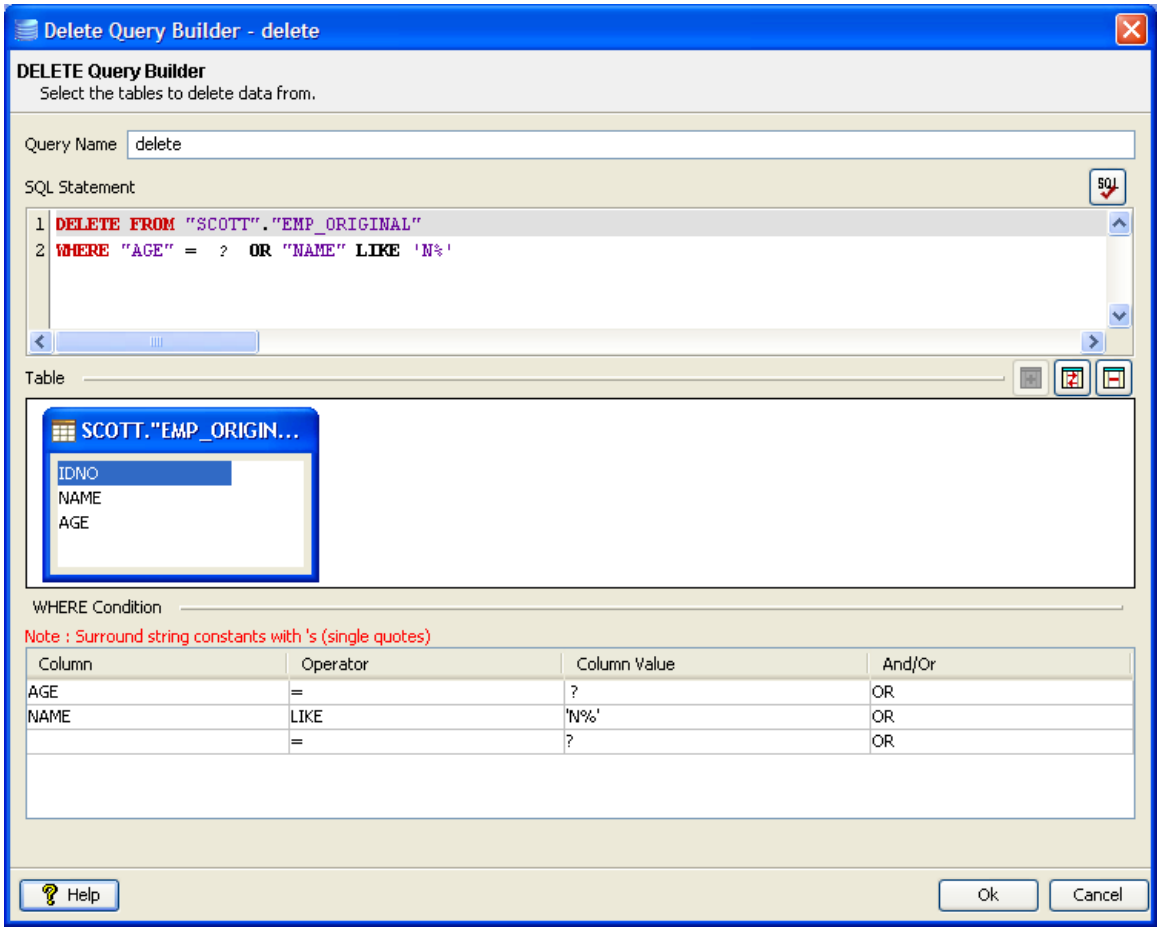


Figure 4.141: Generated delete query

12. Click **Ok** to close the dialog.

4.3.1.2.1.6 Select Statement Configuration

Click on **Add...** → select **Delete Statement** to launch **SELECT Query Builder** (shown in Figure 4.142).

Note:

- Do not type in the text area under before configuring the query.
- Do any modifications after all other configuration is done.
- For user modified values required input/output details (Example: data types) will not be populated and have to configured manually.

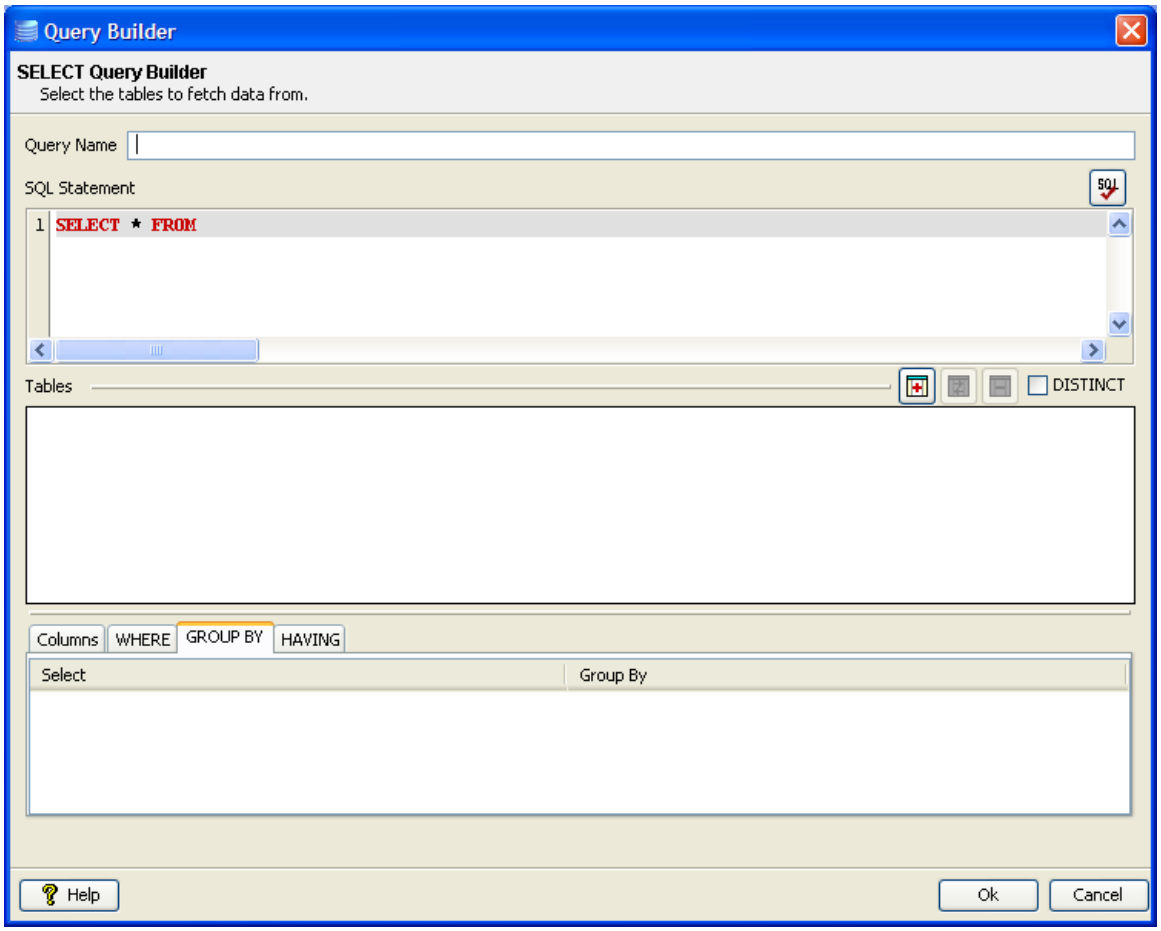


Figure 4.142: Select Query Builder

Simple Select Statement

Behavior: Retrieves data from all columns or from selected columns in a configured database table.

1. Provide a name for the query against Query Name.
2. Click on (**add database table**) button to launch **Table Selection Dialog**.
3. Select required table as explained in **Object Selection** section.
4. Selected table is added to the easel under **Table**. Primary key column, if exists, is marked with adjacent to column name.

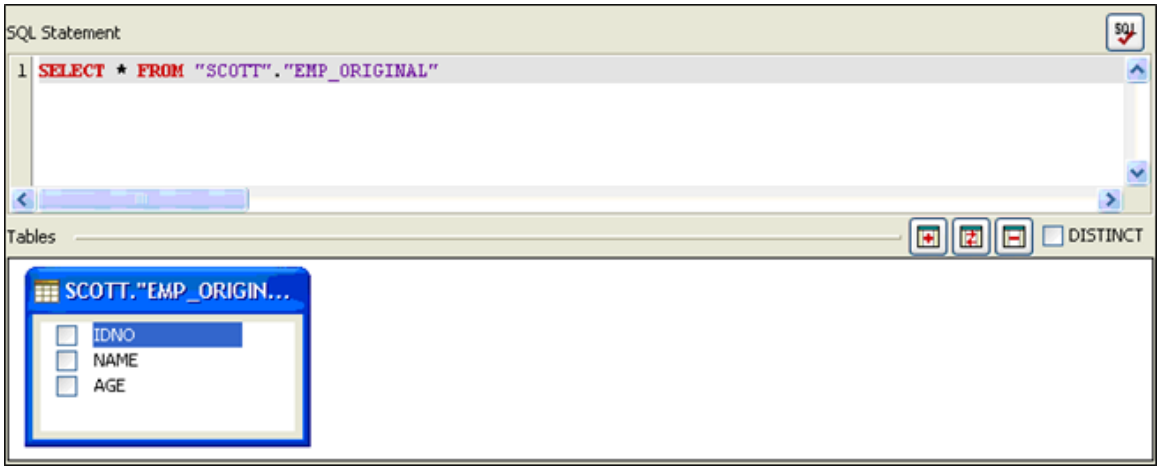


Figure 4.143: Selected table added to easel with all columns

5. Table can be changed by clicking (**replace selected table**) button and removed by clicking (**remove database table**) button.
6. To retrieve specific columns values from the table, check required columns to build a select query with specific columns. If no column is checked, then **SELECT *** is used. Select the columns in order in which they should appear they should appear in select clause.

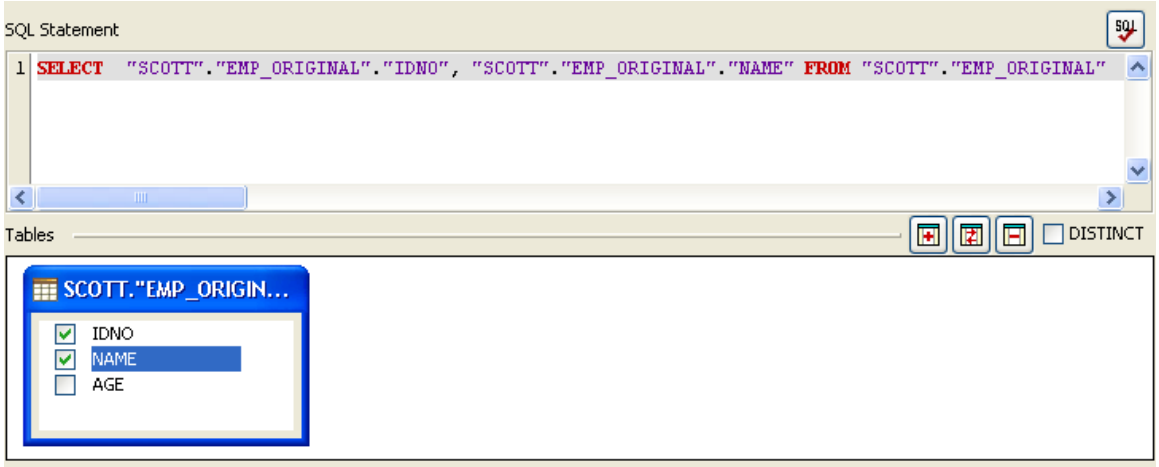


Figure 4.144: Ignoring column for selection

7. Selected columns are shown under **Columns** tab. Check/Uncheck the check box in **Output** column against required column name to show/not show the corresponding column in the output XML.

   For example, configuration in the following image generates IDNO in the output XML but does not generate NAME in output XML, though values for both IDNO and NAME are retrieved from the table.

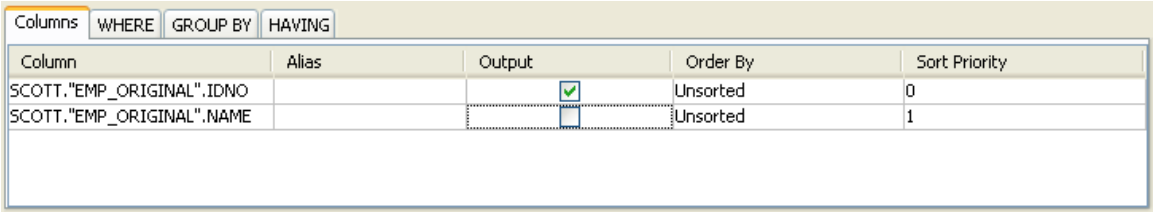


Figure 4.145: Selecting columns for output XML

8. To define a column alias, provide the alias name under Alias column against required column name. Aliases are useful when the column name is not intuitive or too long. When an alias is specify output XML contains an element with defined alias name instead of the column name.

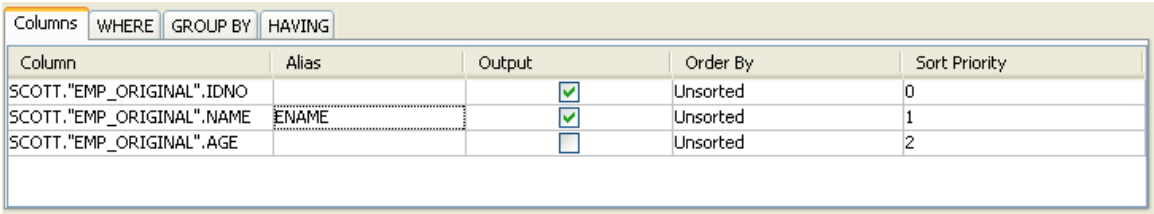


Figure 4.146: Defining Column Alias

9. To return unique rows check **DISTINCT**.

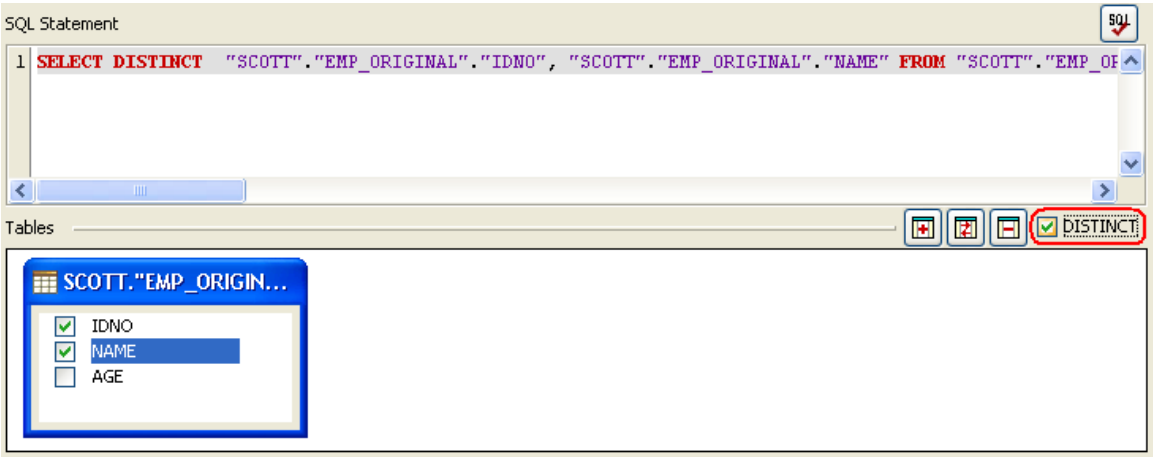


Figure 4.147: Distinct option to return unique values

10. Select statement is automatically generated and shown in the text editor under SQL Statement. The generated SQL can be validated by pressing (**check syntax**) button.

    Note: This feature only checks for invalid tokens, it does not perform a complete syntax check.

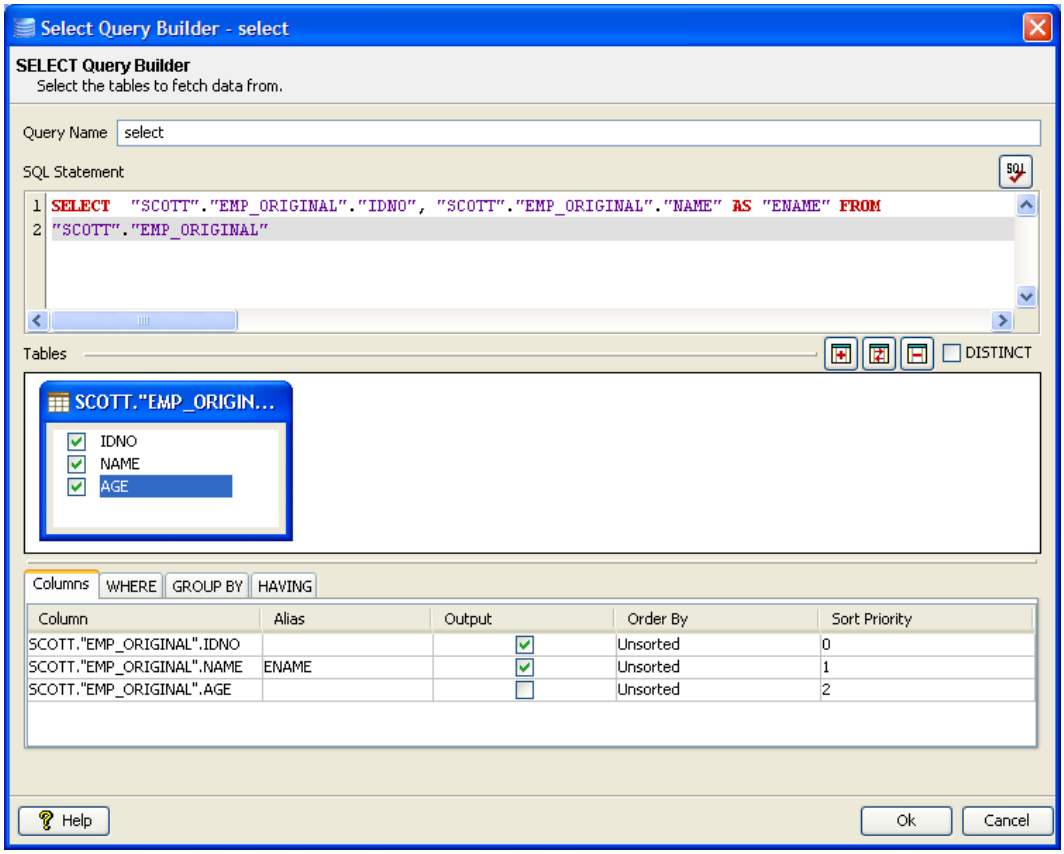


Figure 4.148: Generated select query

11. Click **Ok** to close the dialog.

Select Statement with Filter

Behavior: Retrieves data from all columns or from selected columns in a configured database table after applying specified conditions. Condition values can be provided from input XML or as constant values.

1. Follow the steps from 1 to 8 as described in the section Simple Select Statement.
2. Click on **WHERE** tab and select a **Column** name on which WHERE condition has to be applied.

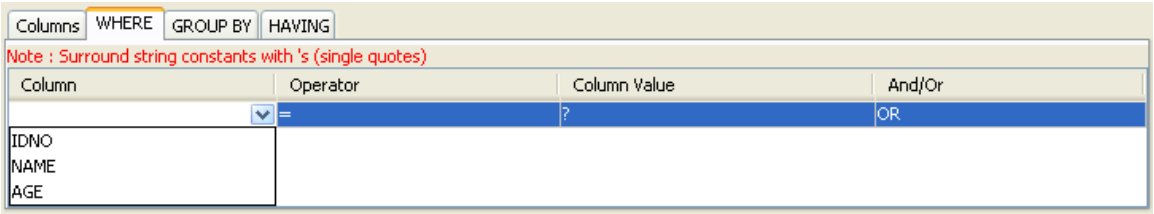


Figure 4.149: Adding condition on column to WHERE clause

3. When selecting multiple columns for WHERE condition, conditions can be combined using **AND** or **OR** under **And/Or** column.

Columns | WHERE | GROUP BY | HAVING

Note : Surround string constants with 's (single quotes)

Column	Operator	Column Value	And/Or
IDNO	=	?	AND
NAME	=	?	OR
	=	?	OR

Figure 4.150: Specifying multiple conditions for WHERE clause

4. Operator of choice can be selected from the drop-down list under **Operator** column.

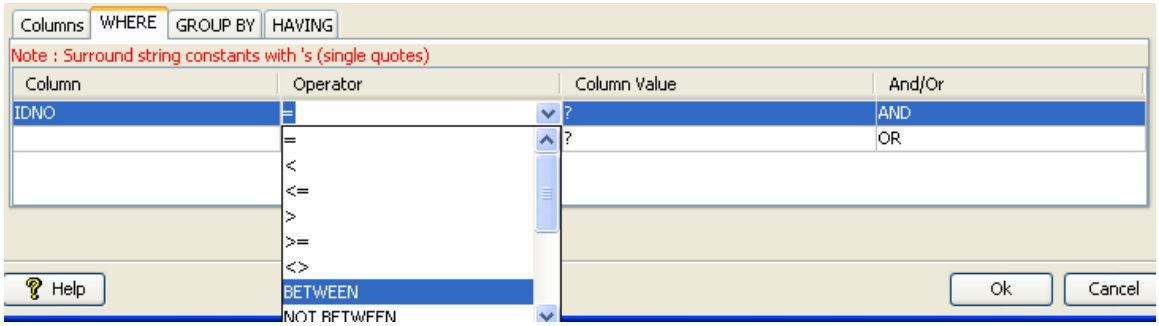


Figure 4.151: Selecting operator for a condition

Columns | WHERE | GROUP BY | HAVING

Note : Surround string constants with 's (single quotes)

Column	Operator	Column Value	And/Or
IDNO	BETWEEN	? AND ?	AND
	=	?	OR

Figure 4.152: WHERE tab with conditions and operators selected

5. Constant values can also be set for values in WHERE condition (under **WHERE** tab).
6. To specify a constant value for WHERE condition on a column, specify the required value in the **Column Value** column against the required column name in **WHERE** tab.

   Notes:

   - If the value is a string value it should be wrapped in single quotes (**' '**).
   - **?** indicates value is taken from input or from the output of another query where possible.

Columns | WHERE | GROUP BY | HAVING

Note : Surround string constants with 's (single quotes)

Column	Operator	Column Value	And/Or
IDNO	=	?	AND
NAME	=	'testname'	OR
	=	?	OR

Figure 4.153: Specifying constant value for a column in SET clause

7. To specify WHERE condition on a **Column** whose value is equal to value defined in another **Column**, select the required **Column** from drop-down list in the **Column Value** against the required column name in **WHERE** tab.

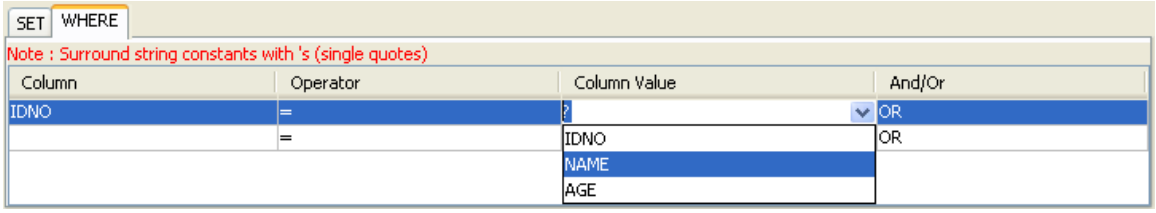


Figure 4.154: Specifying comparison between columns in condition for WHERE

8. Select statement is automatically generated and shown in the text editor under SQL Statement. The generated SQL can be validated by clicking the (**check syntax**) button.

   Note: This feature only checks for invalid tokens, it does not perform a complete syntax check.

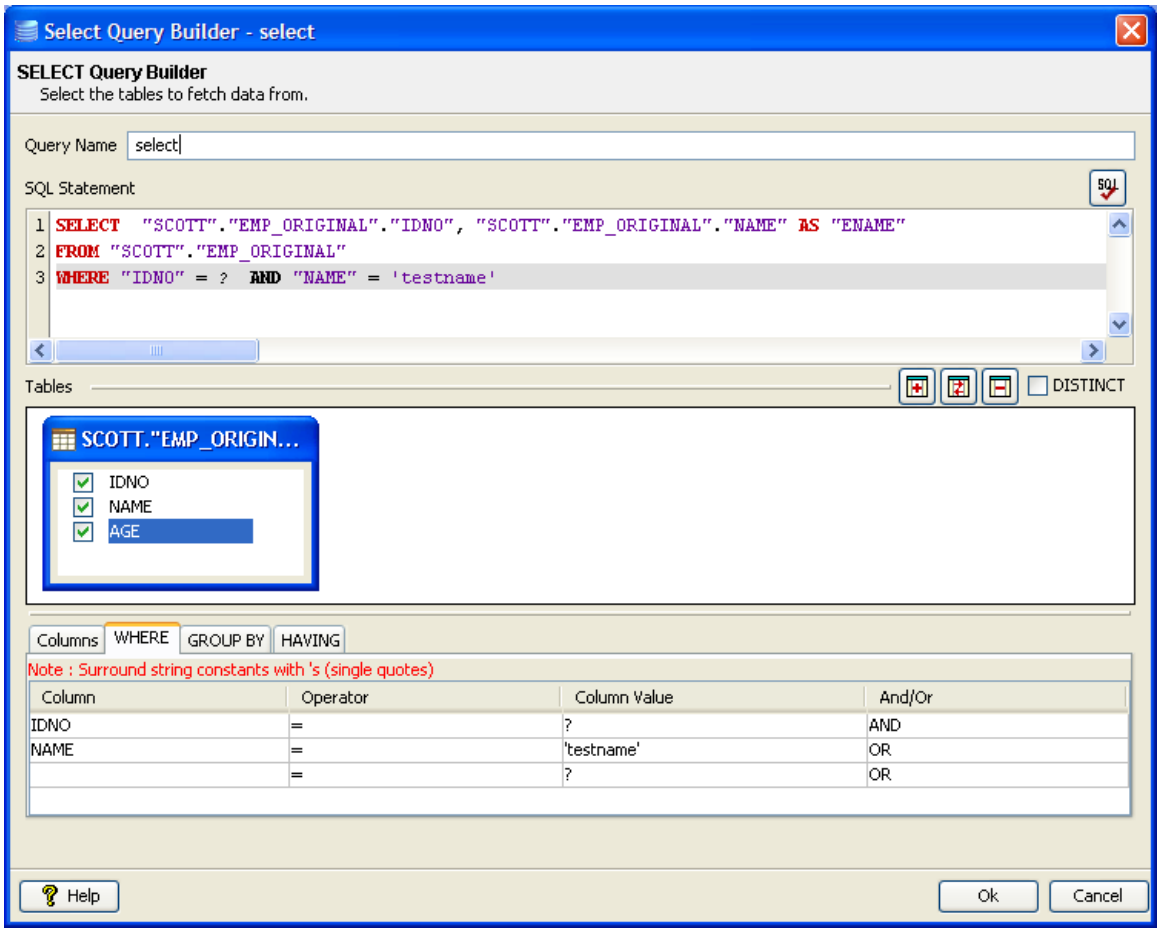


Figure 4.155: Generated select query with filter

9. Click the **Ok** button to close the dialog.

Select Statement with Sorting

Behavior: Retrieves sorted data from all columns or from selected columns in a configured database table. Data is sorted in configured order on columns configured for sorting.

1. Follow steps 1 to 8 in the section **Simple Select Statement.**
2. To specify columns which have to be sorted, select the appropriate sort order from drop-down list under **Order By** column. **Order By** for each columns has one of the following values:

Order By Value	**Explanation**
Unsorted	Data is not sorted on values in the column, that is, no order by clause is added in the SQL statement.
Ascending	Data is sorted in ascending order on values in the column, that is order by clause is added in the SQL statement as ORDER BY <column name> ASC.
Descending	Data is sorted in descending order on values in the column, i.e. order by clause is added in the SQL statement as ORDER BY <column name> DESC.
Default	Data is sorted in default order for order by clause on values in the column, that is, order by clause is added in the SQL statement as ORDER BY <column name>.

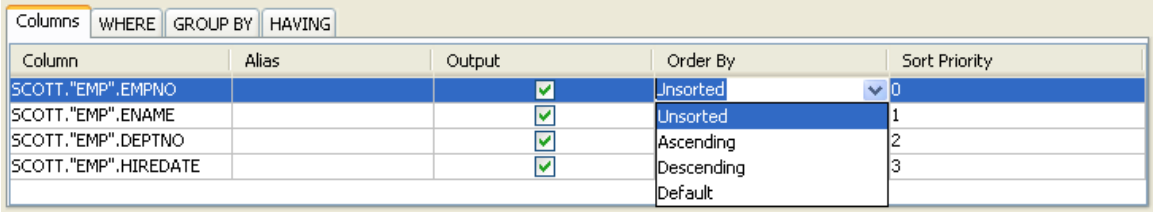


Figure 4.156: Selecting sorting order for column

An example of SQL statement with different sort orders is shown in the Figure 4.157.

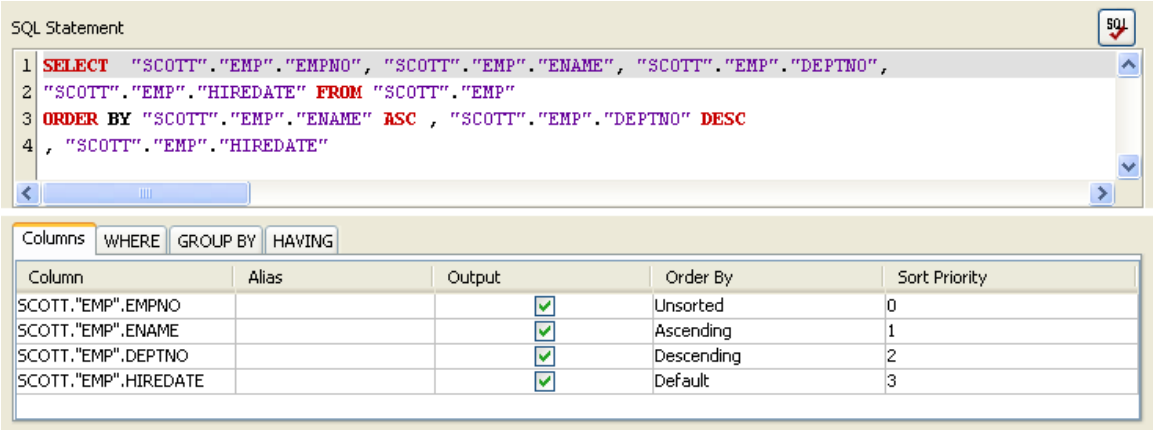


Figure 4.157:SQL Statement with different columns sorted in different order

3. When multiple columns have to be sorted, sorting priority for each column can be set under Sort Priority. Columns are sorted in order of increasing Sort Priority that is column with minimum value for Sort Priority is order first.

When values of Sort Priority for multiple columns are same, columns are sorted in the order in which they appear in select clause.

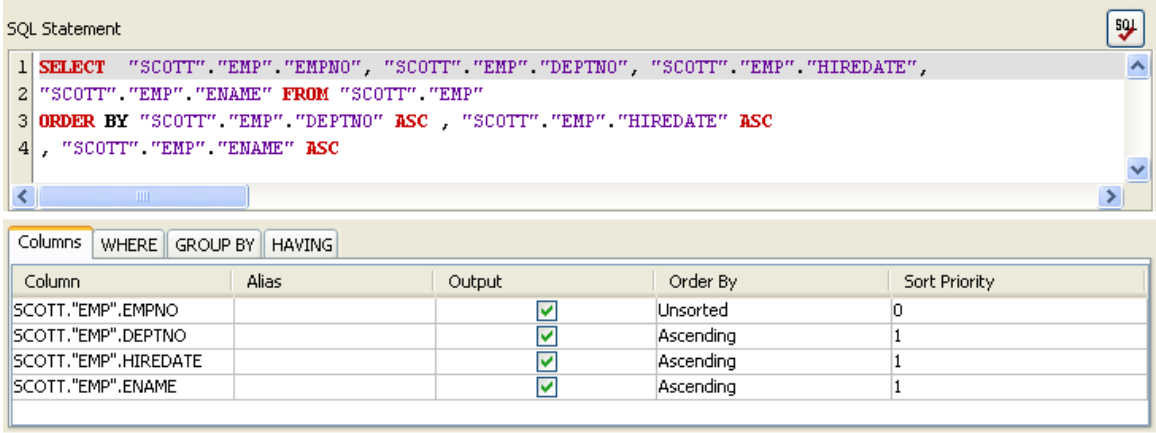


Figure 4.158: SQL Statement with Sort Priority

4. Select statement is automatically generated and shown in the text editor under SQL Statement. The generated SQL can be validated by clicking (check syntax) button.

   Note: This feature only checks for invalid tokens, it does not perform a complete syntax check.

5. Click the **Ok** button to close the dialog.

Select Statement with Grouping

Behavior: Retrieves data, after applying grouping conditions, from all columns or from selected columns in a configured database table.

Note: Grouping functions are not provided in query builder. Grouping conditions have to be explicitly added by editing the SQL statement either before closing the query builder or by launching.

1. Follow the steps from 1 to 8 as described in the section Simple Select Statement.
2. Click on **GROUP BY** tab and check under **Select** against the columns under **Group By** on which group by condition should be applied.

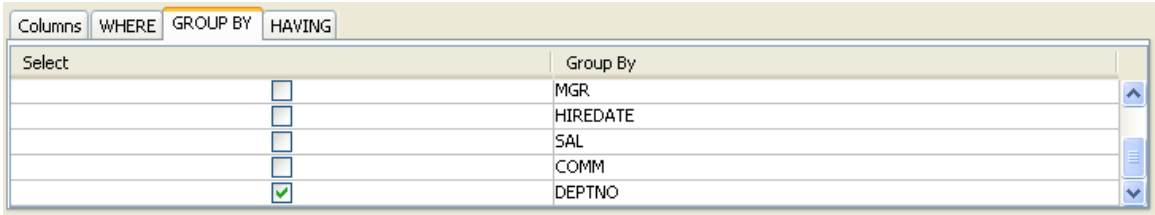


Figure 4.159: Selecting columns for grouping condition

3. To filter the results click on **HAVING** tab and define required conditions. **HAVING** tab has functionality similar to **WHERE** tab (described in Select Statement with filter).

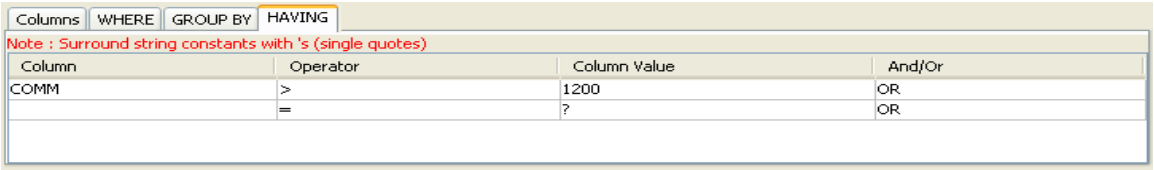


Figure 4.160: Adding condition to HAVING clause

4. Select required columns under Tables.

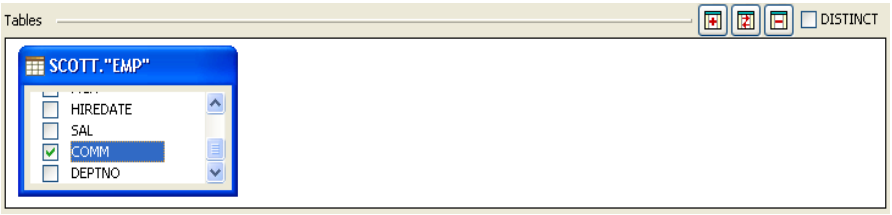


Figure 4.161: Selecting required columns

5. Edit Select and HAVING clauses to apply appropriate grouping condition on selected columns.

   Note: Editing Select and HAVING clauses should be last action before closing the dialog.

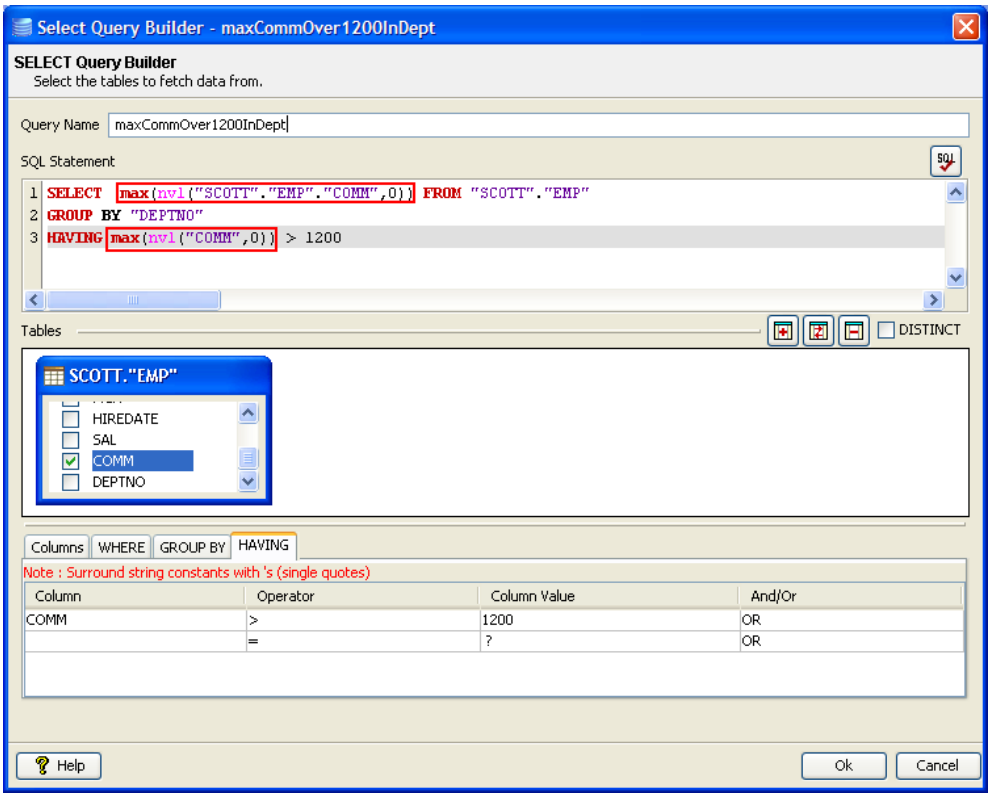


Figure 4.162: Generated select query with grouping

6. Click the **Ok** button to close the dialog.

Select Statement with Multiple Tables

Behavior: Retrieves data from all columns or from selected columns from multiple configured database tables.

1. Follow the steps from 1 to 5 as described in the section Simple Select Statement**.**
2. To add second table, click on (add database table) button to launch **Table Selection** dialog**.**

   Notes:

   - Multiple tables can be added by repeating this step.
   - Specify any conditions after selecting all required tables.

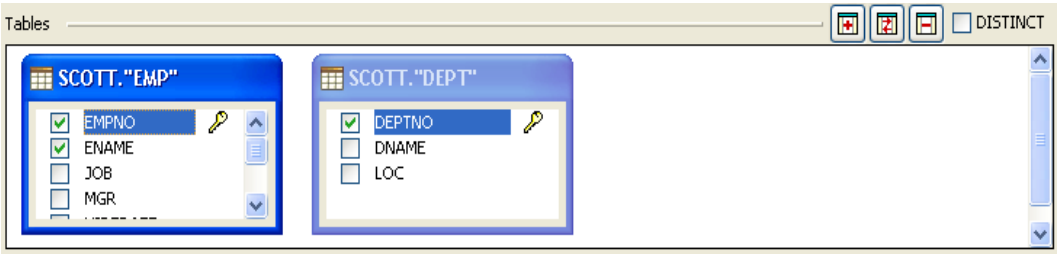


Figure 4.163: Selecting multiple tables

3. Add WHERE condition, described in section **Select Statement with filter**, to perform join on the tables. If no condition is specified, Cartesian product of rows in all selected tables is returned.
4. To specify the join, in **WHERE** tab, select the required column from one table under **Column** and select the required column from another table under **Column Value.**

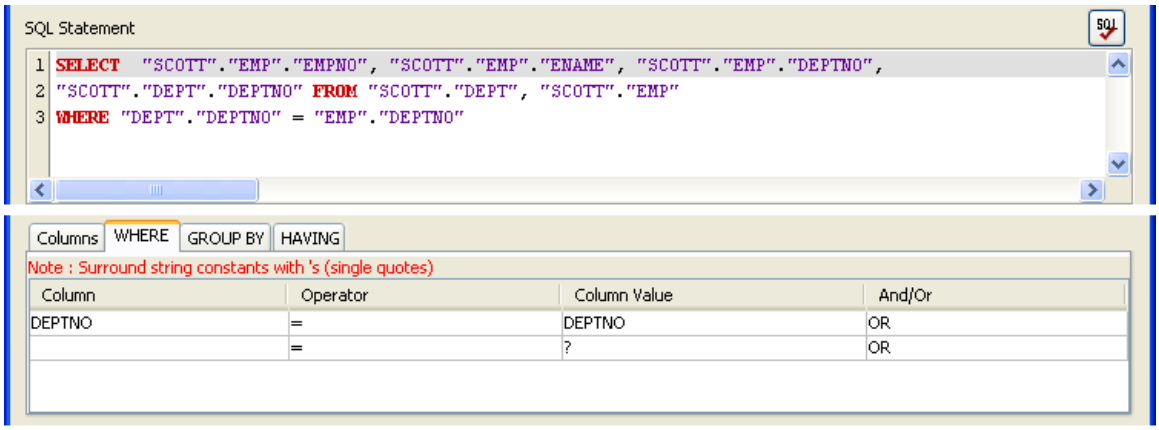


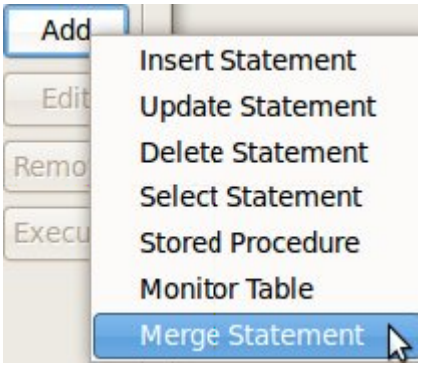
Figure 4.164: Generated SQL statement with join

5. To specify filtering, sorting or grouping conditions refer to sections above.
6. Click the **Ok** button to close the dialog.

4.3.1.2.1.7 Merge Statement Configuration

Note: Merge statement can only be configured in eStudio CPS.

Click on **Add** -> **Merge Statement** to launch Merge Query Builder.



Behavior: Updates or inserts rows in target table with rows selected from one or more source tables.

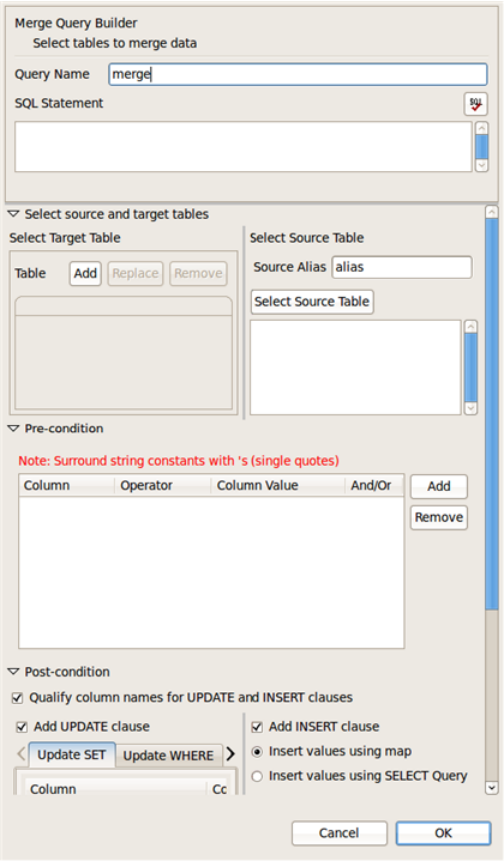


Figure: Merge Query Builder

1. Provide a name for the merge query against Query Name
2. Select Target Table
   - Select the target table using the Object Selection dialog by clicking on Add button under Select Target Table. This table can be replaced or removed by using Replace and Remove buttons respectively.
   - Columns based on which ON condition clause is created should be unchecked as shown in the figure below. These unchecked columns will not be shown in Update SET table (as shown in Figure: Configure Post-Condition)

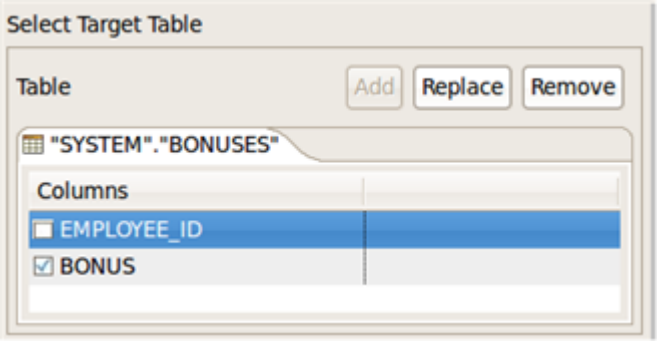


Figure: Select Target Table

3. Select Source Table
   - Source Alias – Specify a correlation name which is an alias for the SELECT subquery to be referenced elsewhere in the MERGE statement.
   - Select the source subquery using the Select Query Wizard under Select Source Table. Configure the SELECT subquery as detailed in section [Select Statement Configuration]

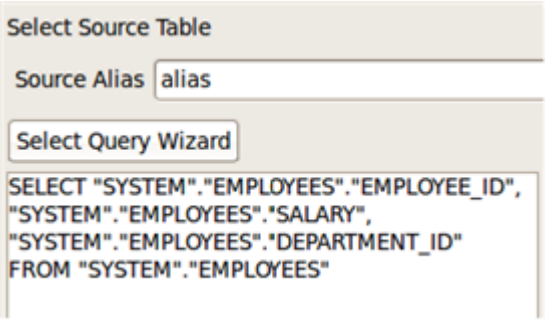


Figure: Select Source Table

4. Configure Pre-Condition
   - Use the Pre-Condition to specify the condition upon which the MERGE operation either updates or inserts. For each row in the target table for which the search condition is true, the component updates the row with the corresponding data from the source table. If the condition is not true for any rows, the component inserts into the target table based on the corresponding row from the source table.
   - The drop-down list under the Column column consists of columns from target table and that of under the Column Value column consists of columns from both target and source tables.

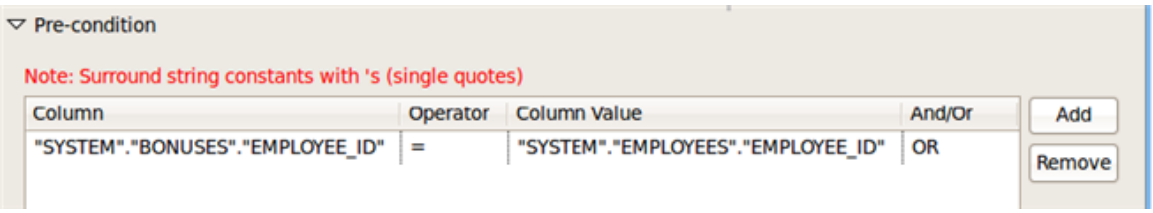


Figure: Configure Pre-Condition

5. Configure Post-Condition
   - Based on whether Pre-Condition is satisfied or not for a given row in the target table, the query either updated or inserts data in the target table.

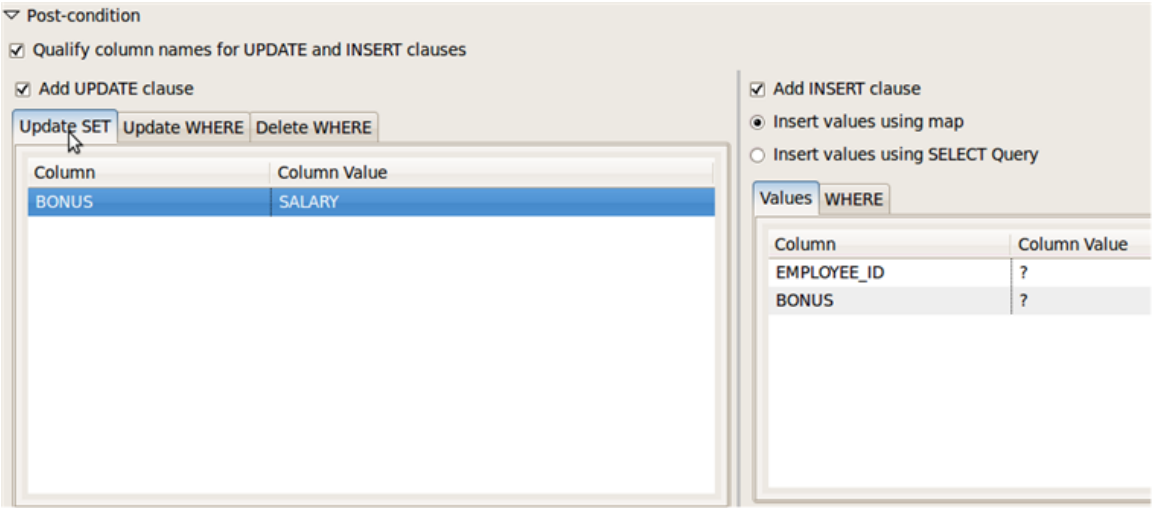


Figure: Configure Post-Condition

- Qualify column names for UPDATE and INSERT clauses – Select this option to qualify, with the schema and table names, names of the columns in the target table.
- Add UPDATE clause – Select this option to add UPDATE clause to the MERGE statement.
- Add INSERT clause – Select this option to add INSERT clause to the MERGE statement.
- Atleast one of UPDATE and INSERT clause should be added.

6. UPDATE
   - The UPDATE clause specifies the new column values of the target table.
   - The Configuration of Update SET and Update WHERE are detailed in section Update Statement Configuration
   - Only the columns that are checked under Select Target Table (as shown in Figure 4.166: Select Target Table) will be shown here.
   - In the Update WHERE table, the drop-down list under the Column column consists of columns from target table and that of under the Column Value column consists of columns from both target and source tables.
7. DELETE
   - Specify the Delete WHERE clause to clean up data in a table while populating or updating it.

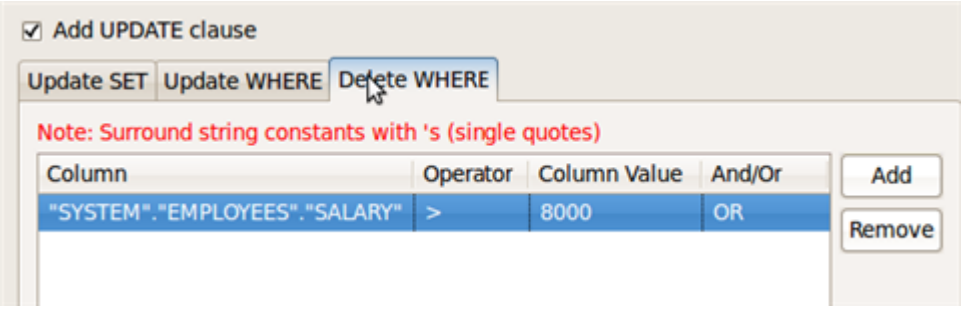


Figure: Configure Delete WHERE Clause

8. INSERT
    - The INSERT clause specifies values to insert into the column of the target table if the condition of the ON clause is false.
    - Configure INSERT query as detailed in Section ...

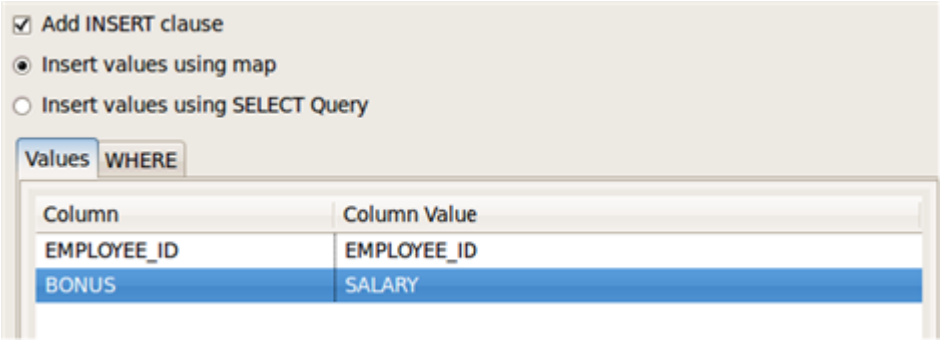


Figure: Configure INSERT clause

Points to Note:

While configuring for MSSQL -

- After configuring the MERGE query using UI, edit the sql statement and add a semi-colon (;) at the end of the query. Do not make any changes in the UI after adding the semi-colon.
- DELETE clause is not supported.
- Un-check the 'Qualify column names for UPDATE and INSERT clauses' option.

4.3.1.2.2 Stored Procedure Configuration

Behavior: Executes a stored procedure and returns the result (returns return value or out parameter values).

Note:

- Functions can also be configured
- Stored Procedure/Function has to be executed at configuration time if it returns a result set to create the output structure.

To configure the Stored Procedure, perform the following steps:

1. Click on **Add** → **Stored Procedure** to launch **Stored Procedure** dialog.

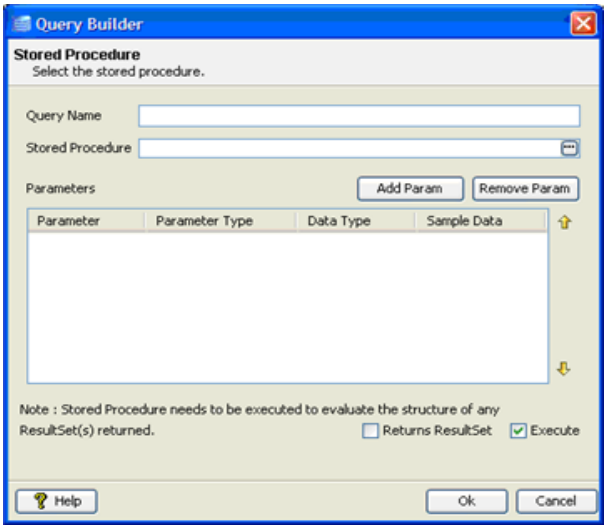


Figure 4.165: Stored Procedure Query Builder

2. Provide a name against Query Name.
3. Click against **Stored Procedure** to launch **Procedure Selection** dialog.
4. Select required procedure as described in section Object Selection.

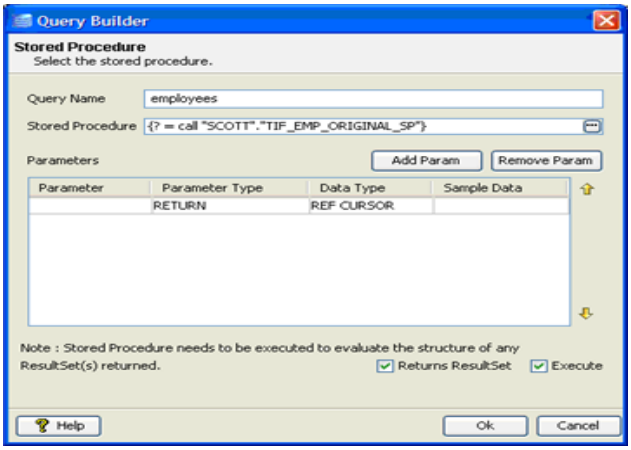


Figure 4.166: Stored procedure details

5. Parameters and their configurations are automatically populated.

Note: Stored Procedures currently does not support User Defined Data types.

Column	Description
Parameter	Name of the parameter for named parameters, blank otherwise.
Parameter Type	Type of parameter – IN, OUT, INOUT, UNKOWN, RETURN, RESULT Values of type OUT, INOUT, RETURN, RESULT form output structure.

Data Type	SQL data type of the parameter.
Sample Data	NA

6. Select **Execute** to execute the stored procedure to create output structure when the Stored Procedure dialog is closed (shown in the Figure 4.167). If Execute is not selected, the output structure will not be defined and has to be manually defined.

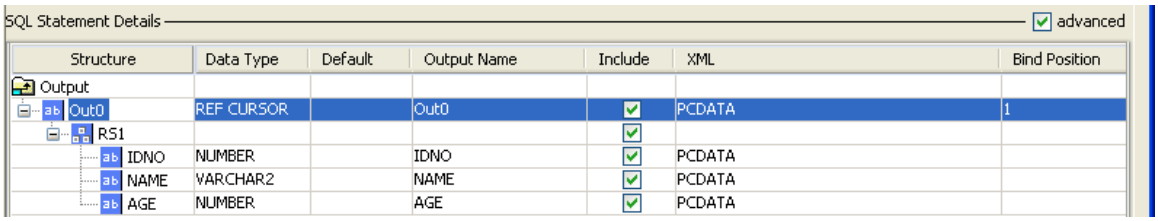


Figure 4.167: Output structure generated for selected stored procedure

7. Click **Ok** to close the dialog.

4.3.1.2.3 Monitor Table Configuration

Behavior: Monitors a configured table for any changes (data addition, data removal and data updates). Monitoring a table requires creation of temporary table/stored procedures and data types and hence is very specific to database in use. This option is **not** supported when **Database** selected is **Other** in MCF panel (see section **Database connection configuration**). This option is supported only for the following databases against **Database** in MCF panel – **IBM DB2, HSQL, Kingbase, Microsoft Access, Microsoft SQL Server, Microsoft SQL Server 2005, Mckoi, MySQL, Oracle, Sybase**.

1. Click on **Add...** → **Monitor Table** to launch **SQL Creation Wizard** (shown in Figure 3.168).

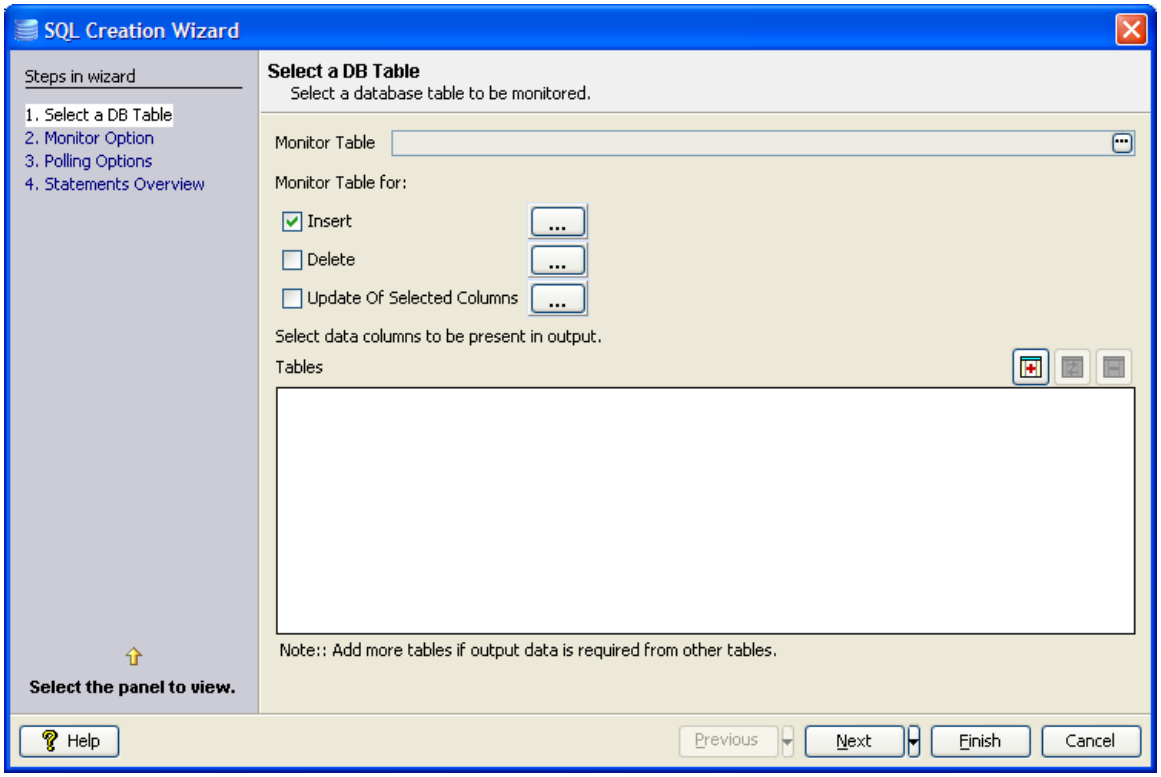


Figure 4.168: Monitor table wizard

Note: Do not use these buttons.

4.3.1.2.3.1 Select DB Table

2. Click ... against **Monitor Table** and choose the table to monitor (refer to section Object Selection).
3. Select actions which have to be monitored.
   - **Insert** – Notifies when a row is added to monitored table.
   - **Delete** – Notifies when a row is deleted from monitored table.
   - **Update Of Selected Columns** – Notifies when a selected column is updated to new value. Column selection panel appears when this option is checked.

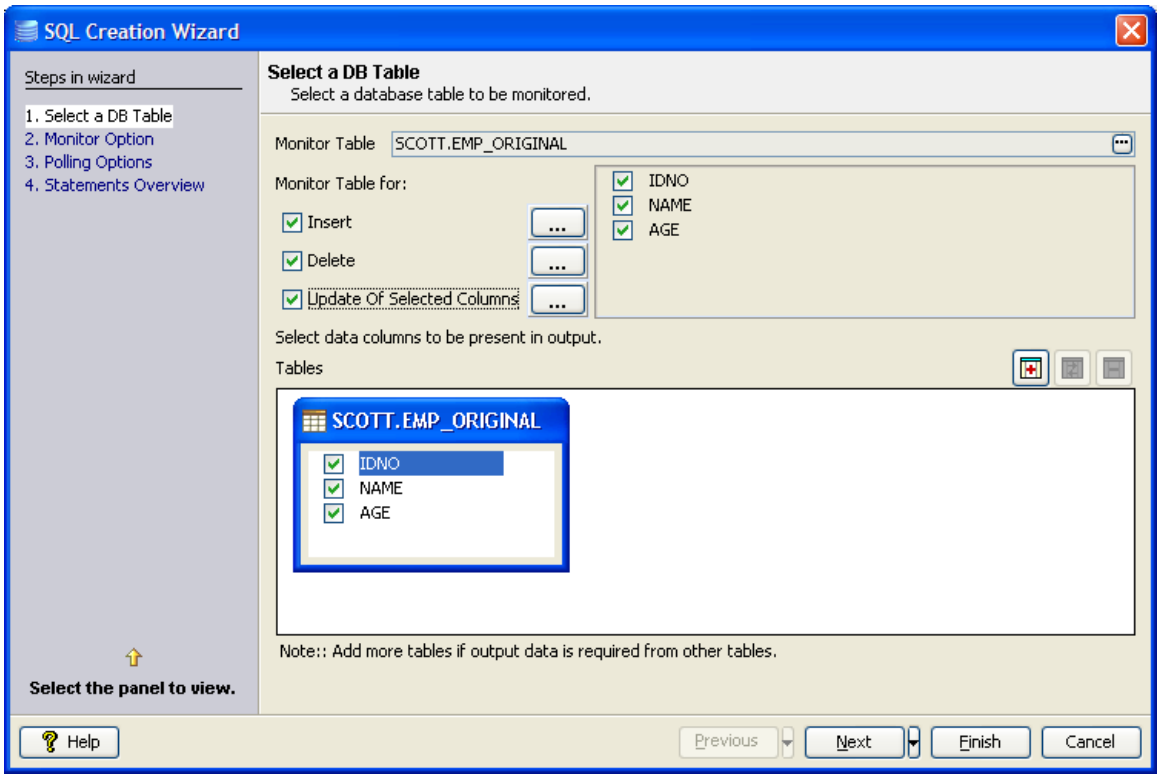


Figure 4.169: Selecting table for monitoring

4. For each action which has to be monitored, specify conditions which filter changes to be notified, click on ... button (configure expression to filter inserted records).
5. Define condition on required columns, similar to WHERE tab in **Select Statement with filter** section. Figure 3.170 shows configuring a condition – send notification if a row is inserted with **IDNO > 500**.

   Note: Do not set column value as? (Like in **WHERE** tab)

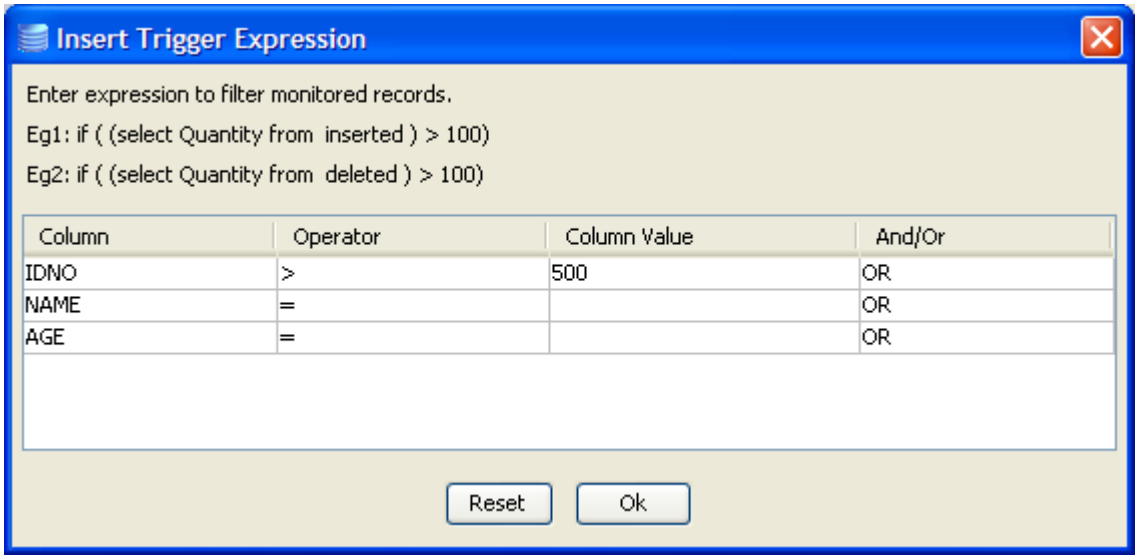


Figure 4.170: Specifying filter condition for monitoring

6. Click the **Next** button.

4.3.1.2.3.2 Monitor Option

7. Select one of the following options to monitor actions on table:

   - **Shadow Table**

     Creates a table containing all columns in the monitored table and a few additional columns (**TIF_RECORDID, TIF_OPERATIONTYPE** and **TIF_STATUS**) required for monitoring. This option is supported only on following databases – **IBM DB2, Kingbase, Microsoft SQL Server, Microsoft SQL Server 2005, Mckoi, Oracle, Sybase**

     Note: Trigger should be allowed by database to use this option.

   - **Alter Main Table**

     Modifies the monitored table to add **TIF_RECORDID, TIF_OPERATIONTYPE** and **TIF_STATUS** columns required for monitoring. This option is supported by all databases that support monitoring.

     Notes:

     - This option should be used with caution as changing table definition might break other applications.
     - **TIF_RECORDID, TIF_OPERATIONTYPE** and **TIF_STATUS** columns should be populated externally.

8. When monitor option is **Shadow Table**, select one the following methods to create a shadow table.

   - **Monitor By REFERENCE**

     Shadow table is created with columns **TIF_RECORDID, TIF_OPERATIONTYPE**, **TIF_STATUS** and primary key of monitored table.

   - **Monitor By VALUE**

Shadow table is created with columns **TIF_RECORDID, TIF_OPERATIONTYPE**, **TIF_STATUS** and all columns of monitored table.

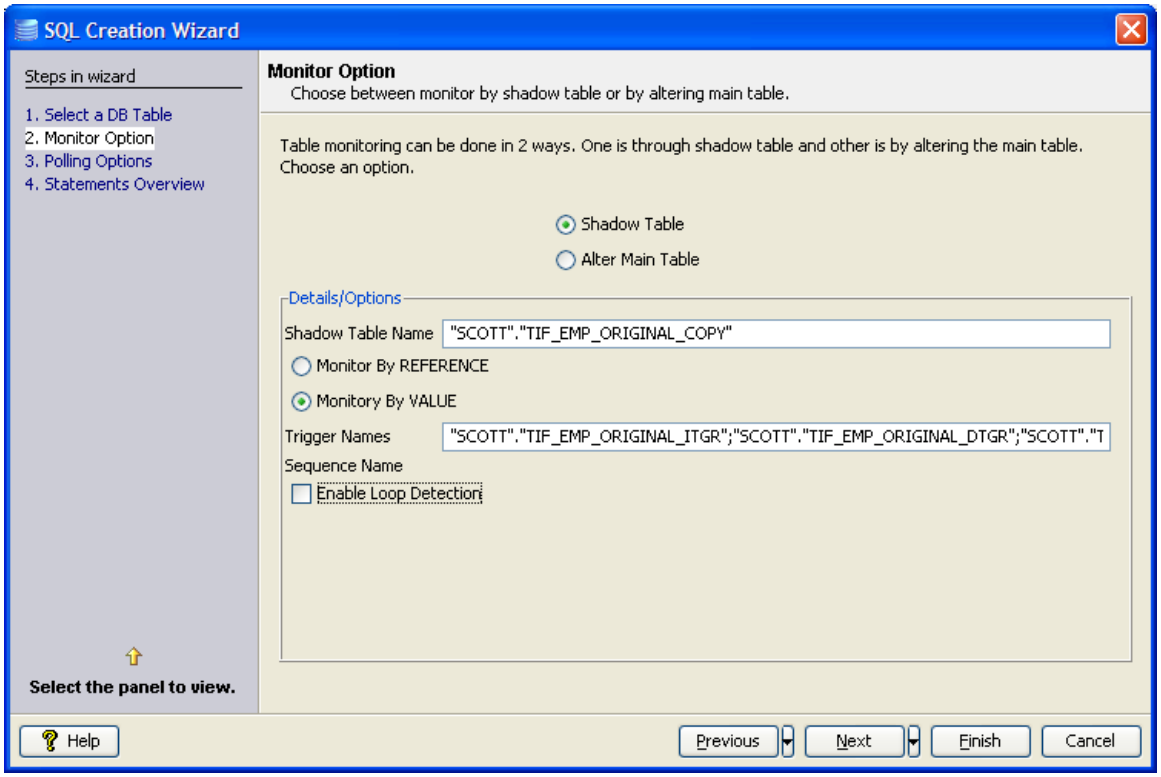


Figure 4.171: Selecting monitor option

9. In case of either monitor option, **Enable Loop Detection** modifies the monitored table to add an additional column **TIF_SOURCE** whose value should be **NULL** for notifications.

   Notes:

   - This option should be used with caution as changing table definition might break other applications.
   - **TIF_**RECORDID**, TIF_OPERATIONTYPE** and **TIF_STATUS** columns should be populated externally.

10. Click the **Next** button.

4.3.1.2.3.3 Polling Options

11. Based on monitor option selected, either shadow table or monitored table should be continuously polled to identify changes done to monitored table and notify. Select one of the following options for polling:

    - **Stored Procedure**

      This option is supported only on following databases – **IBM DB2**, **Kingbase**, **Microsoft SQL Server**, **Microsoft SQL Server 2005**, **Oracle**, and **Sybase.**

      Names for all databases that is created are populated automatically and can be changed.

    - **Select Statement**

This option is supported by all databases that support monitoring. It creates an update and a select statement instead of a single stored procedure.

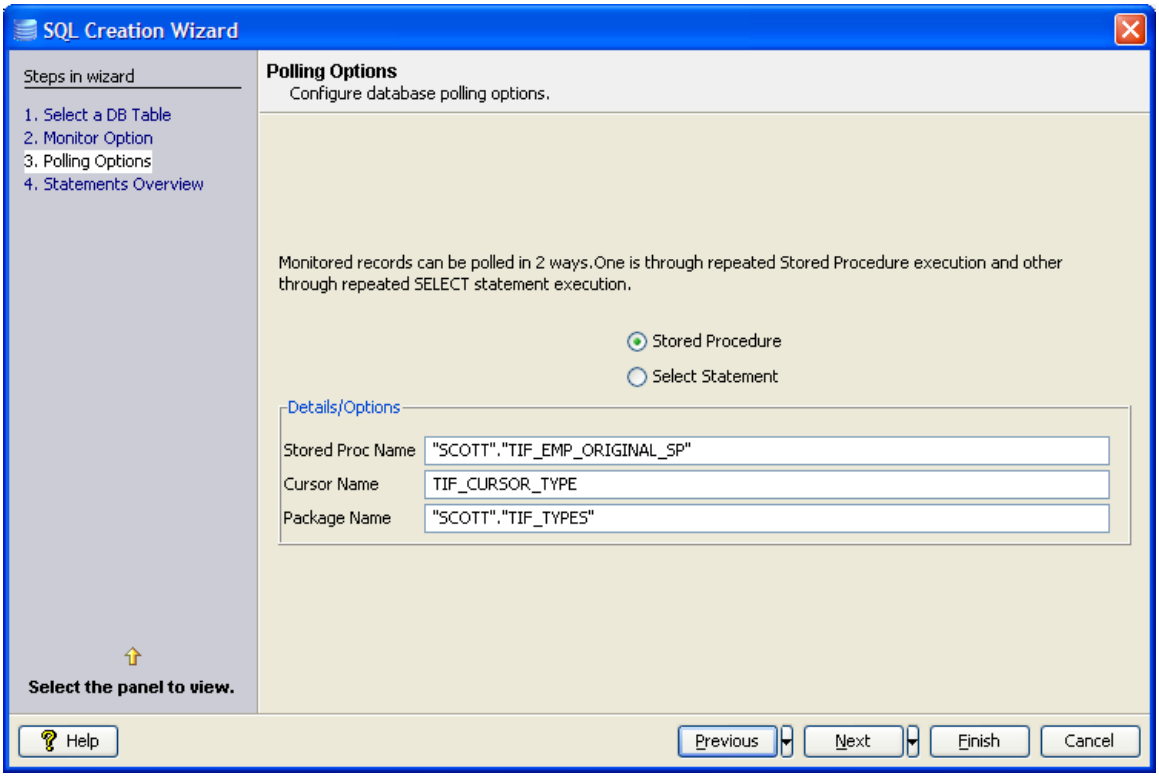


Figure 4.172: Selecting polling option

12. Click the **Next** button.

4.3.1.2.3.4 Statements Overview

13. To check SQLs which create required database objects required for monitoring, click **View SQLs...** These SQLs are by default executed when **Finish** button is clicked.
14. To check SQLs which remove all database objects created for monitoring, click **View Cleanup SQLs...**
15. SQLs and Cleanup SQLs are saved at following locations for future reference.

    SQLs – %ESB_USER_DIR% \ studio \ <build no>\ cache \ components \ DB \ 4.0 \ <monitor table name>_config.sql

    Cleanup SQLs – %ESB_USER_DIR% \ studio \ <build no>\ cache \ components \ DB \4.0 \ <monitor table name>_cleanup.sql
16. Check **Ignore SQL execution errors** to finish the wizard even if some exceptions occur when executing SQLs to create database objects required for monitoring.

    Note: If this is checked, appropriate database objects should be created by user.
17. Check **Do not execute SQLs on Finish** to finish the wizard without creating database objects required for monitoring.

    Note: If this is checked, appropriate database objects should be created by user.
18. Click the **Finish** button.

4.3.1.2.4 SQL Statement Details Configuration

SQL Statement Details shows detailed configuration of the selected query:

- SQL statement in **Query** tab.
- Configuration of input parameters which have to be passed to execute the query.
- Configuration of output parameters which are returned after query execution.

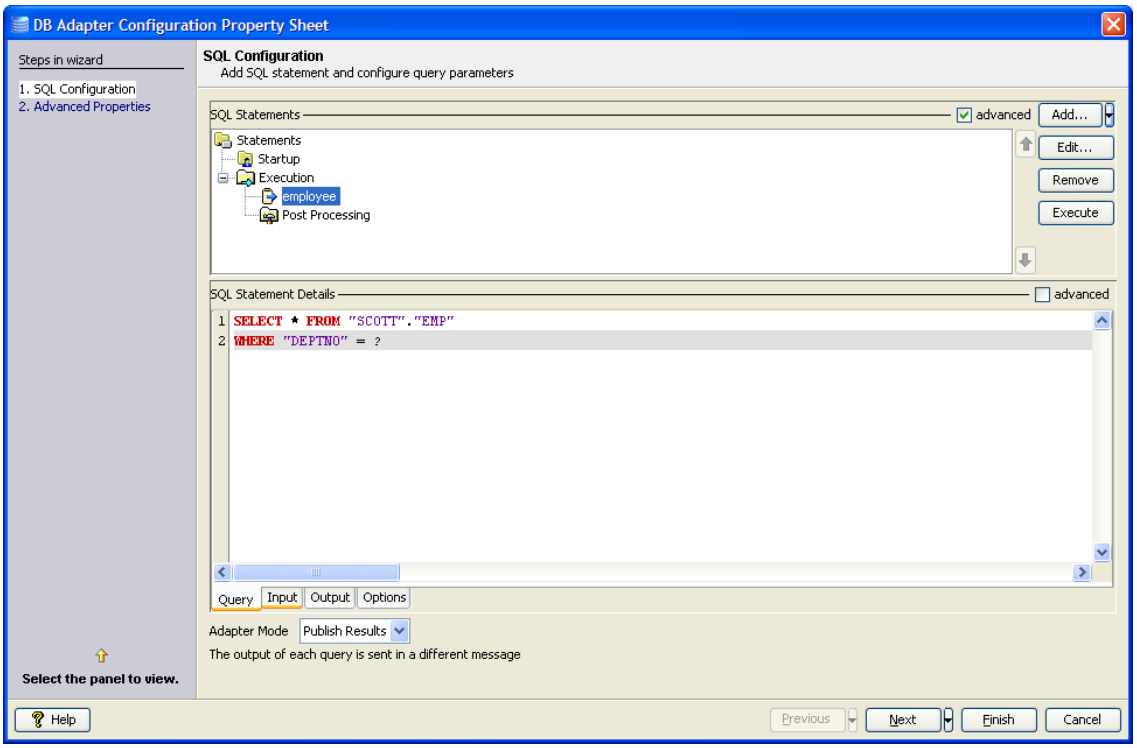


Figure 4.173: SQL Statement Details

Input and output parameters are automatically populated when a query is configured and connection to database is available. However, the populated values can either be modified or defined manually. To define input/output structure manually, a sound understanding of database objects involved is required. ResultSets, parameters, and columns can be added to input or output structure by right clicking **Structure** column.

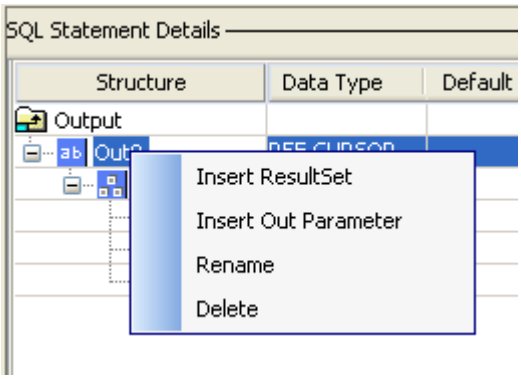


Figure 4.174: Building output structure manually

4.3.1.2.4.1 Configuring Input Parameters

Basic view of input tab is shown in Figure 4.175.

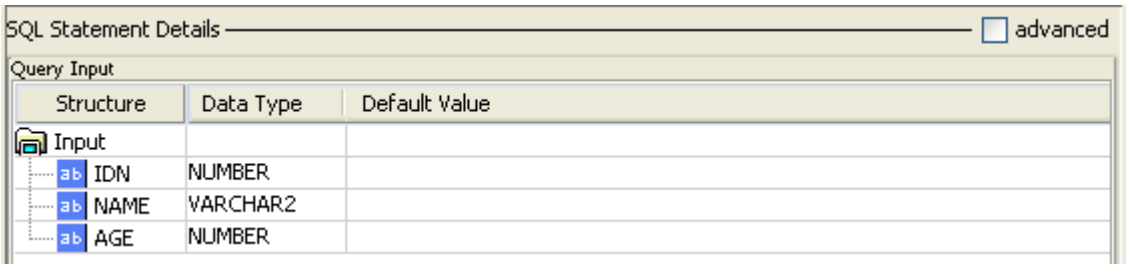


Figure 4.175: Input tab showing basic view of input structure

Check **advanced** check box to see advanced configuration details.

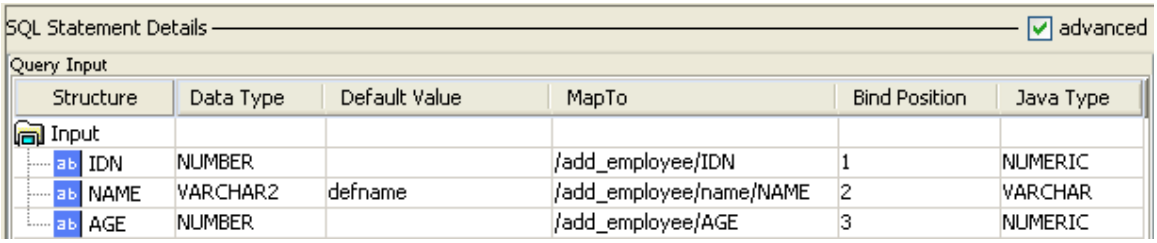


Figure 4.176: Input tab showing advanced view of input structure

Each of the columns in **Input** tab is explained below.

Column Name	Description
Structure	This value is used to generate the schema for the query. In the above figure value for IDNO (field name) is changed to IDN. So the schema generated would contain IDN as the first element instead of default populated value, IDNO.
Data Type	This defines the data type of this column in the database table. This should be correctly defined.
Default Value	This value is taken for the column it is defined against, if the node satisfying the XPath, defined in MapTo, in the input XML is not present. Values $EMPTY_STR and $NULL represent empty string and null values respectively. Note: String literals need not be wrapped in.
Map To	The XPath like expression at which the value for this column is present in the input XML. This can be edited to any value to suit input XML. In case of child queries (nested/post processing/fail over), value from the result of parent can be passed to input of nested query. Value from parent query which should be mapped can be selected from a drop-down list of proprietary expressions ending with the index of output. Further, in case of nested queries, when parent query result is being passed, the following syntax can be used to configure to pass first or last value from list values: $First[<expression>].

Column Name	Description
	$Last[<expression>] Note: Changing this value does not change the input schema. So it is not recommended to change this value.
Bind Position	The position in the query where this value is bound to. Note: Do not change this value.
Java Type	JDBC type which maps to Data Type.

4.3.1.2.4.2 Configuring Output Parameters

Basic view of output tab is shown in the Figure 4.177:

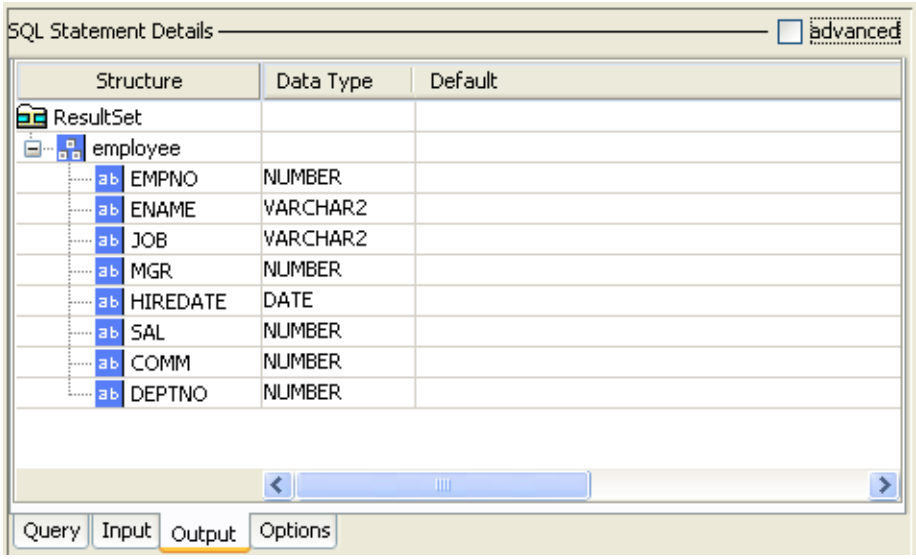


Figure 4.177: Output tab showing basic view of output structure

Check **advanced** check box to see advanced configuration details.

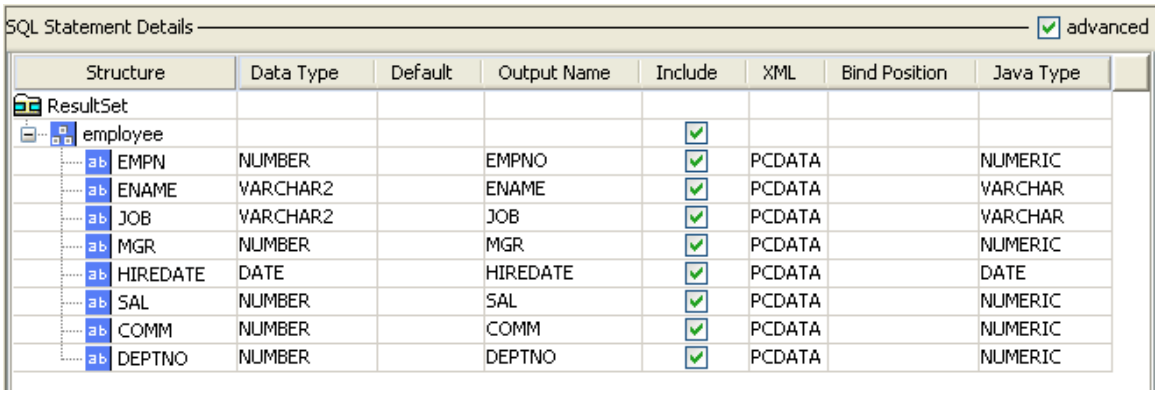


Figure 4.178: Output tab showing advanced view of output structure

Each of the columns in **Output** tab is explained in the table below:

Column Name	Description
Structure	This value is used to generate the schema for the query. In the above figure value for **EMPNO** (field name) is changed to **EMPN**. So the schema generated would contain **EMPN** as the first element instead of default populated value, **EMPNO**
Data Type	This defines the data type of this column in the database table. This should be correctly defined.
Default Value	NA for output
Output Name	NA
Include	If the output XML should contain an element corresponding to column check this check box, else uncheck it. E.g. If the check box against **COMM** is unchecked, the output XML will not contain **COMM** element for any record
XML	NA
Bind Position	NA
Java Type	JDBC type which maps to **Data Type**

4.3.1.2.4.3 Configuring Input/Output Parameters for Inner Queries

The component does not recognize the input/output parameters present in the inner query. These parameters should be manually configured.

If the query has an inner query, when the query wizard is finished, an error occurs. To configure input/output parameters manually, ignore this error.

SQL Statement Details

```
1 SELECT "START_DATE", "SIKYUU_KBN",
2 "HANBAI_TANKA" FROM (SELECT "t1"."START_DATE", "t1"."SIKYUU_KBN",
3 "t1"."HANBAI_TANKA" FROM "ECO"."ECO_KAKAKUHYOU_MEISAI_MST" "t1"
4 WHERE "t1"."KAKAKUHYOU_CD" = '03' AND "t1"."TORISAKI_CD" LIKE '%' AND "t1"."TORISAKI_SITE_CD" LIKE '%' AND
5 "t1"."SEIBAN" = '1002' AND "t1"."HINBAN" LIKE '%' AND "t1"."TUUKA_CD" LIKE '%' AND
6 SUBSTR("t1"."START_DATE",1,8) < SUBSTR('20091015',1,8) AND
7 "t1"."OYA_START_DATE" = '19000101000000 '
8 ORDER BY "t1"."START_DATE" desc)
9 WHERE rownum <= 1
```

Figure: Sample Inner Query

Now, to add Input parameters, click on Input Tab in SQL Statement Details window. Right-click on Input under Structure column and select Add In Parameter.

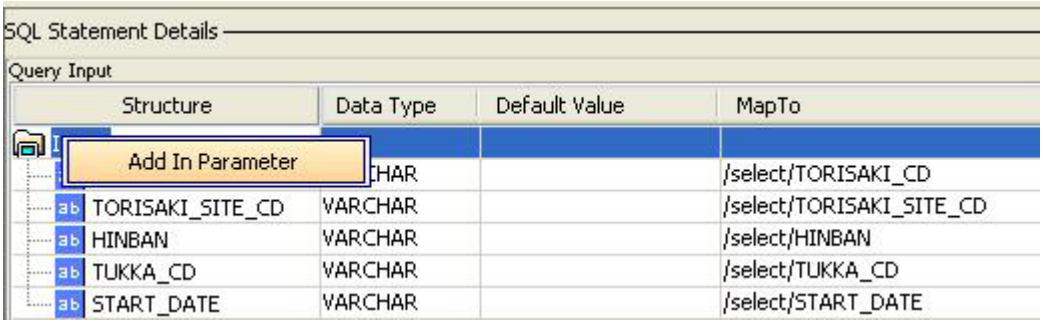


Figure: Configuring Input Parameters for Inner Query

Similarly, to add ResultSets, click on Output tab. Right-click on Output structure and select Add ResultSet.

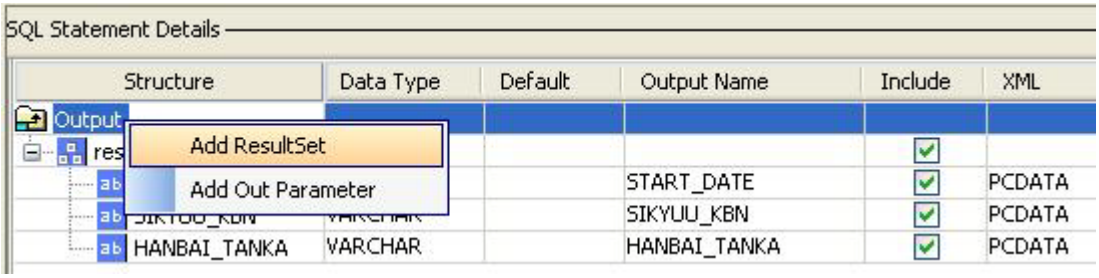


Figure: Configuring Output Parameters for Inner Query

4.3.1.2.5 Editing Query Configuration

4.3.1.2.5.1 Editing DML Statements

1. Select a configured stored procedure under **SQL Statements**.
2. Click **Edit** to launch **Query Builder** in edit mode. This mode is same for all DML statements (Select, Insert, Update, Delete).

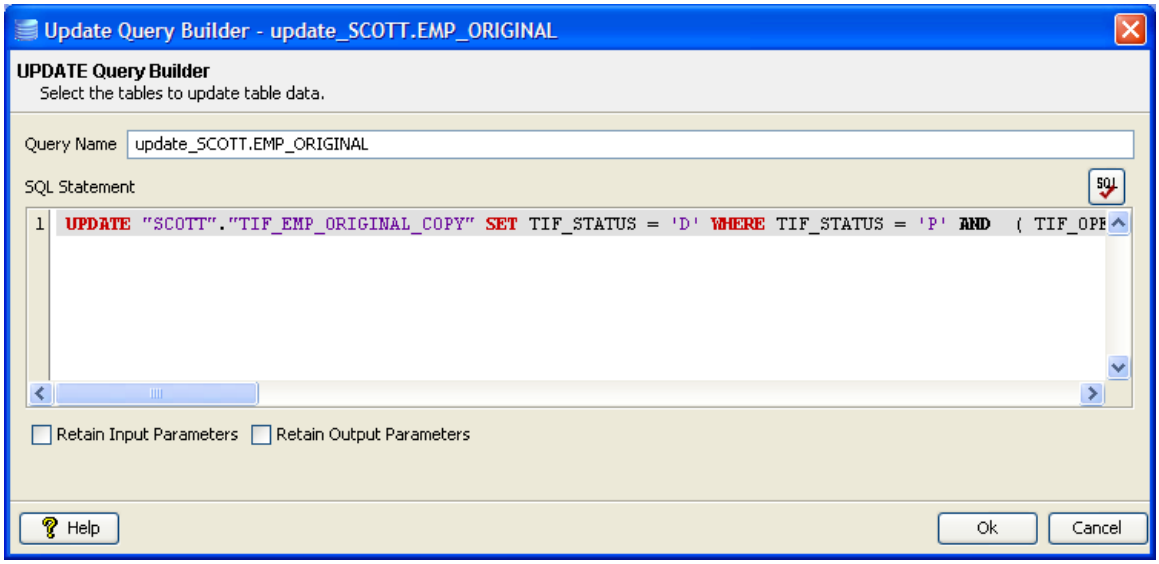


Figure 4.179: Editing configured SQL query

3. Make necessary changes in the **SQL Statement**.
4. The changed SQL can be validated by pressing (**check syntax**) button.

   Note: This feature only checks for invalid tokens, it does not perform a complete syntax check
5. When the dialog is closed, the input / output parameters in **Input** / **Output** tab in **SQL Statement Details Configuration** are regenerated. If these parameter configurations are previously changed from generated values and should not be lost, check **Retain Input Parameters** / **Retain Output Parameters** respectively.
6. Click **Ok** to close the dialog.

4.3.1.2.5.2 Editing Stored Procedure

1. Select a configured stored procedure under **SQL Statements**.
2. Click **Edit...** to launch **Query Builder** for Stored Procedure and follow steps in section Stored Procedure Configuration.

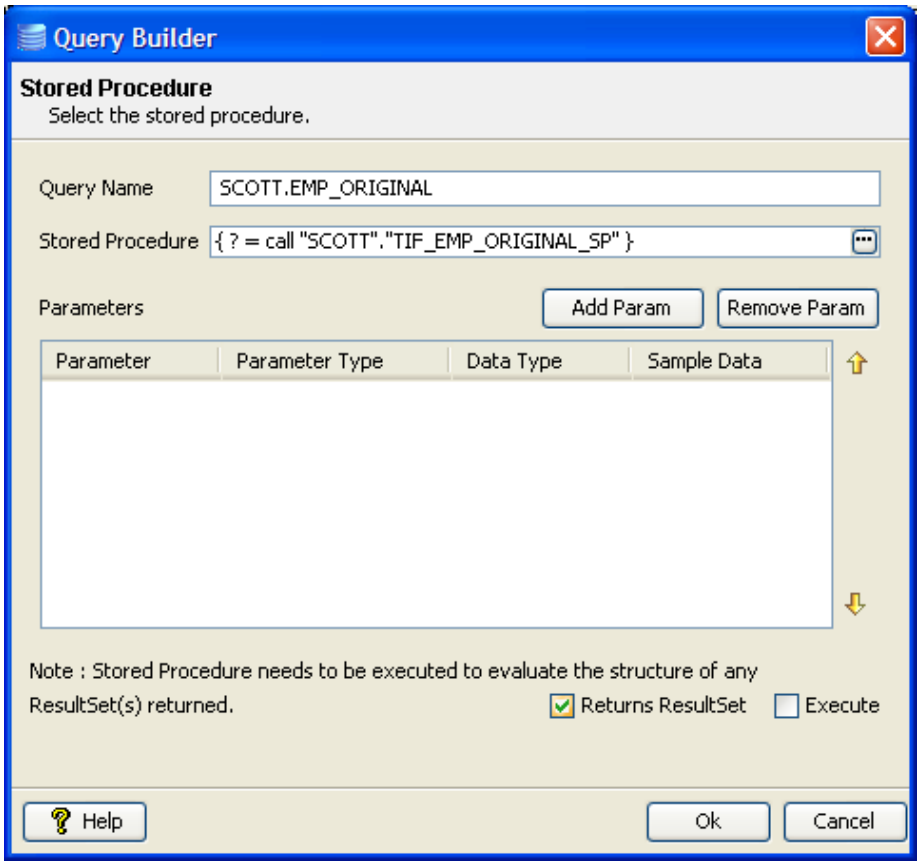


Figure 4.180: Editing stored procedure

3. Check **Execute** check box before closing **Query Builder**, if the structure of result set returned by stored procedure is changed and **Output** tab in **SQL Statement Details Configuration** have to be regenerated.

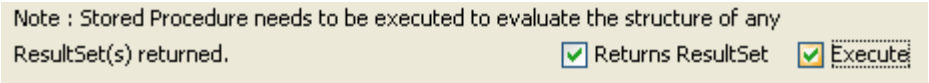


Figure 4.181: Execute option to execute stored procedure

4. Click **Ok** to close the dialog box.

4.3.1.2.6 Removing Query Configuration

Select the query to remove and click **Remove** button.

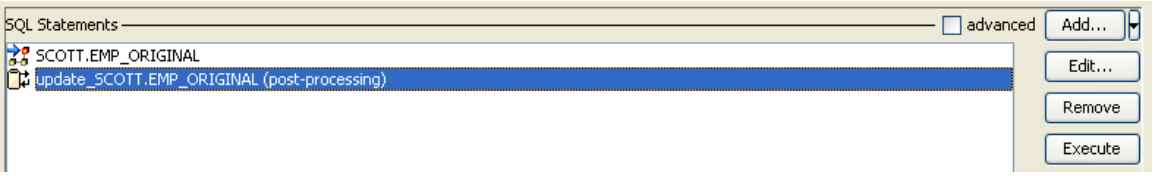


Figure 4.182: Selecting query to be removed

4.3.1.2.7 Testing Query Configuration

1. Any configured can be tested from **SQL Configuration** panel. To test a query, select the query and click **Execute** button.

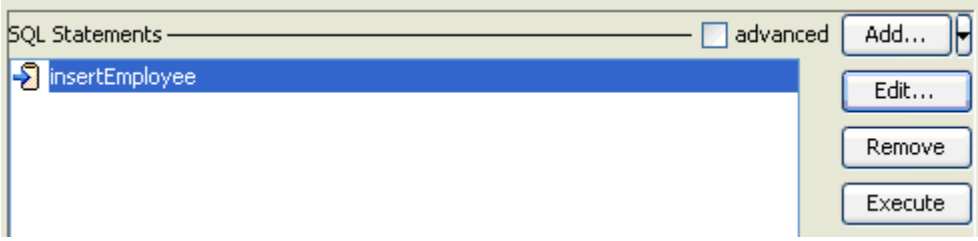


Figure 4.183: Selecting query to be tested

2. **Specify Variable Value** dialog opens.

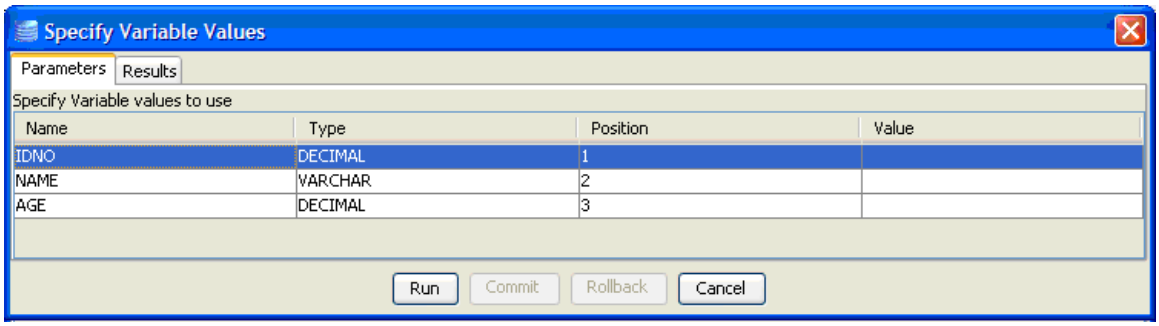


Figure 4.184: Input parameters which require user values

3. Specify values for parameters which require user input (marked **?** in the SQL statement) in the **Parameters** tab under **Value** column. All other columns are not editable.

Example: INSERT INTO "SCOTT"."EMP_ORIGINAL" ("IDNO", "NAME", "AGE") VALUES (?, ?, ?)

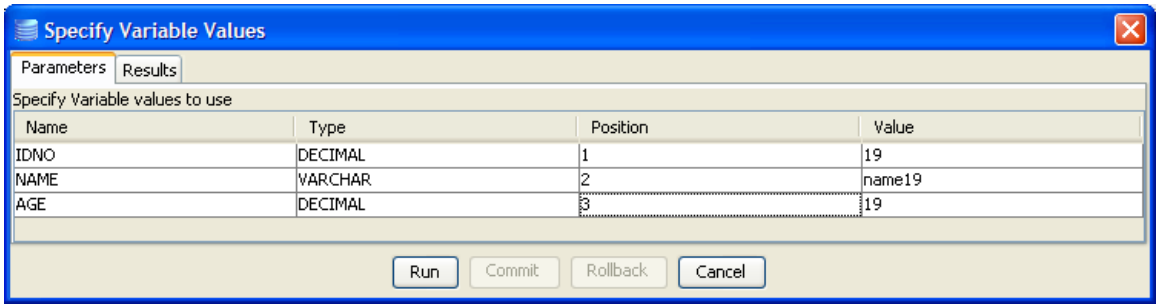


Figure 4.185: Specifying values for input parameters

4. Click **Run**. Result of the query is shown under **Results** tab.
5. Click **Commit** to commit an insert / update or a delete to database, click **Rollback** otherwise.
6. Click **Cancel** to close the dialog.

4.3.1.3 Child Queries

For each configured query, different types of child queries can be configured. Different types of child queries are listed below:

- Nested queries
- Post processing queries
- Failover queries

4.3.1.3.1 Nested Query

A query which executes once for each record returned from parent query.

- Nested query should ideally be configured for select statements.
- Nested query takes values from input.
- Nested query sends values in output.
- Nested query can have a failover query as a child query.

Example:

- For every row in employee table, get the department details to which the employee belongs.
- For every row in employee table, compute total income (salary + commission) and update in incomes table.

4.3.1.3.2 Post Processing Query

A query which is executed after the parent query is executed.

- Post processing query should ideally be configured as an insert or update or delete statements or as a stored procedure which updates database.
- Post processing query takes values from input.
- Post processing query does not send values in output.
- Post processing cannot have any child query.

4.3.1.3.3 Failover Query

A query which is executed when the parent query failed to execute, because of an exception.

- Failover query should ideally be configured as an insert or update or delete statements or as a stored procedure which updates database.
- Failover query should be configured to take same value, as the parent query, from the input XML. This can achieved using **MapTo** column in **Output** tab of **SQL Statement Details Configuration**.
- Output of the parent query and the failover query should match. For example, if both are either insert or update or delete independently, then the output matches (only update count is returned).
- Failover query cannot have any child query.

4.3.1.3.4 Child Query Configuration

1. Configure any query.
2. Check **advanced** check box against SQL Statements.

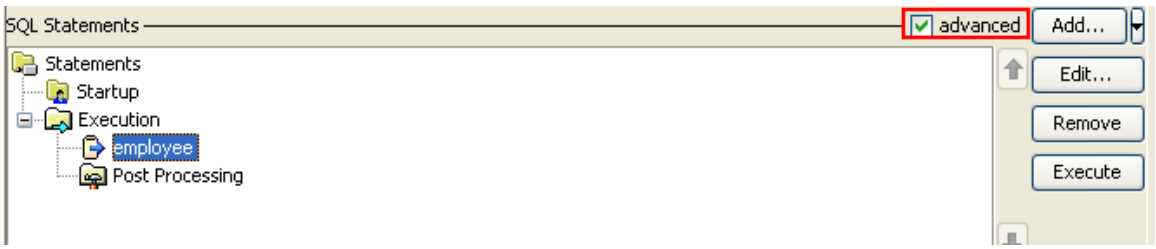


Figure 4.186: Advanced option for SQL Statements

3. Right-click on the query, from the popup navigate to:
    a. **Add Nested Query → <query of interest>** for nested query.
    b. **Define Failover Query → <query of interest>** for failover query.
    c. **Add Post Processing Query → <query of interest>** for post processing query.

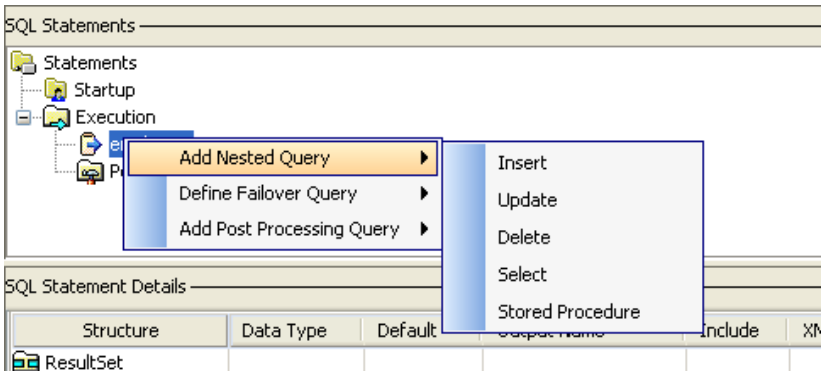


Figure 4.187: Adding a child query

4. A query builder is launched. Refer to appropriate section based on the query that has to be configured. Configured query is shown as a child node to initial query.

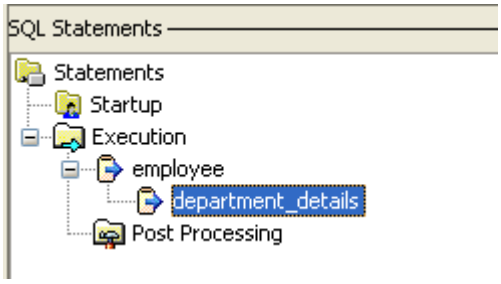


Figure 4.188: Configured query is shown as a child

5. If the child query requires any input, it is by default configured to be taken from input XML. Schema generated on the input port is computed to take inputs for child query as well.
6. Child query can also take input from the result of parent query.
7. To configure child query to take input from the result of parent query
    a. Select the child query.
    b. View **Input** tab in **SQL Statement Details** panel.
    c. Check **advanced** against **SQL Statement Details**.

d. In the **MapTo** column against the required column, click on the drop-down list to see a list of entries, one for each column in the parent queries result, as shown in Figure 3.189.

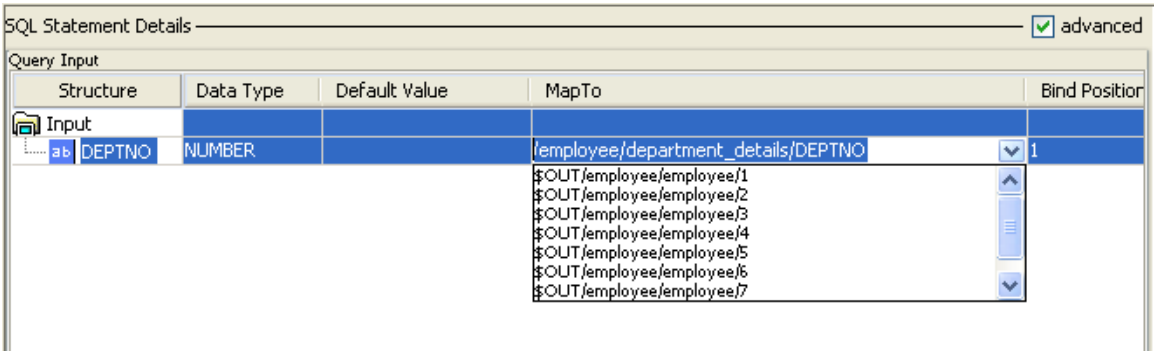


Figure 4.189: MapTo entries for result of parent query

e. From the drop-down list, select appropriate value. For example, Figure 4.190 shows that department number is the eighth field in the output of parent query.

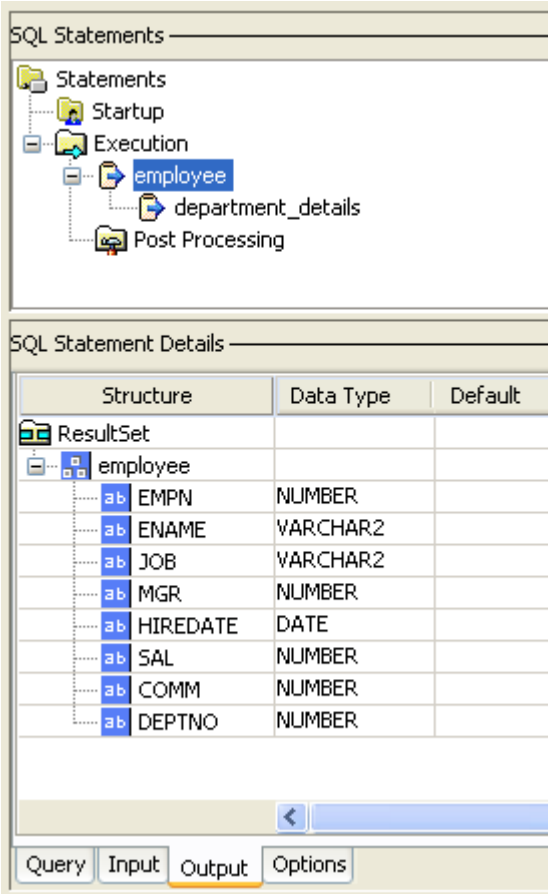


Figure 4.190: Output of parent query

So select **$OUT/employee/employee/8** to map the **DEPTNO** of parent query (**employee**) to input of child query (**department_details**)

Note: $OUT/employee/employee/8 is computed using proprietary formula and should not be modified

f. When parent query returns multiple rows, input for child query can be specified as value at **$OUT/employee/employee/8** from first or last row returned by parent query by using **$First[<MapTo>]** or **$Last[<MapTo>]**, that is, as **$First[$OUT/employee/employee/8]** or **$Last[$OUT/employee/employee/8]**

4.3.1.4 Miscellaneous Configurations

4.3.1.4.1 Request Level Post Processing Query

Post processing query configuration under **Child Queries** executes once for every execution of parent query. Request level post processing query is similar to post processing query with respect to input / output and child queries. However:

- Request level post processing query executes once for each request (input message) after all configured queries are executed, even when multiple queries are configured.
- Request level post processing query has no parent query.

Steps to configure request post processing query:

1. Check **advanced** check box against **SQL Statements**.

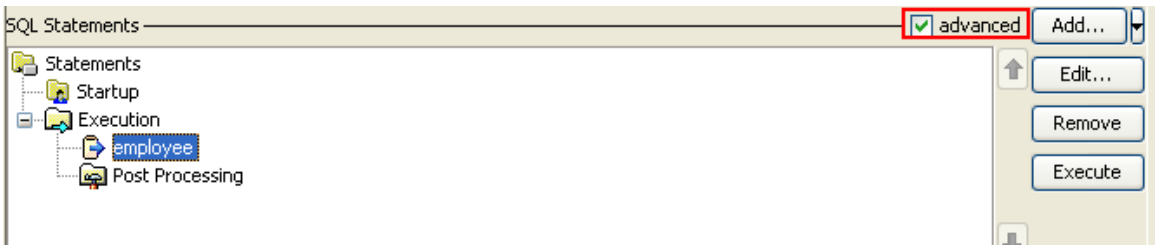


Figure 4.191: Advanced view showing Post Processing

2. Right-click on **Post Processing** and navigate to **Add Query → <query of interest>**
3. A query builder is launched. Refer to appropriate section based on the query that has to be configured.

4.3.1.4.2 Adapter Mode

Adapter mode can be selected from the **Adapter Mode** drop-down list in **SQL Configuration** panel as shown in Figure 4.192:

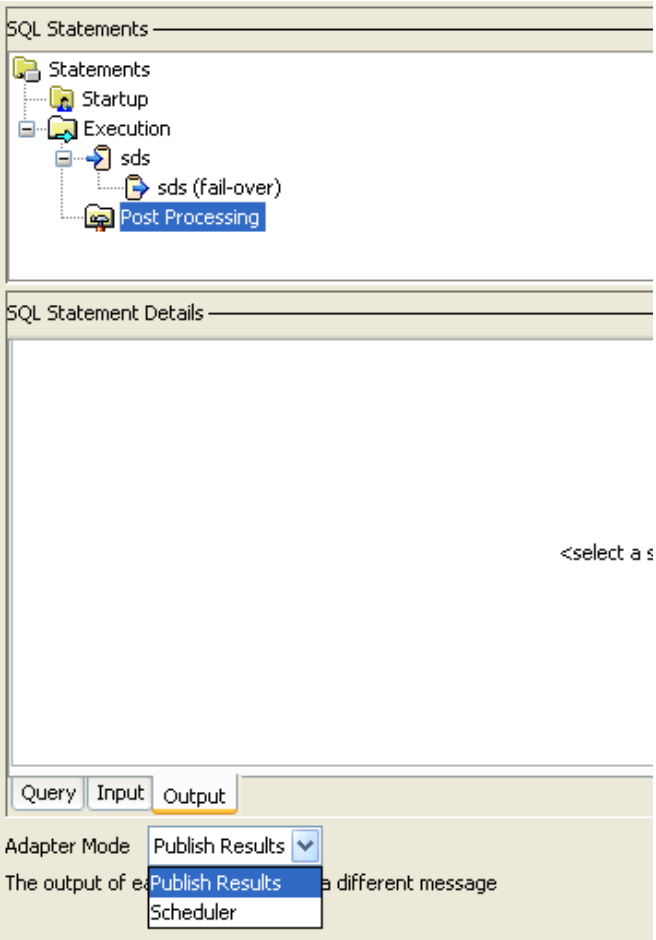


Figure 4.192: Adapter Mode

Publish Results – Component waits for input message and executes when an input message is received.

Scheduler – Component is scheduled and will have no input port. Scheduler configuration can be specified in **Scheduler Configurations** panel.

4.3.1.4.3 Output Options

1. Check **advanced** check box against **SQL Statements** and select **Execution** node.

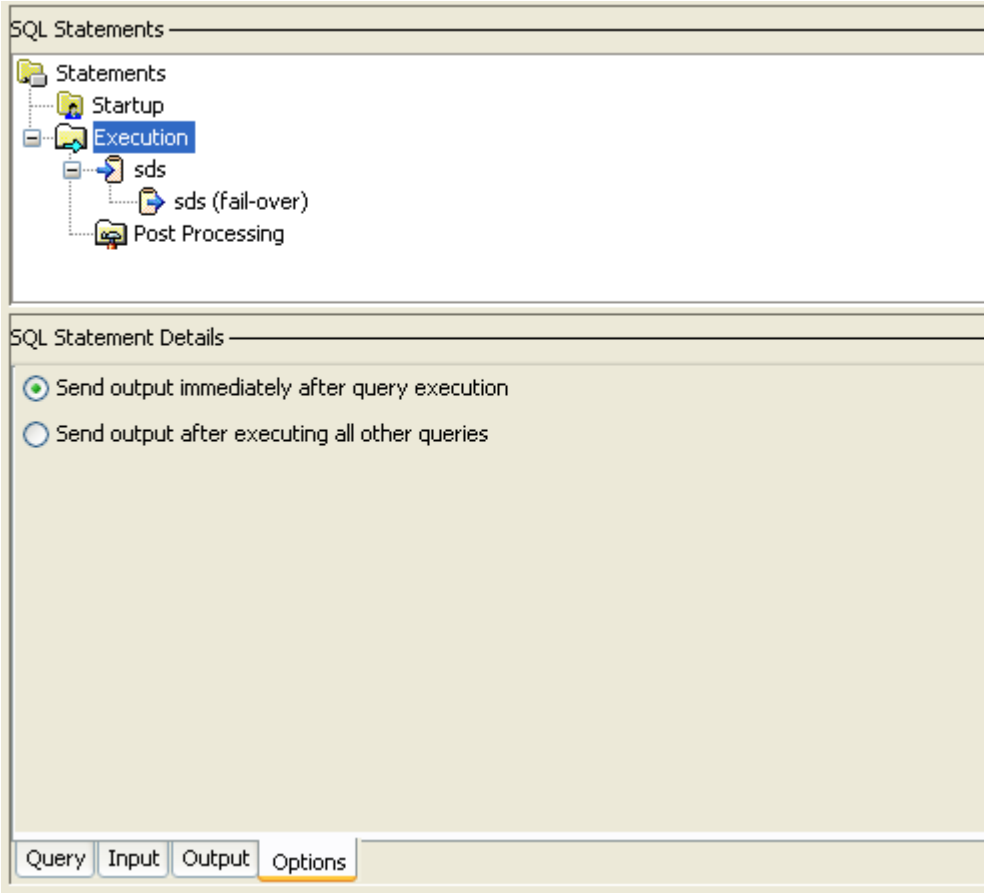


Figure 4.193: Options on Execution for sending output

2. Go to **Options** tab.
   a. Select **Send output immediately after query execution** to send output of each configured query in a separate message.
   b. Select **Send output after executing all other queries** to combine and send output of all queries in one message (as long as total response size does not exceed **Max Response Size** in **Advanced Configuration**).

4.3.1.4.4 Post Processing Execution

1. Check **advanced** check box against **SQL Statements** and select any top level query node.

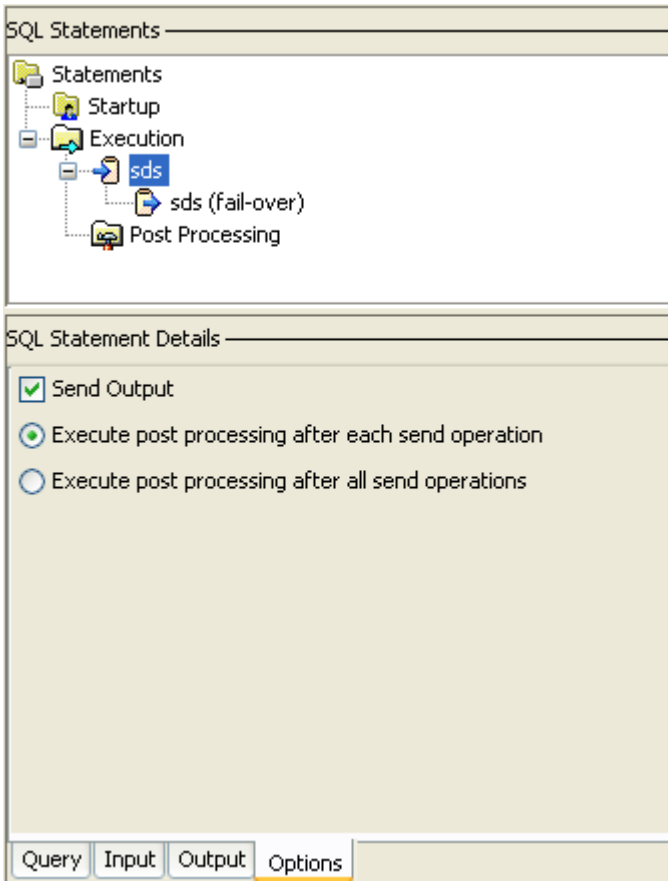


Figure 4.194: Options on configured query for post processing query execution

2. Check **Send Output** check box if the result of the selected query has to be sent in output message, else uncheck.
3. When response size of a query exceeds **Max Response Size** in **Advanced Configuration**, multiple responses are sent for each request. Select **Execute post processing after each send operation** if configured query level post processing query has to executed once for each output message sent, else select **Execute post processing after all send operations**.

Example: If a select statement returns 500 rows and **Max Response Size** in **Advanced Configuration** is configured as 200 rows. A post processing query, if defined, executes 3 times if **Execute post processing after each send operation** is selected, else it is executed once.

4.3.1.4.5 Advanced Configuration

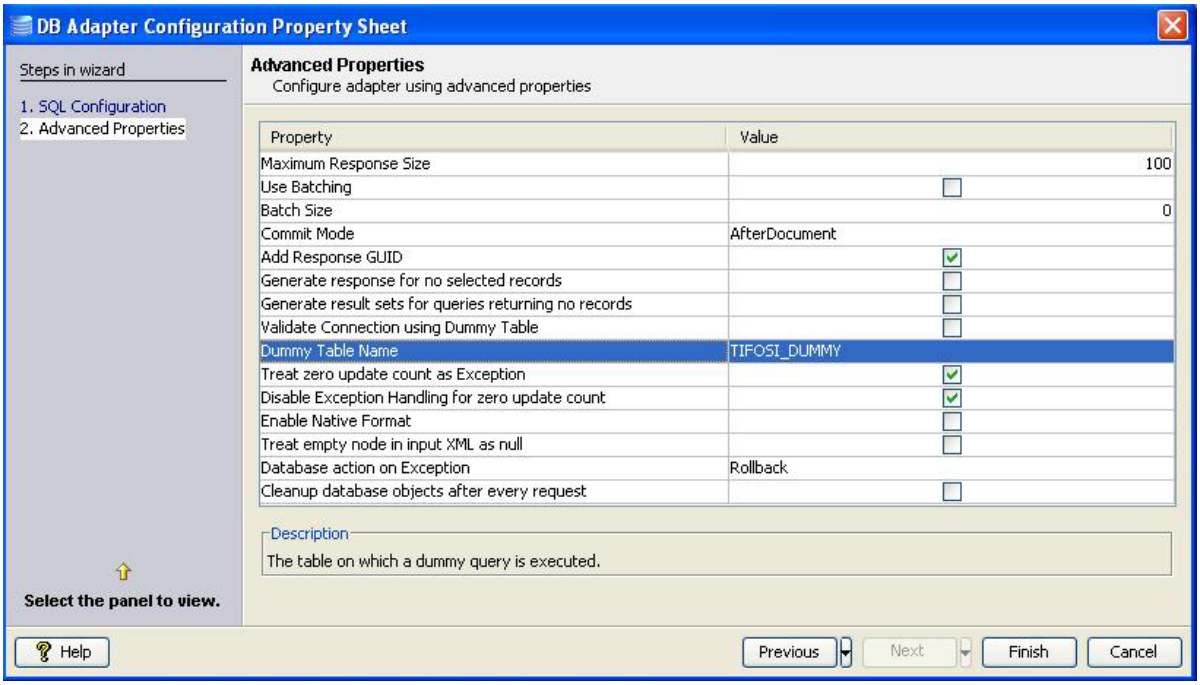


Figure 4.195: Advanced Properties

4.3.1.4.5.1 Maximum Response Size

The maximum number of records each output message can contain.

For example, if a query returns 900 records, and **Maximum Response Size** is set as 200, then for each request there are 5 responses of which 4 responses contain 200 records each and last response contains 100 records.

4.3.1.4.5.2 Use Batching

Determines whether batching should be used or not. Batching should be used only for insert, update or upsert. **After Row** commit mode cannot be used when batching is used. The size of the batch can be specified by the property **Batch Size.**

4.3.1.4.5.3 Use External Transactions

Determines whether External Transactions should be used or not. If this property is set, then the Commit Mode will be automatically set to Transaction. This is the only way to set Transaction as commit mode. User needs to send the commands BEGINTRANSACTION and ENDTRANSACTION to begin and end the external transactions respectively. ROLLBACK command will rollback the transactions done up to that point in the current transaction, but this command will not end the current transaction. DB component commits the database operations done in current transaction and ends the transaction on ENDTRANSACTION command. If the commands ENDTRANSACTION/ROLLBACK or data recieved before BEGINTRANSACTION command, then the component throws an exception of type INVALD REQUEST ERROR

Note: when this property is set, connection pooling should be enabled and number of connections should be set to 1. This is required since a commit or rollback should be done on the same connection object and having more than one connections will not gaurantee same connection received everytime especially if there are multiple sessions enabled on input port resulting in undesired behavior.

Even when the connection pool size is set to 1, multiple connections are maintained. Different connections will be distinguished based on the value of the message property ConnectionID. So this property can be used to correlate different messages to transactions.

4.3.1.4.5.4 Commit Mode

Granularity of transaction is determined by the value specified against **Commit Mode** when **Auto Commit** is set to **no** in MCF panel.

Commit Mode	Granularity / Behavior
Auto	Request database is automatically committed by JDBC driver implicitly when an operation is performed on the database.
After Document	Request – Database is committed after all the queries in the request are executed.
After Row	Query Database is committed after each top level query is executed for one input part in the input document. If there are any nested queries, commit is performed after the nested queries are executed. This is not relevant when **Use Batching** is selected.
After Batch	Request Database is committed after executing **n** top-level queries when **n** is the batch size. If there are any nested queries, commit is performed after the nested queries corresponding to top-level are executed. This mode is visible only when **Use Batching** is selected.
BasedOnInput	Commit is only done when commit instruction is received on the input port. **Note:** While using this commit mode, connection pooling should be enabled and number of connections should be set to 1. This is required since a commit or rollback should be done on the same connection object and having more than one connections will not guarantee same connection received every time especially if there are multiple sessions enabled on input port resulting in undesired behavior. Even when the connection pool size is set to 1, multiple connections are maintained. Different connections will be distinguished based on the value of the message property ConnectionID. So this property can be used to correlate different messages to transactions.
AfterEachOutput	Request Database is committed after sending every output message. If this commit mode is selected then the last message from the component (in case of multiple messages for single request) doesn't contain the property "CLOSE_EVENT" set to true.
BeforeEachOutput	Request Database is committed before sending every output message. If this commit mode is selected then the last message from the component (in case of multiple messages for single request) doesn't contain the property "CLOSE_EVENT" set to true.

If an exception / error (which does not require creating new connection like request processing) occurs during the execution, then the action is taken based on the value for property **Database action on Exception**. If this action is set to **Rollback**, then a rollback is issued and all queries performed after last commit (based on commit mode) will be rolled back, that is, they will not take any effect. If this action is set to **Commit**, then all the queries except to the query that resulted in exception, performed until the exception are committed to database.

If a connection error occurs, then it will be equivalent to the rollback action on exception since commit should be performed on the same connection which executed the queries. Since the connection is no longer present, a new connection is created and all the uncommitted transactions are lost.

Example 1:

```
<ns1:SQL_CFG_1 xmlns:ns1="http://www.fiorano.com/fesb/activity/DB1/Request"
id="7590437537112108032">
 <ns1:insert><ns1:IDNO>401</ns1:IDNO></ns1:insert>
 <ns1:insert><ns1:IDNO>402</ns1:IDNO></ns1:insert>
 <ns1:insert><ns1:IDNO>403</ns1:IDNO></ns1:insert>
 <ns1:insert><ns1:IDNO>404</ns1:IDNO></ns1:insert>

</ns1:SQL_CFG_1>
```

- **After Document** – commits all 4 inserts at once
- **After Row** – commits one insert at a time when **use Batching** is not selected
- **After Batch** – commits after 2 inserts, if batch size is 2 and **useBatching** is selected
- **BasedOnInput** –The transaction will be committed if the following message is received on input port

  <ns1:SQL_CFG_1 xmlns:ns1=\"http://www.fiorano.com/fesb/activity/DB1/Response\" id=\"-4393189459883103232\"><ns1:COMMIT/></ns1:SQL_CFG_1>

  It will be rolled back if the following message is received on input port

  <ns1:SQL_CFG_1 xmlns:ns1=\"http://www.fiorano.com/fesb/activity/DB1/Response\" id=\"-4393189459883103232\"><ns1:ROLLBACK/></ns1:SQL_CFG_1>
- **AfterEachOutput -** commits after sending every message, if **Maximum Response Size** is set to 1 there will be 4 message from the DB component each contains single output record.
- **BeforeEachOutput** - commits before sending every message, if **Maximum Response Size** is set to 1 there will be 4 message from the DB component each contains single output record.

Example 2:

If we have a insert query q1 and a nested update query nq1 and let us assume the input has details for 4 inputs is as shown below

```
<SQL_CFG_1 >
  <q1>
    <col1>col1value1</col1>
    <col2>col2value1</col2>
    <nq1>
```

```
      <ncol1>ncol1value1</ncol1>
      <ncol2>ncol2value1</ncol2>
    </nq1>
  </q1>
  <q1>
    <col1>col1value2</col1>
    <col2>col2value2</col2>
    <nq1>
      <ncol1>ncol1value2</ncol1>
      <ncol2>ncol2value2</ncol2>
    </nq1>
  </q1>
  <q1>
    <col1>col1value3</col1>
    <col2>col2value3</col2>
    <nq1>
      <ncol1>ncol1value3</ncol1>
      <ncol2>ncol2value3</ncol2>
    </nq1>
  </q1>
  <q1>
    <col1>col1value4</col1>
    <col2>col2value4</col2>
    <nq1>
      <ncol1>ncol1value4</ncol1>
      <ncol2>ncol2value4</ncol2>
    </nq1>
  </q1>
</SQL_CFG_1>
```

- When commit mode is **Auto**, 8 commits are done by JDBC driver implicitly, one for each of the inputs for q1 and nq1. If any error occurs when executing nq1 from 3rd input (having value 'ncol1value3' and 'ncol2value3'), then, only the 3rd input nested query will not be present in database irrespective of type of exception and the value for **Database action on Exception**.
- When commit mode is **AfterDocument**, only 1 commit is done by DB component after processing all inputs for q1 and nq1. If any error happens when executing nq1 from 3rd input (having value 'ncol1value3' and 'ncol2value3'), then, if the **Database action on Exception** is set to **commit**, then the inputs for first, second and third 'q1' and the inputs for first and second 'nq1' are committed. If the **Database action on Exception** is set to **rollback**, nothing will be committed for the entire request.
- When commit mode is **AfterRow**, 4 commits are done, this happens after each q1 and nq1 pair are committed successfully to the database. If any error happens when executing nq1 from 3rd input (having value 'ncol1value3' and 'ncol2value3'), then, if the **Database action on Exception** is set to **commit**, then the inputs for first, second, third and fourth 'q1' and inputs for first, second and fourth 'nq1' are committed. If the **Database action on Exception** is set to **rollback**, then the inputs for first, second and fourth 'q1' and the inputs for first, second and fourth 'nq1' are committed.
- When commit mode is **BasedOnInput**, any action will be taken based on input sent. This option is usually used when a commit should be done after processing more than one message.

- When the commit mode is **AfterBatch**, commits are done after processing 'n' inputs from input message where 'n' is the value mentioned against **Batch Size** property. If the Batch Size is 0, then it will be same as **AfterDocument**. So in the Example 2, if the batch size is 2, then only two commits are done. This happens after first and second 'q1', 'nq1' pairs; and second after third and forth 'q2', 'nq2' pairs. If any error occurs when executing 'nq1' from 3rd input (having value 'ncol1value3' and 'ncol2value3'), then, if the **Database action on Exception** is set to **commit**, then the inputs for first, second and third 'q1' and inputs for first and second 'nq1' are committed. If the **Database action on Exception** is set to **rollback**, then the inputs for first and second 'q1' and inputs for first and second 'nq1' are committed.
- When commit mode is **AfterEachOutput**, if the **Maximum Response Size** is set to n, then the number of commits will be: 4 commits if n=1, 2 commits if n=2or 3, 1 commit if n is greater than or equal to 4. Commits will be happen after sending every message. If **Maximum Response Size** is set to 4 and any error happens when executing nq1 from 3rd input (having value 'ncol1value3' and 'ncol2value3'), then, if the **Database action on Exception** is set to **commit**, then the inputs for first, second, third and fourth 'q1' and inputs for first, second 'nq1' are committed, if the **Database action on Exception** is set to **rollback**, then the inputs for first, second and fourth 'q1' and the inputs for first, second 'nq1' are rollbacked. If any exception occurs while executing a third input of q1 then DB component will not process the fourth input.
- When commit mode is **AfterEachOutput**, if the **Maximum Response Size** is set to n, then the number of commits will be: 4 commits if n=1, 2 commits if n=2or 3, 1 commit if n is greater than or equal to 4. Commits will be happen before sending every message. If **Maximum Response Size** is set to 4 and any error happens when executing nq1 from 3rd input (having value 'ncol1value3' and 'ncol2value3'), then, if the **Database action on Exception** is set to **commit**, then the inputs for first, second, third and fourth 'q1' and inputs for first, second 'nq1' are committed, if the **Database action on Exception** is set to **rollback**, then the inputs for first, second and fourth 'q1' and the inputs for first, second 'nq1' are rollbacked. If any exception occurs while executing a third input of q1 then DB component will not process the fourth input.

4.3.1.4.5.5 Batch Size

The Batch size when batching is used. It indicates number of operations of main query that have to be performed in single batch. The value cannot be less than 0. If it is 0, all operations are performed in a single batch. This is valid when the property **Use Batching** is selected.

4.3.1.4.5.6 Add Response GUID

When checked, an additional attribute **id** – if present in input message on **SQL_CFG_1** element, is set onto all output messages for that particular request. If the input message does not contain id attribute, a unique value for each request is generated and set on all output messages for that particular request.

Note: id attribute value can be used to map request with all responses or responses for a particular request.

4.3.1.4.5.7 Generate response for no selected records

When all queries fail to return any data, an empty message is generated if this property is checked else there is no response message coming out.

Example: Assume a DB adapter is configured to get data from tables – table1 and table2 and both table do not have any data in them. If this property is not checked, there is no message from adapter, else following message appears **<SQL_CFG_1/>**

4.3.1.4.5.8 Generate result sets for queries returning no records

When one of the queries does not return any results, an empty element is generated if this property is checked; else it is excluded from result.

Example: Assume a DB adapter is configured to get data from tables – table1 and table2 and table1 has some data but table2 does not have any data in it.

If this property is not checked there output is:

```
<SQL_CFG_1>
                        <table1>
                        …..data here…..
                        </table1>
</SQL_CFG_1>
else following message comes out
<SQL_CFG_1>
                        <table1>
                        …..data here….
                        </table1>
                        <table2/>
</SQL_CFG_1>
```

4.3.1.4.5.9 Validate Connection using Dummy Table

Database connectivity, in case of SQL Exception, is validated by querying a dummy table (created for this purpose alone). Value specified against **Dummy Table Name** is used as the table to query for validating connection failure.

While creating a connection to Database:

- If this option is checked and a table name is specified against **Dummy Table Name**, a table with name as value specified against **Dummy Table Name** is created using the following SQL statement:

CREATE TABLE <DUMMY TABLE NAME>

- If a table with this name already exists, then that table is used for validation
- If a table with this name does not exist and an exception occurs while creating dummy table, then table with this name should be manually created, else any exception is treated as a connection failure exception
- If this option is checked and table name is not specified against **DUMMY TABLE NAME** connection creation fails
- When a SQL exception occurs while executing any query, if this option is checked, connection is validated by executing SELECT COUNT(*) FROM <Start wrap character><DUMMY TABLE NAME><End wrap character>

4.3.1.4.5.10 Dummy Table Name

Name of the table which should be queried to validate connection when a SQL exception occurs while executing any query

4.3.1.4.5.11 Treat zero update count as Exception

For queries returning an update count – **insert** or **update** – an update count of **0** is treated as an exception if this option is checked, else the query execution is assumed to be successful.

Note: This should definitely be checked when an **upsert** query is being used.

4.3.1.4.5.12 Enable Native Format

Sends/accepts binary data contained serialized objects. This option should be used only in case where the output format and input format of data is same (that is similar XSDs if this option is not checked)

Example: In case of database synchronization where data read from one table on a database is inserted without any transformation into exactly same table on a different database, check this option. This option provides better performance, since additional transformation is not required.

4.3.1.4.5.13 Treat empty node in input XML as null

Empty nodes in input XML (for example, <empno/>) implies corresponding column value is treated as a null value if this option is checked and treated as empty string value otherwise

4.3.1.4.5.14 Database action on Exception

When auto commit is not turned on and an exception occurs database transaction is committed if this option is checked and rolled back otherwise. This option provides atomicity for transactions when auto commit is not turned on.

Example: Consider a request containing 10 instances of an insert query is to be executed such that either all 10 queries are executed or none of them have to be executed. To achieve this, set **Auto Commit** to **false** in **MCF** panel, **Commit Mode** to **After Document** and **Database action on Exception** to **false**.

4.3.1.4.6 Using Named Configurations

Changes are done in eStudio w.r.t to Named Configurations. With the implementation of Named Configurations MCF panel is changed as shown in Figure below.

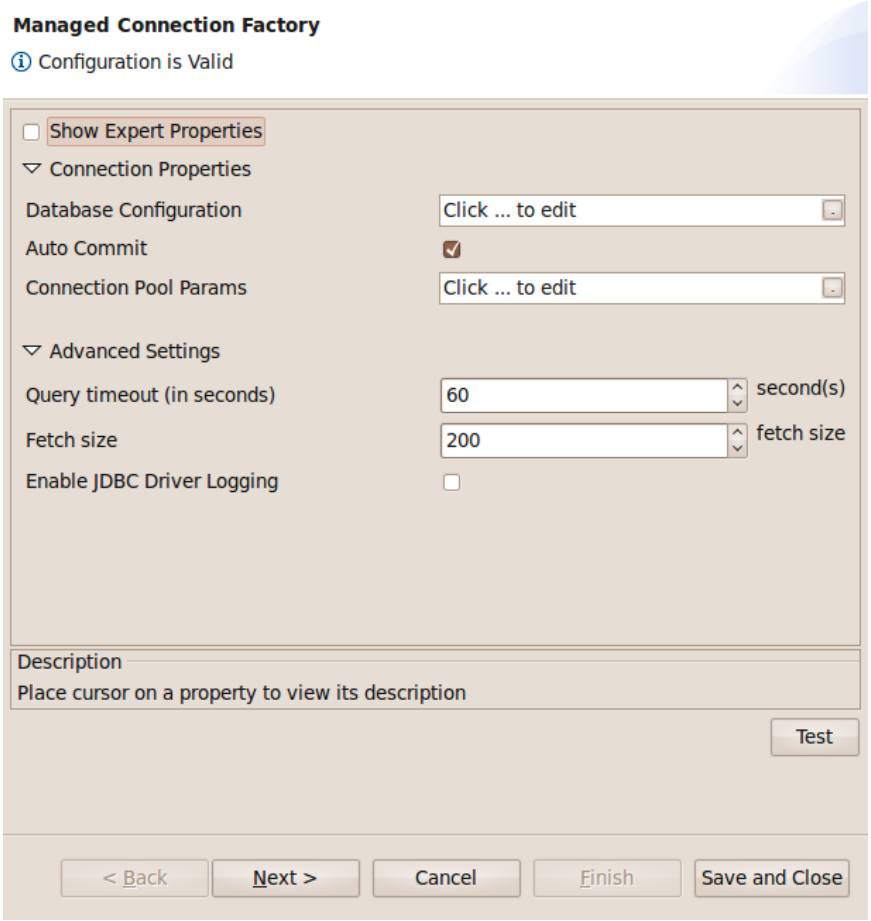


Figure: Modified Managed Connection Factory

Some properties are grouped as one property in the modified configuration panel. Database, Driver, URL, User name, Password, Connection ping sql and Connection Properties which were present previously in the MCF panel are grouped under the name Database Configuration.

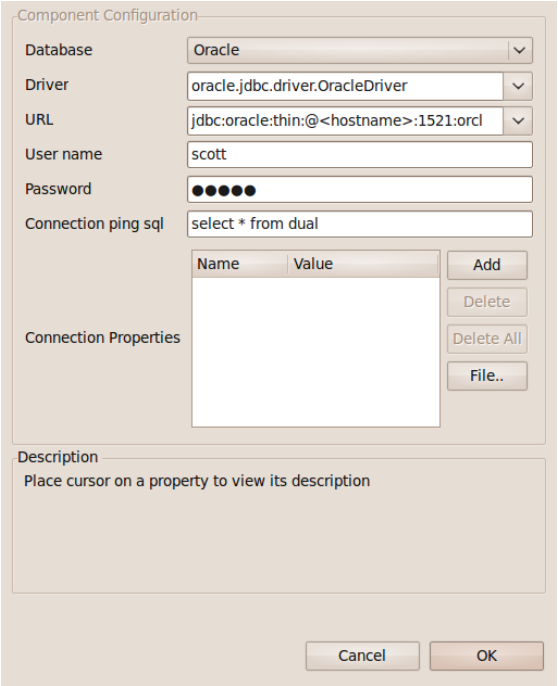


Figure: Database Configuration

4.3.1.5 Input Schema

The input schema is auto generated based on the configuration provided. For the configuration shown above, the schema would be

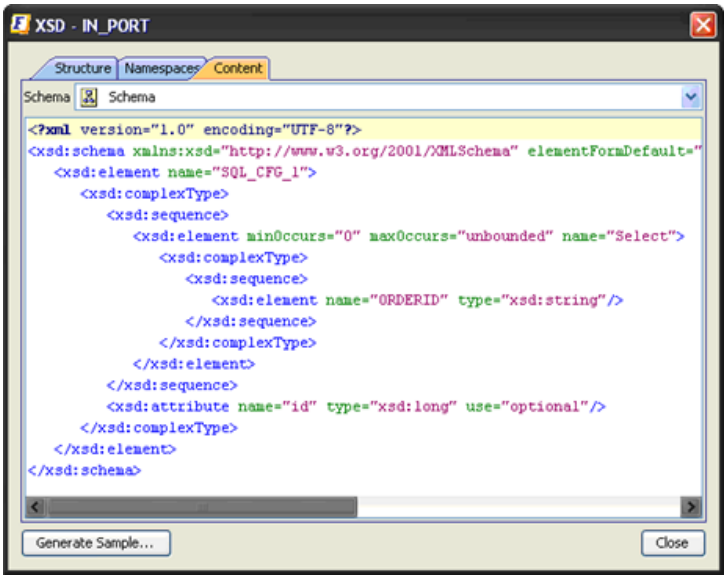


Figure 4.196: Input schema

When the property Use Connection details from input is chosen, an additional element ConnectionFactorySettings is added to the input schema, as shown in the figure. Properties that are used to create the connection are present under this element.

XML Item	Value Type
<ns1:SQL_CFG_1>	
<ns1:ConnectionFactorySettings>?	
<ns1:ConnectionProperties>?	
<ns1:Database>?	HSQL | IBM DB2 | Kingbase | Mckoi | Microsoft Access | M...
<ns1:Driver>?	#string
<ns1:URL>?	#string
<ns1:Username>?	#string
<ns1:Password>?	#string
<ns1:AdvancedConnectionProperties>?	#string
<ns1:AutoCommit>?	true | false
<ns1:AdvancedProperties>?	
<ns1:QueryTimeOutInSec>?	#int
<ns1:FetchSize>?	#int
<ns1:ConnectionPingSQL>?	#string
<ns1:EnableJDBCLogging>?	true | false
<ns1:Select>*	
<ns1:ARG1>	#string

Figure: Input schema with ConnectionFactorySettings

4.3.1.6 Output Schema

The output schema is auto generated based on the configuration provided. For the configuration shown above, the schema would be

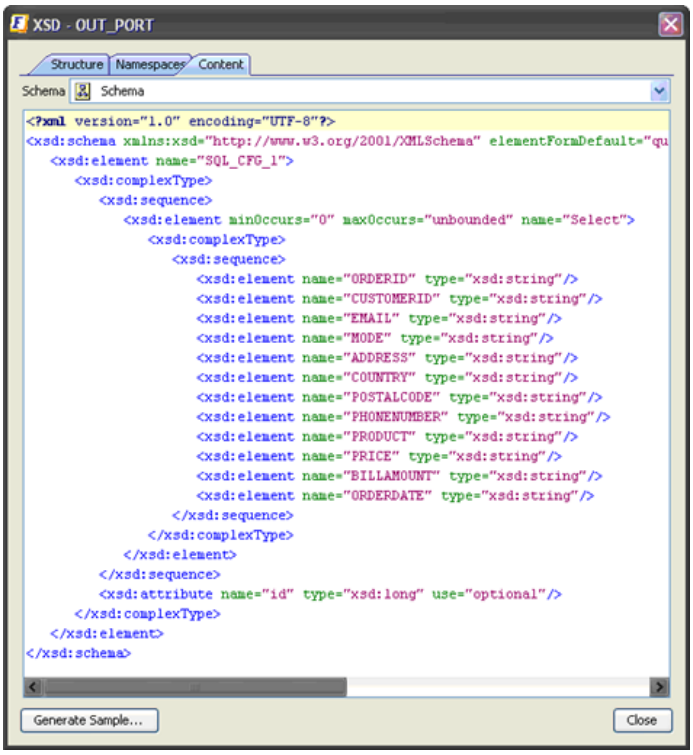


Figure 4.197: Output schema

4.3.1.7 Functional Demonstration

4.3.1.7.1 Scenario 1

Executing multiple queries using a DB component: The given scenario executes a select query and if successful executes an update query which changes the e-mail address of the same record which was selected.

Configure the DB component as described in section *Configuration and Testing* and use feeder and display component to send sample input and check the response respectively.

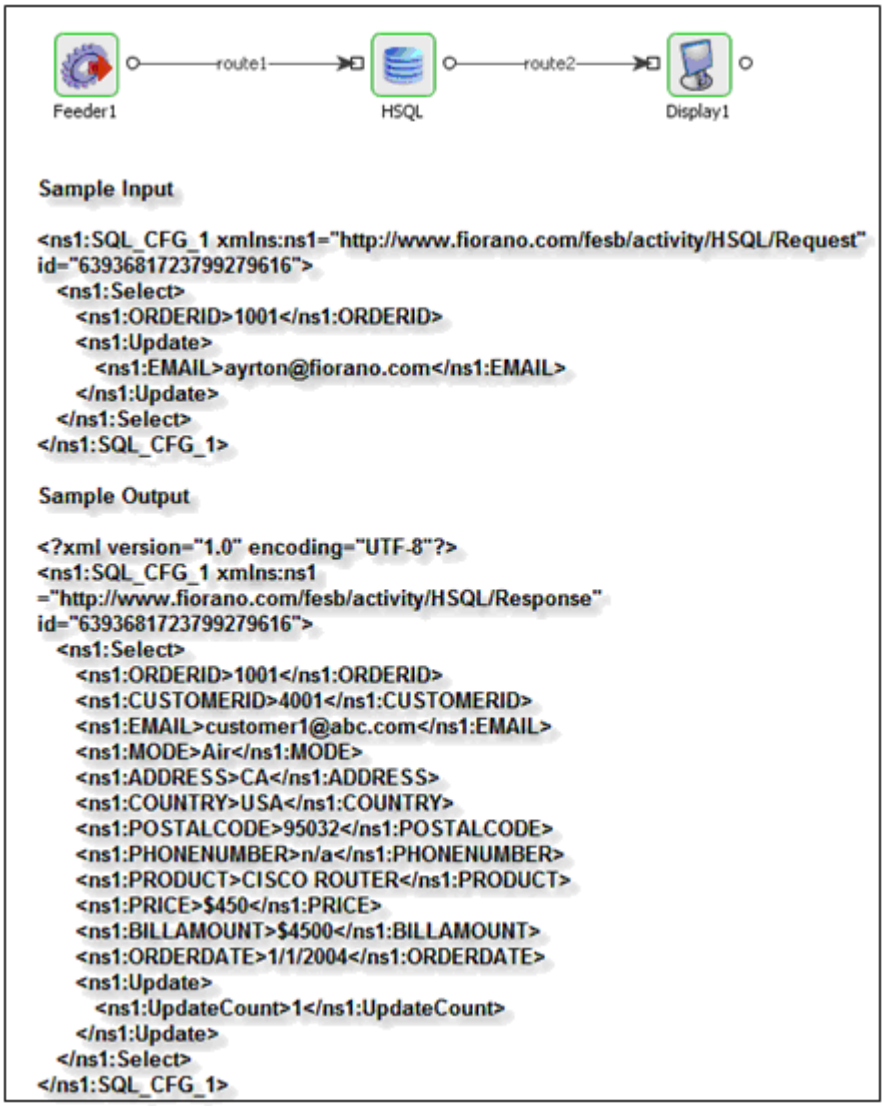


Figure 4.198: Demonstrating Scenario 1 with sample input and output

4.3.1.8 Use Case Scenario

4.3.1.8.1 Scenario 1

In a **database replication** scenario, updates to one database need to be monitored and subsequently updated in another database.

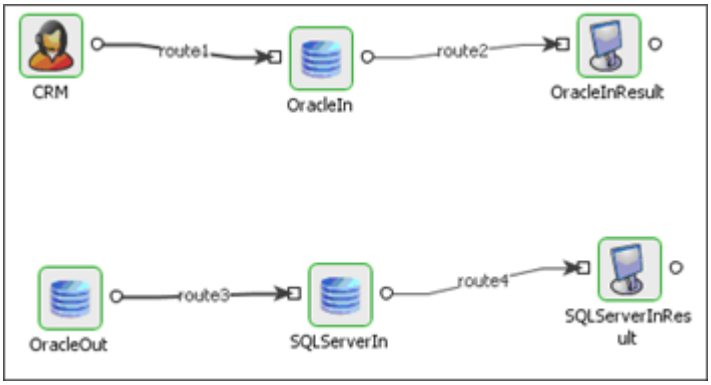


Figure 4.199: DB replication demonstration

The event process demonstrating this scenario is bundled with the installer.

Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Studio.

4.3.1.8.2 Scenario 2

In **DB transaction support** scenario, transactions can be done across multiple steps in an event process.

The event process demonstrating this scenario is bundled with the installer.

Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Studio.

4.3.1.9 Scheduling

In the DB component, scheduling cannot be directly enabled from the scheduling panel. Scheduling can be enabled in the SQL configuration panel. The scheduling interval and rate is determined in the scheduling panel. This is set to scheduler by default when **Monitor Table** option is chosen.

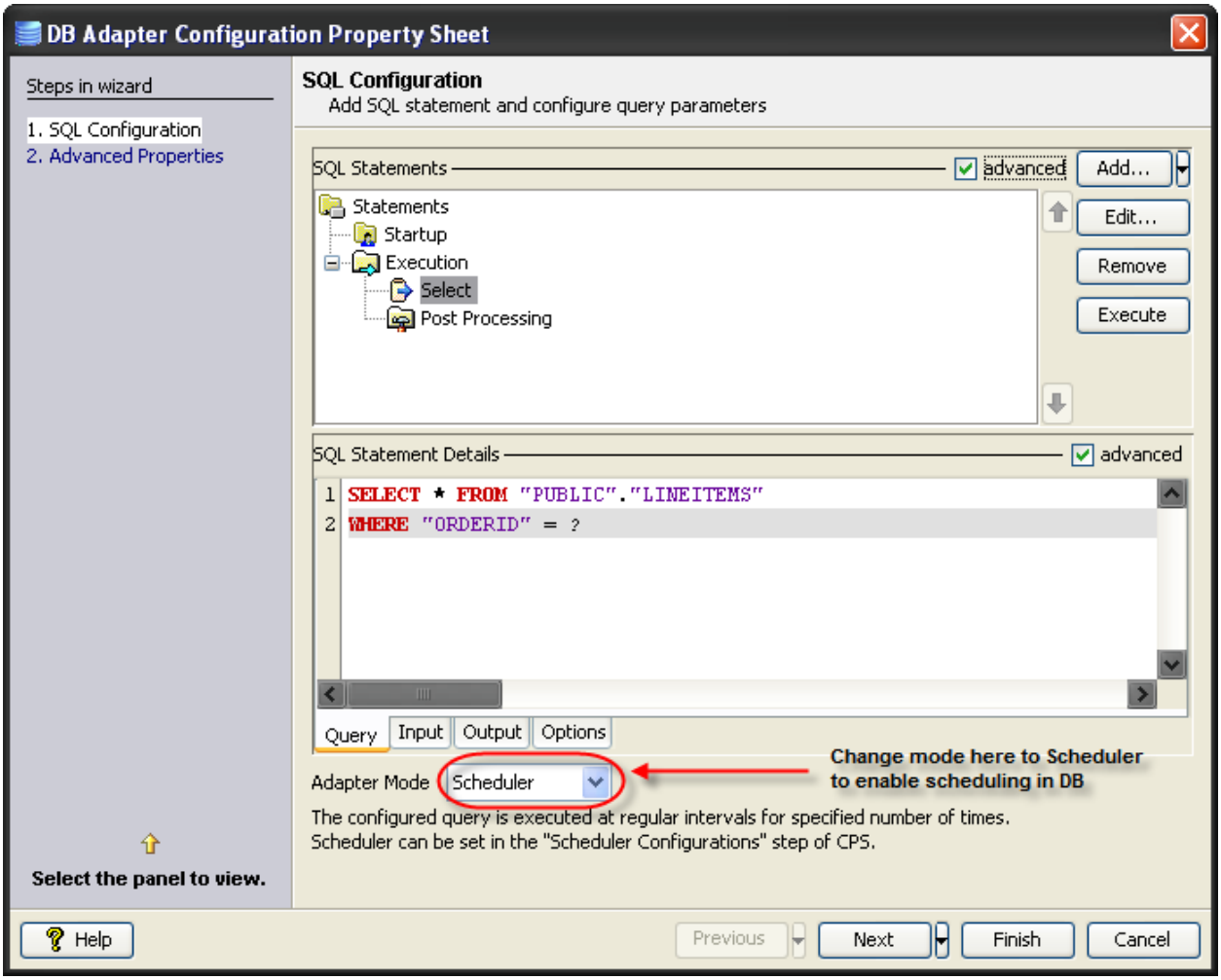


Figure 4.200: DB adapter Configuration Property Sheet – SQL Configuration

4.3.1.10 Useful Tips

5. It is recommended NOT to use JDBC-ODBC Bridge driver to connect to any RDBMS in a production environment. Please use a commercial JDBC driver instead.
6. Configuring DB adapter to read Ms Office 2007 excel sheet(.xlsx) :
    - If the machine has Ms office 2007 already installed then this step can be ignored. Otherwise, install the 2007 Office System Driver from the link given. http://www.microsoft.com/downloads/en/details.aspx?FamilyID=7554F536-8C28-4598-9B72-EF94E038C891&displaylang=en

- Change the URL in Connection Properties(MCF) to "jdbc:odbc:Driver={Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)};DBQ=(provide file path here).

7. The JDBC drivers or the resources must be directly added onto the JDBC system lib and not as resource to the DB component itself. To add JDBC drivers to DB component, please refer to section 3.3.3.3.
8. To connect to MSSQL server, the following changes have to be done.
   a. sqljdbc_auth.dll must be loaded.
      This can be done by adding system property "java.library.path" and set it to the path of the directory which contains sqljdbc_auth.dll file.
      To resolve this file during configuration of component, navigate to Tools->Options->Service Wizard and add "java.library.path" system property.
      To resolve this file during component runtime add "-Djava.library.path=$directory_of_sqljdbc_auth.dll" to runtime arguments of the component.
   b. "integratedSecurity=true" should be appended to the database connection URL.
      Example : To connect to MSSQL 2005 the databse connection URL should be "jdbc:sqlserver:$server_host:$port;databaseName=$database_name;integratedSecurity=true"
   c. Username and password need not be provided additionally.

4.3.2 DBProc

The DBProc component is used to execute Database Stored Procedures. The CPS of this component allows configuring stored procedures for execution using the **Design mode** in CPS. There is no coding effort involved in the configuration.

Points to note:

- Only one stored procedure can be configured in the adapter. Please use the DB component if multiple stored procedures need to be configured for a single instance.
- It is recommended NOT to use JDBC-ODBC Bridge driver to connect to any RDBMS in your production environment. Please use a commercial JDBC driver instead.
- The JDBC drivers or the resources must be directly added onto the JDBC system lib and not as resource to the DBProc component itself. To add JDBC drivers to DBProc component, please refer to section 3.3.3.3.

4.3.2.1 Managed Connection Factory Panel

Connection details are configured in the **Managed Connection Factory** (**MCF**) panel.

Figure below illustrates the panel with expert properties view enabled.

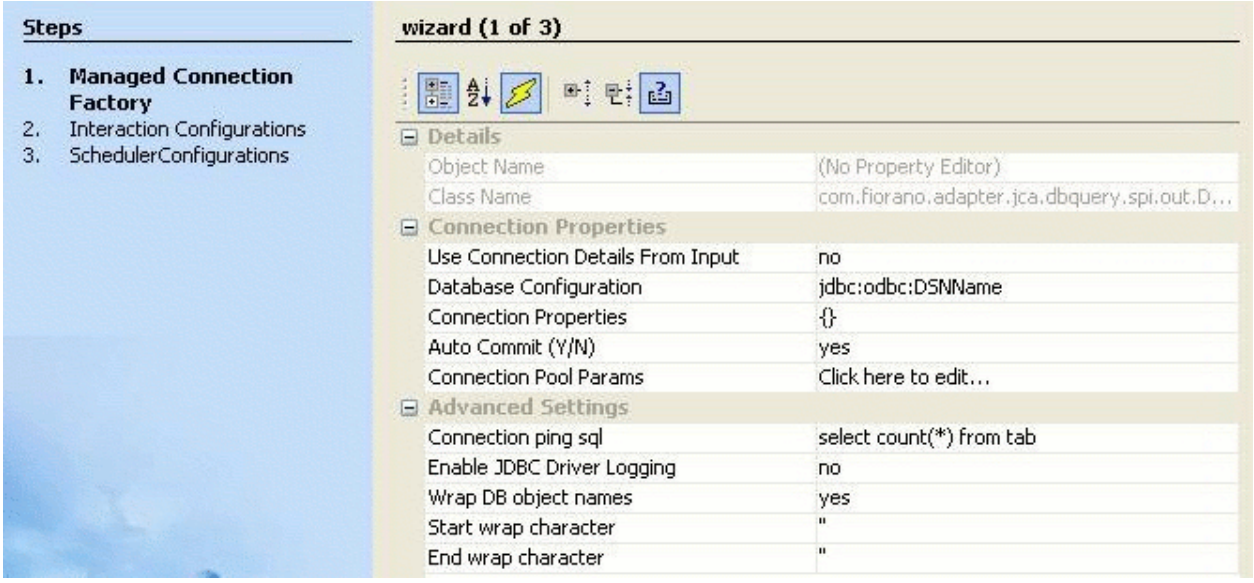


Figure1: Connection configuration details in MCF panel

4.3.2.1.1 Connection Properties

4.3.2.1.1.1 Use Connection Details From Input

Parameters to create the connection can be specified in the input message when this property is set to true. If this property is selected the validation errors in the managed connection factory panel of the CPS are treated as warnings. So user can bypass this step without giving valid configuration and complete the configuration of the component. If valid properties are not provided even in the input message exception will be thrown at runtime.

4.3.2.1.1.2 Database Configuration

Click ellipsis button to launch **Database Configuration** panel shown in figure 2. Details of the database to which the component should be connected and configured in this panel.

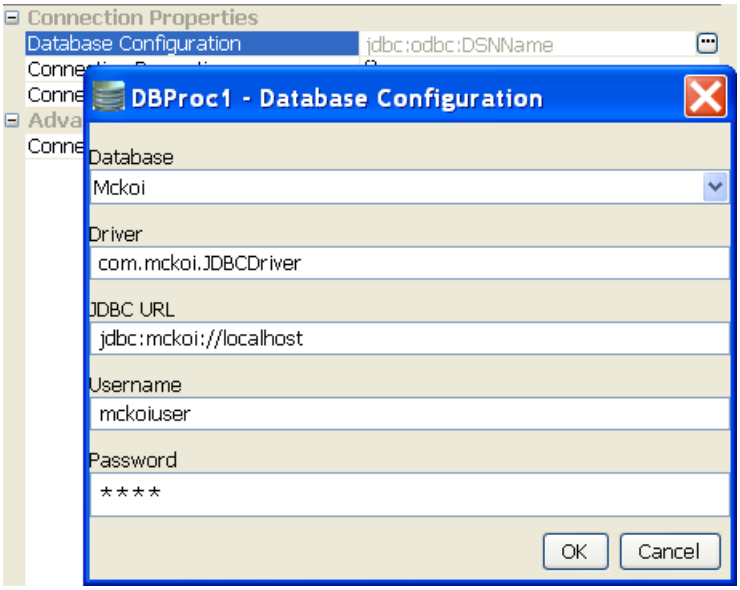


Figure 2: Database Configuration editor with mckoi database details

- **Database**

  This property determines the vendor of the database to which the component has to connect.

  - Vendor name is marked in red if the default JDBC driver class is not present in component's class path.
  - Even if a particular database vendor name is not present in the drop-down list, the component can still connect to the database.
  - If a vendor name is specified in the drop-down list, it only means that vendor specific handling is done. Example: vendor specific handling for data types, naming conventions and so on.
  - To connect to a database from a vendor whose name is not specified in the drop-down list, select Other and provide the correct values for **Driver** and **JDBC URL**.

- **Driver**

  Driver class name that should be used to connect to the database. On selecting required value for **Database**, Driver value is populated with standard value (This can be changed to required value based on driver being used).

  Note: The jar file(s) that are part of JDBC client libraries for selected vendor have to be added as resources to **JDBC** system library.

- **JDBC URL**

  This property determines the location at which the required database is running. On selecting required database, **URL** value is populated with standard value (This can be changed to required values based on driver being used).

  Note: The populated value will have place holders which have to be replaced to point to the correct database location.

Example: In figure 2 **<hostname>** is replaced with localhost IP, indicating that the database is running on a local machine.

- **Username**

  User name that should be used to connect to the database.

- **Password**

  Password of the specified user.

4.3.2.1.1.3 Connection Properties

Any driver specific connection properties which may have to be passed while creating a JDBC connection should be provided against Connection Properties as shown in figure 3. For example, fixedString=true uses FIXED CHAR semantics for string values in oracle.

Note:

- Please refer to documentation of driver that is being used for valid name-values for connection properties.
- Connection properties can be loaded from a properties file using **File...** button. Refer to http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html#load(java.io.InputStream) for details on properties file.

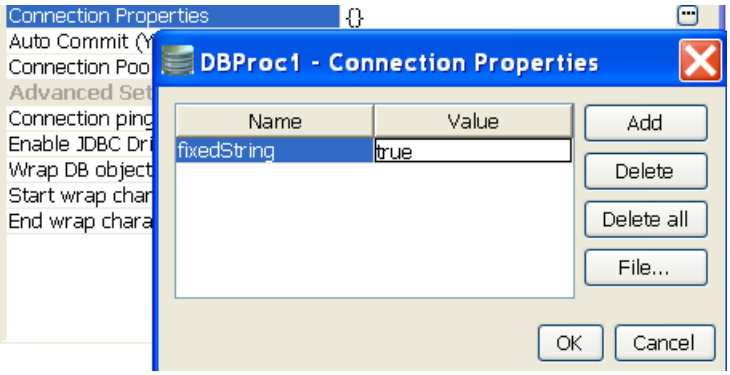


Figure 3: Connection property for oracle

4.3.2.1.1.4 Auto Commit

Commit mode that should be used by the JDBC connection.

- **yes**

  Any transactions (queries executed) will be automatically and implicitly committed to the database. This is done even before the response is generated.

- **no**

  Any transactions (queries executed) will be committed after the request is processed successfully and response is generated, but before the message is sent out of the component.

4.3.2.1.2 Advanced Settings

4.3.2.1.2.1 Connection ping sql

A SQL statement which is guaranteed to execute without exception, except when connection to database is lost. When a SQL exception occurs on executing a configured query, this SQL statement gets executed. If execution of this SQL statement fails as well, then it is assumed that connection to database is lost and appropriate configured action say, reconnect) is taken. For example, select * from dual for oracle, select 1 for MS SQL

4.3.2.1.2.2 Enable JDBC Driver Logging

Value yes implies that logging at the driver level should be enabled. This is used as a debugging option.

4.3.2.1.2.3 Wrap DB object names

When database object names (table names, column names, schema names…) contain spaces, they should be wrapped in database dependent special characters. For example, **" "** for **Oracle**, **[ ]** for **Microsoft Excel** and no wrap characters for **MySQL**.

Database object names are wrapped as shown below -

Start wrap character + object name + End wrap character

Note: Providing a wrong wrap character may lead to problems.

4.3.2.1.2.4 Start wrap character

Character which should be used before the object name.

4.3.2.1.2.5 End wrap character

Character which should be used after the object name.

4.3.2.2 Interaction Configurations Panel

Business logic configuration details are configured in the second panel, **Interaction Configurations**. Figure 4 illustrates the panel with expert properties view enabled.

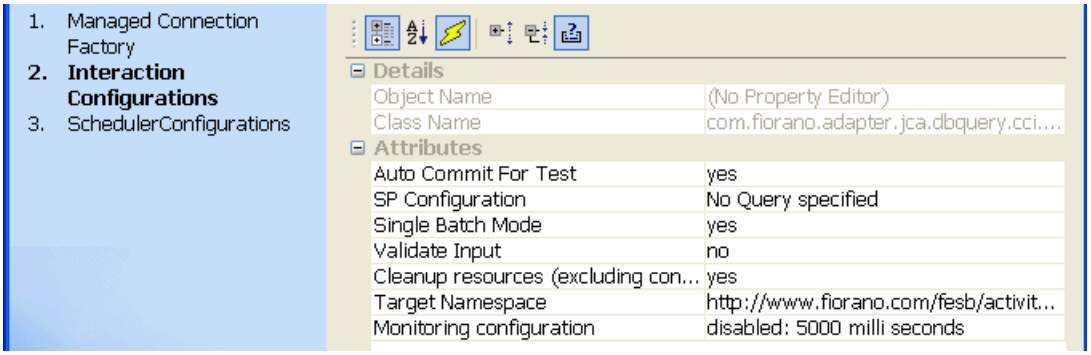


Figure 4: Business logic configuration in Interaction configurations panel

4.3.2.2.1 Attributes

4.3.2.2.1.1 Auto Commit For Test

This property determines whether auto-commit should be enabled when testing from the CPS.

- **yes**

  Any transactions (queries executed) will be automatically committed to the database while testing. Performed transactions will have to manually undone

- **no**

  Any transactions (queries executed) will be rolled back at the completion of test

This property will override the value provided for property **Auto Commit (Y/N)** in the **MCF** panel.

4.3.2.2.1.2 SP Configuration

1. Click **ellipsis** button against property **SQL configuration** property will launch the **SP Configuration** wizard which can be used to configure call statement.

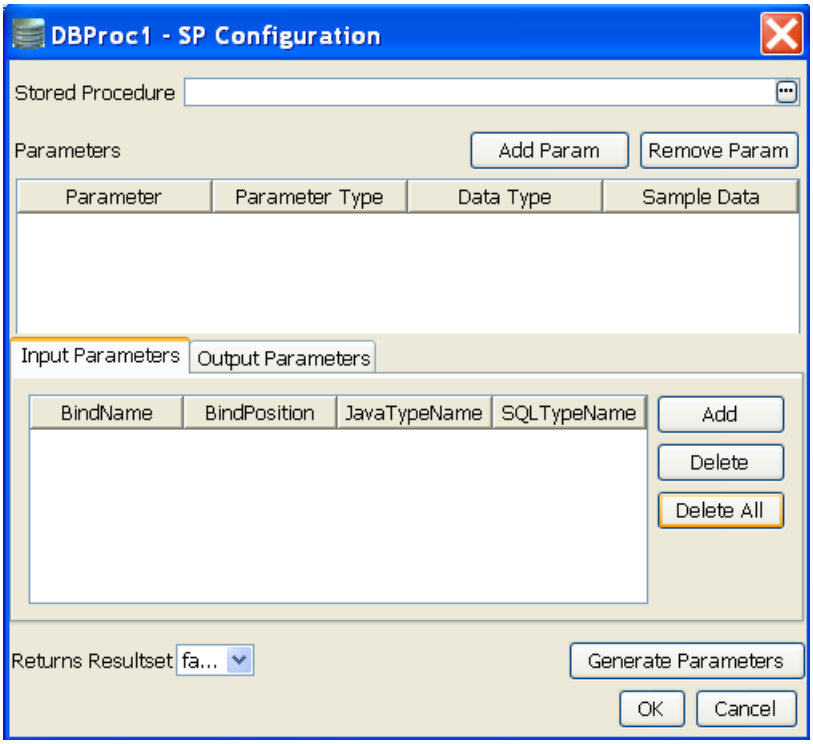


Figure 5: SP Configuration Wizard

2. Click ellipsis button against Stored Procedure to launch Procedure Selection dialog box. Select the Procedure/Function which has to be executed.

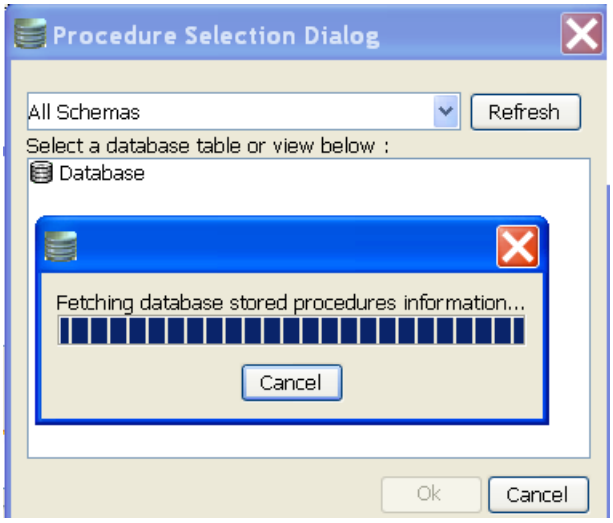


Figure 6: Procedure Selection Dialog

3. We can filter the Procedure/Function by selecting the required schema and clicking **Refresh** button.

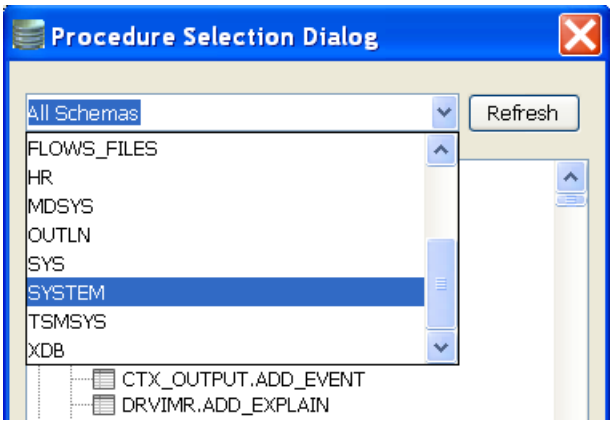


Figure 7: Filtering Stored Procedures based on Schemas

4. Parameters and their configurations are automatically populated.

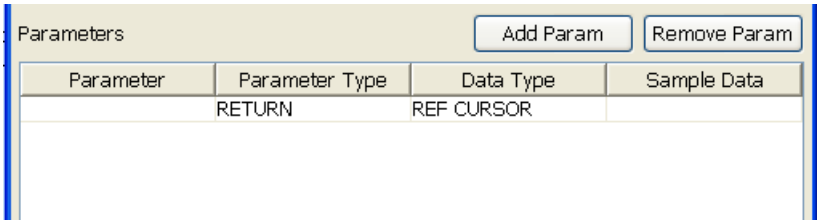


Figure 8: Populated Parameters

Column	Description
Parameter	Name of the parameter for named parameters, blank otherwise
Parameter	Type of parameter – IN, OUT, INOUT, UNKOWN, RETURN, RESULT

Column	Description
Type	Values of type OUT, INOUT, RETURN, RESULT form output structure
Data Type	SQL data type of the parameter
Sample Data	NA

5. For parameters whose data type is a user defined data type **Data Type** column will be populated by value **OBJECT** as shown in figure 9.

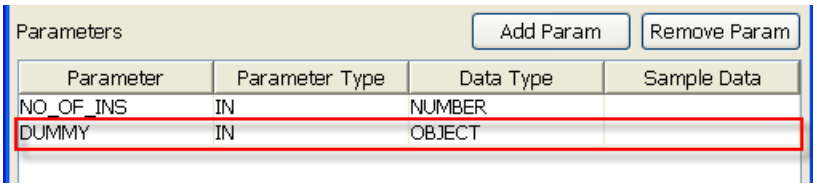


Figure 9: populated Parameter of type User defined

6. We need to explicitly select the user defined data type from **User Types Selection** dialog. Select the **User Defined Type** from the **Data Type** drop-down list as shown in figure 10 to launch **User Types Selection** dialog.

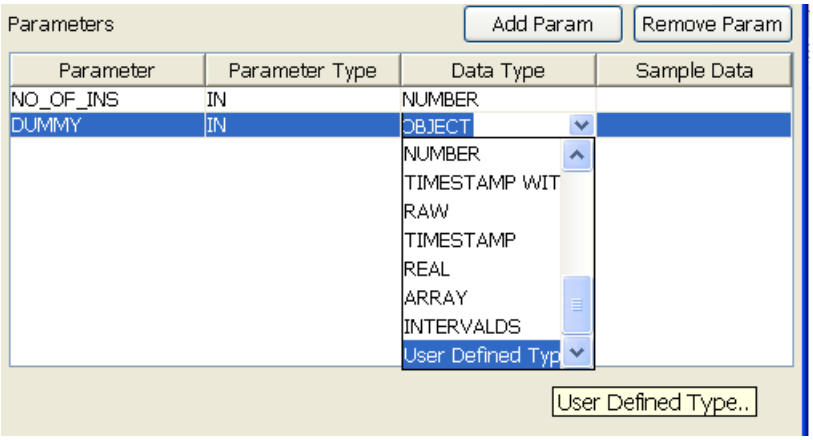


Figure 10: Selecting User Defined Type

7. Select the appropriate data type from the dialog.
8. After selecting user defined data type, it will populate in the **Data Type** column for the respective parameter.

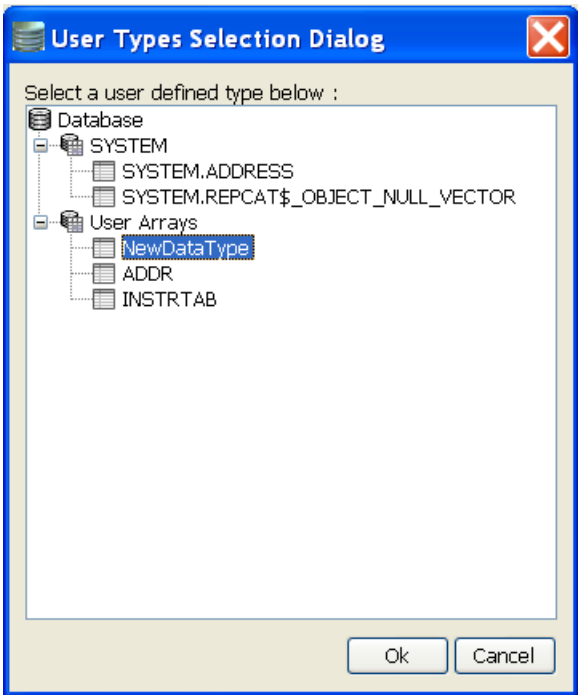


Figure 11: User Types Selection Dialog to select Data type

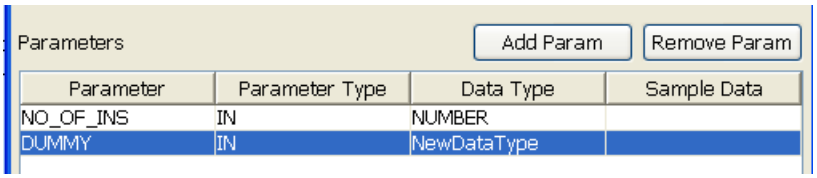


Figure 12: Populated User Defined Datatype

9. We need to generate the Input/Output parameters by clicking the **Generate Parameters** button. Then the input/output parameters will be generated as shown in below figure. After the generation of input/output parameters these parameters will be included in the input and output port xsd's.

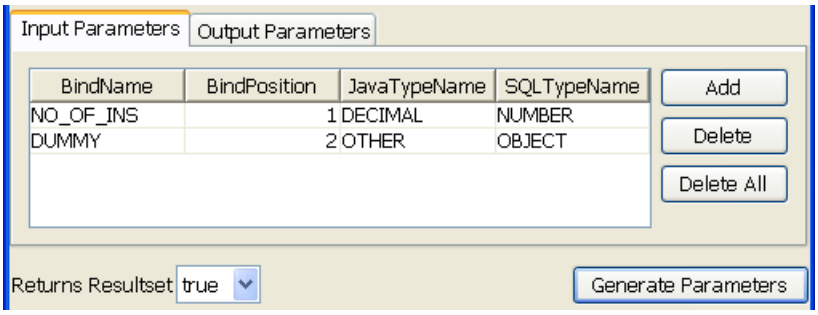


Figure 13: Generated Input/Output Parameters

Each of the columns is explained below.

Column Name	Description
BindName	This value is used to generate the schema for the query. In the above figure value for EMPLOYEE_ID (field name) is changed to EMPLOYEE_I. So the schema generated would contain EMPLOYEE_I as the first element instead of default populated value, EMPLOYEE_ID.
BindPosition	The position in the query where this value is bound to. Note: Do not change this value.
JavaTypeName	JDBC type which maps to Data Type.
SQlTypeName	This defines the data type of this column in the database table. This should be correctly defined.

10. Returns **Resultset** determines whether the configured stored procedure returns resultsets or not.
11. Click **Ok** to close the dialog.

4.3.2.2.1.3 Single Batch Mode

This option determines whether the component should send entire result of a query as a single message or as multiple messages.

- **yes -** Complete result of the query from input request is sent out as a single message. If the result set returned is huge then the component can run into memory problems and stop. When this value is selected, property **Batch Size** is hidden.
- **no -** Result of query from input is split and sent out as multiple messages. Number of rows from result to be included in each output message is determined by property **Batch Size**. When this value is selected, property **Batch Size** is shown.

Example: If a query returns 100 rows, and the batch size is set to 10, then 10 outputs will be generated each containing 10 rows.

4.3.2.2.1.4 Batch Size

This property is visible when the value of property **Single Batch Mode** is set as yes. The property determines the number of units of result an output message contains.

Each row in a result set (typically result of a select query) or an update result (result of update, delete, insert operations) is treated as unit of result.

Example: Consider a stored procedure that returns a result of select query followed by 3 update queries and another select query. Assume first select return 18 rows and second query returns 11 rows. If **Single Batch Mode** is set as no and **Batch Size** is set as 10 then there will be four output messages

- **first message:** first 10 rows from first query
- **second message:** remaining 8 rows from first query and 2 update query results
- **third message:** 3rd update query result and first 9 rows of second select query

- **fourth message:** remaining 2 rows from second query.

4.3.2.2.2 Using Named Configurations

Changes are done in eStudio w.r.t to Named Configurations. With the implementation of Named Configurations MCF panel is changed as shown in Figure below.

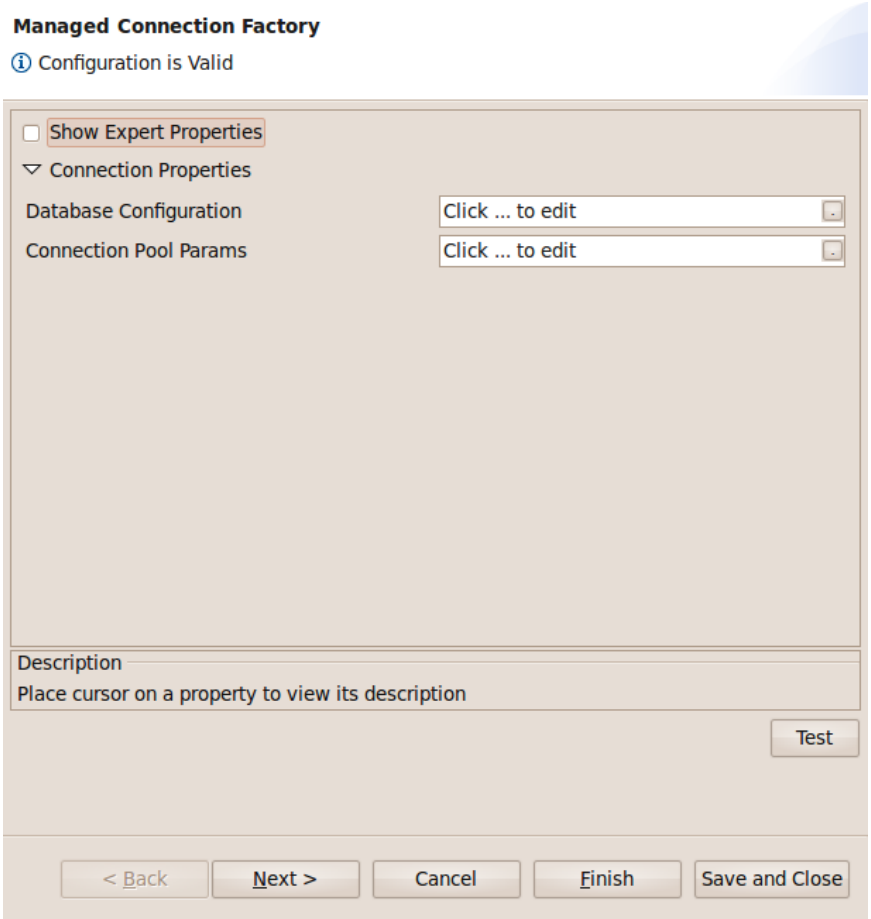


Figure: Modified Managed Connection Factory

Some properties are grouped as one property in the modified configuration panel. Database, Driver, URL, User name, Password, Connection ping sql and Connection Properties which were present previously in the MCF panel are grouped under the name Database Configuration.

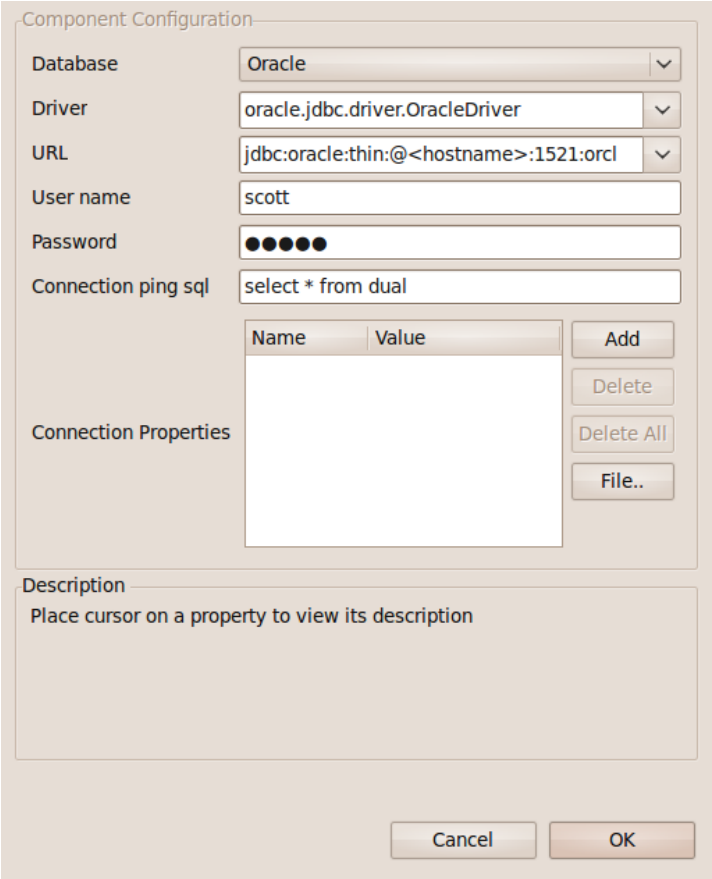


Figure: Database Configuration

4.3.2.3 Input Schema

The input schema is auto generated based on the configuration provided. When **Generate Parameters** button is clicked the input parameters required for the execution of the procedure will be added as child elements to the **CALL** element in the input schema as shown in figure 14.

XML Item	Value Type
<ns1:CALL>	
<ns1:param1>	#string

Figure 14: Input schema for Procedure with input parameter param1

The input XML to the component will thus be in the format shown in figure 15.

```
<ns1:CALL xmlns:ns1="http://www.fiorano.com/fesb/activity/DBProc1/In">
   <ns1:param1>param1</ns1:param1>
</ns1:CALL>
```

Figure 15: Sample XML corresponding to the Input Schema

When the property Use Connection details from input is chosen, an additional element ConnectionFactorySettings is added to the input schema, as shown in the figure. Properties that are used to create the connection are present under this element.

XML Item	Value Type
<ns1:CALL>	
<ns1:ConnectionFactorySettings>?	
<ns1:ConnectionProperties>?	
<ns1:Database>?	HSQL | IBM DB2 | Kingbase | Mckoi |
<ns1:Driver>?	#string
<ns1:JDBCURL>?	#string
<ns1:Username>?	#string
<ns1:Password>?	#string
<ns1:AdvancedConnectionProperties>?	#string
<ns1:AutoCommit>?	true | false
<ns1:AdvancedProperties>?	
<ns1:ConnectionPingSQL>?	#string
<ns1:EnableJDBCLogging>?	true | false

Figure 16: Input schema with ConnectionFactorySettings

4.3.2.4 Output Schema

This is auto generated based on the configuration provided. When **Generate Parameters** button is clicked the output parameters, if any, required for the execution of the procedure will be added as child elements to the **RESULT** element in the input schema as shown in figure 17.

<ns1:Result>	
<ns1:param2>	#string

Figure 17: Output schema when there are output parameters

<ns1:Result>	
<ns1:ResultSet>	
<ns1:Row>	
WC[##any]	

Figure 18: Output Schema when the Procedure returns result set

If the **Return Result Set** is set to true then an element **ResultSet** will be added and results appear as row elements in the output XSD as shown in figure 19.

When the procedure has both Result Set and parameters as output both elements will appear in output schema as shown in figure 19.

<ns1:Result>	
<ns1:param2>	#string
<ns1:ResultSet>	
<ns1:Row>	
WC[##any]	

Figure 19: Result Set and parameters as output elements in output schema

4.3.2.6 Functional Demonstration

4.3.2.6.1 Scenario 1

Execution of a Stored Procedure:

Start mckoiDB present at %FIORANO_HOME%\esb\samples\mckoiDB by executing CreateMckoiDB.bat and RunMckoi.bat. Configure the DBProc component as described in section SP Configuration. Use feeder and display components (shown in figure 20) to create a flow to send sample input and check the response respectively.

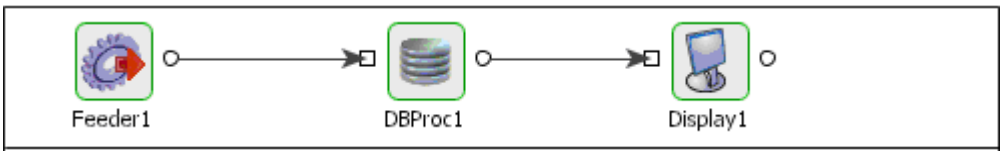


Figure 20: Configure the DB Proc component

```
<ns1:CALL xmlns:ns1="http://www.fiorano.com/fesb/activity/DBProc1/In">
   <ns1:I>10</ns1:I>
</ns1:CALL>
```

Figure 21: Demonstrating Scenario 1 with Sample Input

```
<?xml version="1.0" encoding="UTF-8"?>
<Result xmlns="http://www.fiorano.com/fesb/activity/DBProc1/Out">
   <O>20</O>
</Result>
```

Figure 22: Demonstrating Scenario 1 with Sample output

4.3.2.7 Useful Tips

- Only one stored procedure can be configured in the adapter. Please use the DB component if multiple stored procedures need to be configured for a single instance.
- It is recommended NOT to use JDBC-ODBC Bridge driver to connect to any RDBMS in your production environment. Please use a commercial JDBC driver instead.
- The JDBC drivers or the resources must be directly added onto the JDBC system lib and not as resource to the DBProc component itself. To add JDBC drivers to DBProc component, please refer to section 3.3.3.3.

4.3.3 DBQueryOnInput

The DBQueryOnInput component is used to execute different SQL statements with each request on a configured database. The SQL to be executed is not configured in the CPS and is taken from the input message. In other DB components, the component is configured with predefined SQL statement(s) in the CPS and some or all of these statements are executed for all requests.

Points to note:

- Only one query can be processed per message. If multiple queries have to be processed, use XMLSplitter to split the message into multiple messages each containing a single query.
- It is recommended NOT to use JDBC-ODBC Bridge driver to connect to any RDBMS in a production environment. Please, use a commercial JDBC driver instead.
- The JDBC drivers or the resources must be directly added onto the JDBC system lib and not as resource to the DBQueryOnInput component itself. To add JDBC drivers to DBQueryOnInput component, please refer to section 3.3.3.3.

4.3.3.1 Managed Connection Factory Panel

The connection details are configured in the first panel, **Managed Connection Factory** (**MCF**). Figure 1 illustrates the panel with expert properties view enabled.

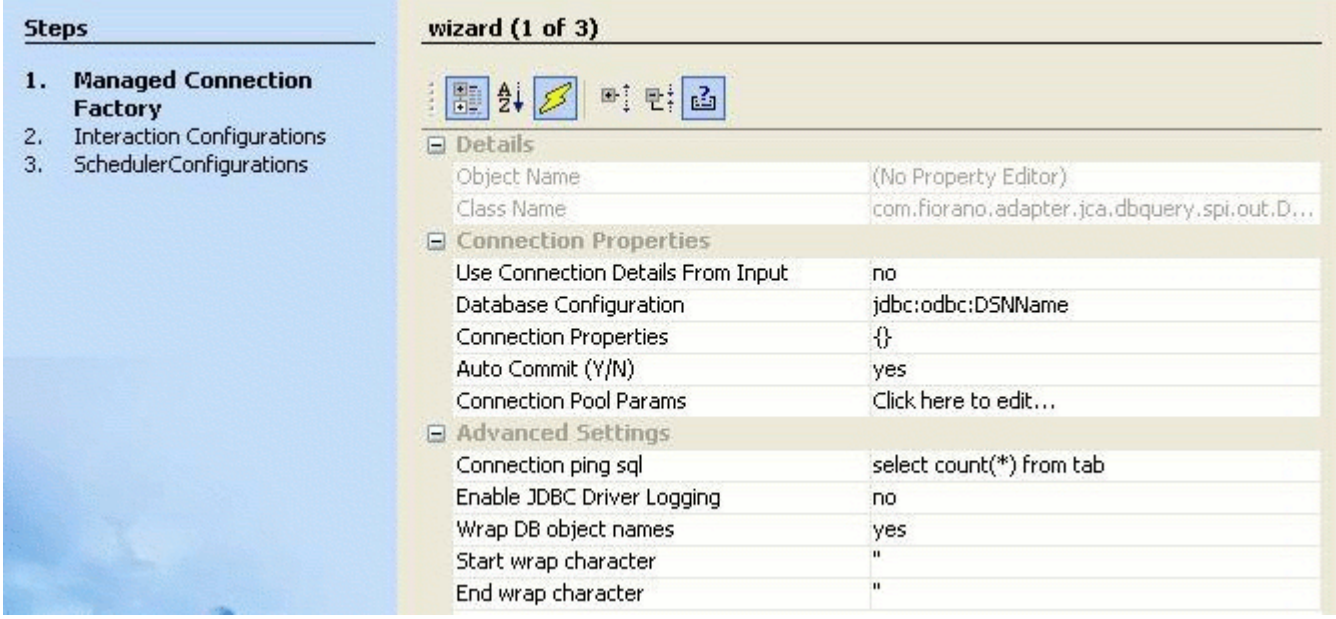


Figure 1: Connection configuration details in MCF panel

4.3.3.1.1 Connection Properties

4.3.3.1.1.1 Use Connection Details From Input

Parameters to create the connection can be specified in the input message when this property is set to true. If this property is selected the validation errors in the managed connection factory panel of the CPS are treated as warnings. So user can bypass this step without giving valid configuration and complete the configuration of the component. If valid properties are not provided even in the input message exception will be thrown at runtime.

4.3.3.1.1.2 Database Configuration

Click ellipsis button to launch the **Database Configuration** editor as shown in Figure 2, where details of the database to which the component should connect are configured.

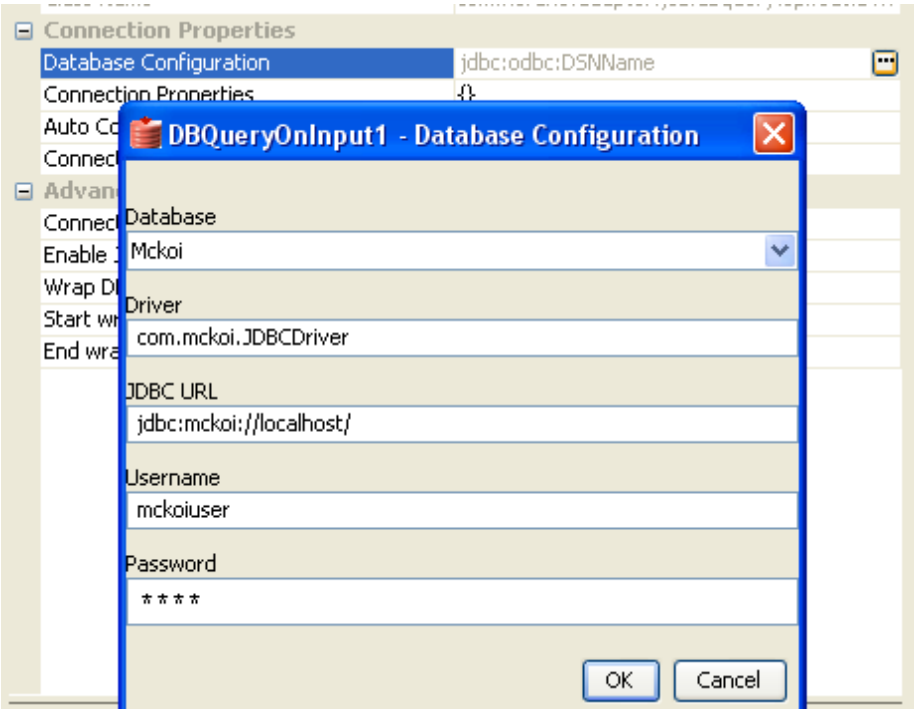


Figure 2: Database configuration editor with mckoi database details

- **Database**

  This property determines the vendor of the database to which the component has to connect.

  1. The vendor name is marked in red color if the default JDBC driver class is not present in component's class path.
  2. Even if a particular database vendor name is not present in the drop-down list, the component can still connect to the database.
  3. If a vendor name is specified in the drop-down list, it only means that vendor specific handling is done. **Example:** vendor specific handling for data types, naming conventions etc
  4. To connect to a database from a vendor whose name is not specified in the drop-down list, select Other and provide the correct values for **Driver** and **JDBC URL**.

- **Driver**

  The driver class name that should be used to connect to the database. On selecting required value for **Database**, driver values are populated with standard value (This can be changed to required value based on driver being used).

  Note: The jar file(s) that are part of JDBC client libraries for selected vendor have to be added as resources to **JDBC** system library.

- **JDBC URL**

  This property determines the location at which the required database is running. On selecting required database, **URL** value is populated with standard value (This can be changed to required values based on driver being used).

  Note: The populated values have place holders which have to be replaced to point to correct database location, Example: In Figure 2 **<hostname>** is replaced with localhost IP indicating that the database is running on local machine.

- **Username**

  This property determines the user name that should be used to connect to the database.

- **Password**

  This property determines the password for the specified user.

4.3.3.1.1.3 Connection Properties

Any driver specific connection properties which may have to be passed while creating a JDBC connection should be provided against Connection Properties (shown in Figure 3). For example, fixedString=true uses FIXED CHAR semantics for string values in oracle.

Note:

- Please refer to documentation of driver that is being used for valid name-values for connection properties.
- Connection properties can be loaded from a properties file using **File...** button. Refer to the following link http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html#load(java.io.InputStream) for details on properties file.

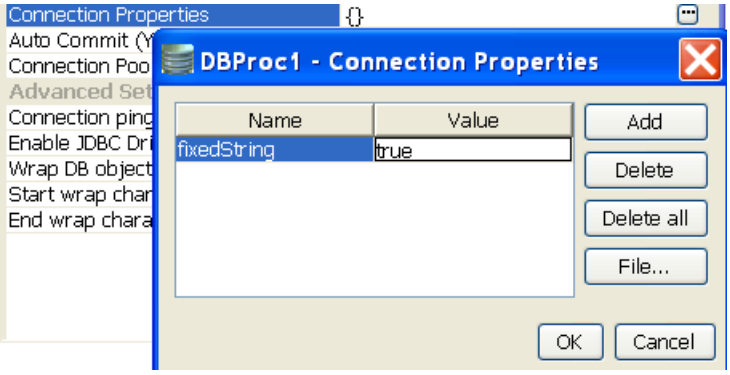


Figure 3: Connection property for oracle

4.3.3.1.1.4 Auto Commit (Y/N)

Commit mode that should be used by the JDBC connection.

- **yes** - Any transactions (queries executed) are automatically and implicitly committed to the database. This is done even before the response is generated.
- **no** - Any transactions (queries executed) are committed after the request is processed successfully and response is generated, but before the message is sent out of the component.

4.3.3.1.2 Advanced Settings

4.3.3.1.2.1 Connection ping sql

A SQL statement which is guaranteed to execute without exception, except when connection to database is lost. when a SQL exception occurs on executing a configured query, this SQL statement is executed. If execution of this SQL statement fails as well, then it is assumed that connection to database is lost and appropriate configured action (say, reconnect) is taken.

Example: select * from dual for oracle, select 1 for MS SQL

4.3.3.1.2.2 Enable JDBC Driver Logging

Value yes implies that logging at the driver level should be enabled. This is used as a debugging option.

4.3.3.1.2.3 Wrap DB object names

When database object names (viz. table names, column names, schema names…) contain spaces, then they should be wrapped in database dependent special characters. For example, " " for Oracle, [ ] for Microsoft Excel, and no wrap characters for MySQL.

Database object names are wrapped as shown below -

Start wrap character + object name + End wrap character

Note: Providing a wrong wrap character may lead to problems.

4.3.3.1.2.4 Start wrap character

Character which should be used before the object name.

4.3.3.1.2.5 End wrap character

Character which should be used after the object name.

4.3.3.2 Interaction Configurations Panel

Business logic configuration details are configured in the second panel, **Interaction Configurations** panel. Figure 4 illustrates the panel with expert properties view enabled.

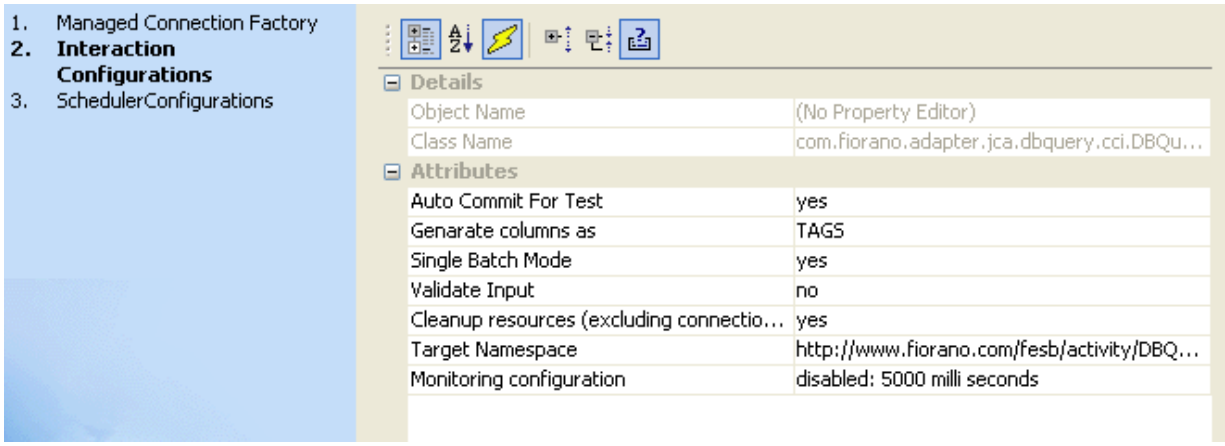


Figure 4: Business logic configuration in Interaction configurations panel

4.3.3.2.1 Attributes

4.3.3.2.1.1 Auto Commit For Test

This property determines whether auto-commit should be enabled when testing from the CPS.

- **yes** - Any transactions (queries executed) are automatically committed to the database while testing. Performed transactions will have to manually undone.
- **no** - Any transactions (queries executed) are rolled back at the completion of test.

This property will override the value provided for property **Auto Commit (Y/N)** in the **MCF** panel.

4.3.3.2.1.2 Generate columns as

This property determines how any data returned (result set) by the component is represented in the output message. Result sets are tabular data returned for a database query. Select queries always return result sets and stored procedures may return result sets based on the type of variables returned by stored procedures.

- **TAGS** - Column names of result set will be generated as XML elements in the output message (shown in Figure 5). The schema on the output port is not completely defined (shown in Figure 6) in this case as the schema varies depending on the query sent in input.

```
<?xml version="1.0" encoding="UTF-8"?>
<Result xmlns="http://www.fiorano.com/fesb/activity/DBQueryOnInput1/Out">
   <query>select * from mckoiuser.POsNY</query>
   <row resultcount="1">
      <InternalPO>12456</InternalPO>
      <ArrivalTime>1/1/2004</ArrivalTime>
   </row>
   <row resultcount="1">
      <InternalPO>12457</InternalPO>
      <ArrivalTime>1/1/2004</ArrivalTime>
```

Figure 5: Output when columns (InternalPO, ArrivalTime) are generated as TAGS

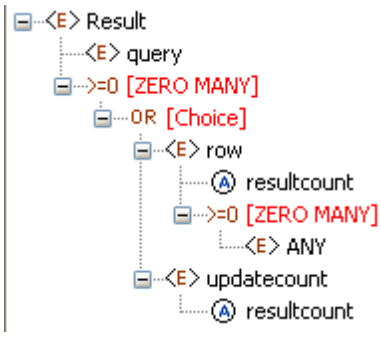


Figure 6: Output schema when the columns are generates as TAGS

However, since the column names are generated as elements, if the columns in result are known and same for all inputs of business scenario, a schema for the output can be generated by defining rowType element type manually (shown in Figure 7) and loaded into Fiorano Mapper.

```
   <xsd:complexType name="rowtype">
      <xsd:sequence>
<!--          <xsd:any namespace="##targetNamespace"
processContents="lax" minOccurs="0" maxOccurs="unbounded"/> -->
      <xsd:element name="InternalPO" type="xsd:string"/>
      <xsd:element name="ArrivalTime" type="xsd:string"/>
      </xsd:sequence>
      <xsd:attribute name="resultcount" type="xsd:positiveInteger"
      use="required"/>
   </xsd:complexType>
```

Figure 7: Modified rowType element for query returning InternalPO and ArrivalTime columns

This option allows direct mappings in XSLT (shown in Figure 8)

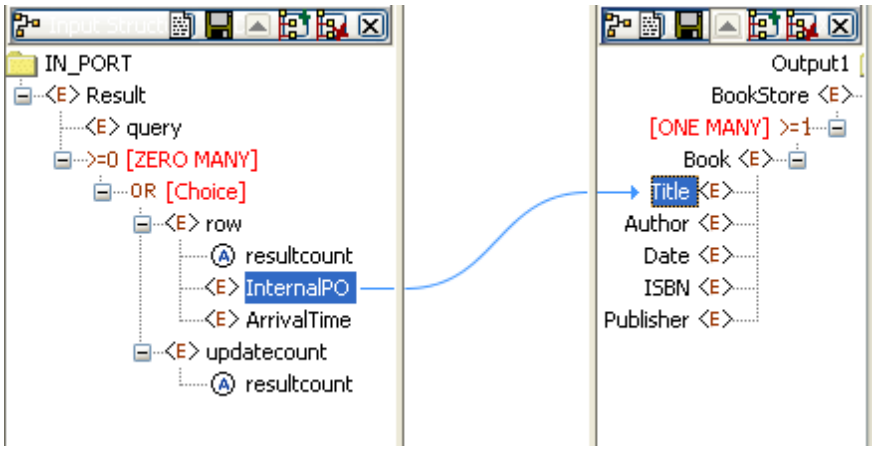


Figure 8: Mapping to set value from InternalPO to Title

- **ATTRIBUTES** - Column names of result set will be generated as value of attribute name of element column in the output message (shown in Figure 9). The output structure (shown in Figure 10) is completely defined and will not vary based on the input.

```
<?xml version="1.0" encoding="UTF-8"?>
<Result xmlns="http://www.fiorano.com/fesb/activity/DBQueryOnInput1/Out">
   <query>select * from mckoiuser.POsNY</query>
   <row resultcount="1">
      <column name="InternalPO" type="VARCHAR">12456</column>
      <column name="ArrivalTime" type="VARCHAR">1/1/2004</column>
   </row>
   <row resultcount="1">
      <column name="InternalPO" type="VARCHAR">12457</column>
      <column name="ArrivalTime" type="VARCHAR">1/1/2004</column>
   </row>
```

Figure 9: Output when columns names are generated as ATTRIBUTES

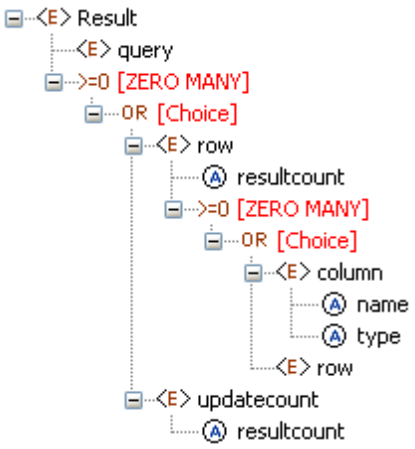


Figure 10: Output structure when the column names are generated as ATTRIBUTES

Defining mappings in XSLT using this option will require writing some user defined XSL to loop through all columns in a row and map only data from column element whose attribute name contains required column name to required output element.

4.3.3.2.1.3 Single Batch Mode

This option determines whether the component should send entire result of a query as a single message or as multiple messages.

- **yes** - Complete result of the query from input request is sent out as a single message. If the result set returned is huge then the component can run into memory problems and stop. When this value is selected, property **Batch Size** is hidden.
- **no** - Result of query from input is split and sent out as multiple messages. Number of rows from result to be included in each output message is determined by property **Batch Size**. When this value is selected, property **Batch Size** is shown.

Example: If a query returns 100 rows, and the batch size is set to 10 then 10 outputs will be generated each contains 10 rows.

4.3.3.2.1.4 Batch Size

This property is visible when the value of property **Single Batch Mode** is set as yes. The property determines the number of units of result an output message contains.

Each row in a result set (typically result of a select query) or an update result (result of update, delete, insert operations) is treated as unit of result.

Example: Consider a stored procedure that returns a result of select query followed by three update queries and another select query. Assume first select return 18 rows and second query returns 11 rows. If **Single Batch Mode** is set as no and **Batch Size** is set as 10 then there will be four output messages.

- **first message:** first 10 rows from first query
- **second message:** remaining 8 rows from first query and 2nd update query results.
- **third message:** 3rd update query result and first 9 rows of second select query.
- **fourth message:** remaining two rows from second query.

4.3.3.3 Input and output

4.3.3.3.1 Input

Input schema for the component is shown in Figure 14

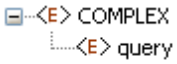


Figure 11: Input schema for the component

Input message (shown in Figure 15) contains only one element query whose value contains the query that has to be executed.

```
<ns1:COMPLEX xmlns:ns1="http://www.fiorano.com/fesb/activity/DBQueryOnInput1/In">
   <ns1:query>select * from mckoiuser.POsNY</ns1:query>
</ns1:COMPLEX>
```

Figure 12: Sample input message

When the property Use Connection details from input is chosen, an additional element ConnectionFactorySettings is added to the input schema, as shown in the figure. Properties that are used to create the connection are present under this element.

XML Item	Value
<ns1:COMPLEX>	
<ns1:ConnectionFactorySettings>?	
<ns1:ConnectionProperties>?	
<ns1:Database>?	HSQL | IBM DB2 | Kingbase |
<ns1:Driver>?	#string
<ns1:JDBCURL>?	#string
<ns1:Username>?	#string
<ns1:Password>?	#string
<ns1:AdvancedConnectionProperties>?	#string
<ns1:AutoCommit>?	true | false
<ns1:AdvancedProperties>?	
<ns1:ConnectionPingSQL>?	#string
<ns1:EnableJDBCLogging>?	true | false
<ns1:query>	#string

Figure 13: Input schema with ConnectionFactorySettings

4.3.3.3.2 Output

Output schema for the component depends on the value configured for property **Generate columns** as in **Interaction Configurations panel**. Output schemas based on the property value are shown in Figures 6 and 10 and sample inputs for them are shown in Figures 5 and 9.

4.3.3.4 Functional Demonstration

4.3.3.4.1 Scenario 1

Execution of a select query with ATTRIBUTES as the mode of column generation.

Start mckoiDB present in %FIORANO_HOME%\esb\samples\mckoiDB by executing CreateMckoiDB.bat and RunMckoi.bat **files**. Configure the DBQueryOnInput component as shown in Figures 14 and 15. Use feeder and display components (shown in Figure 16) to create a flow to send sample input and check the response respectively.

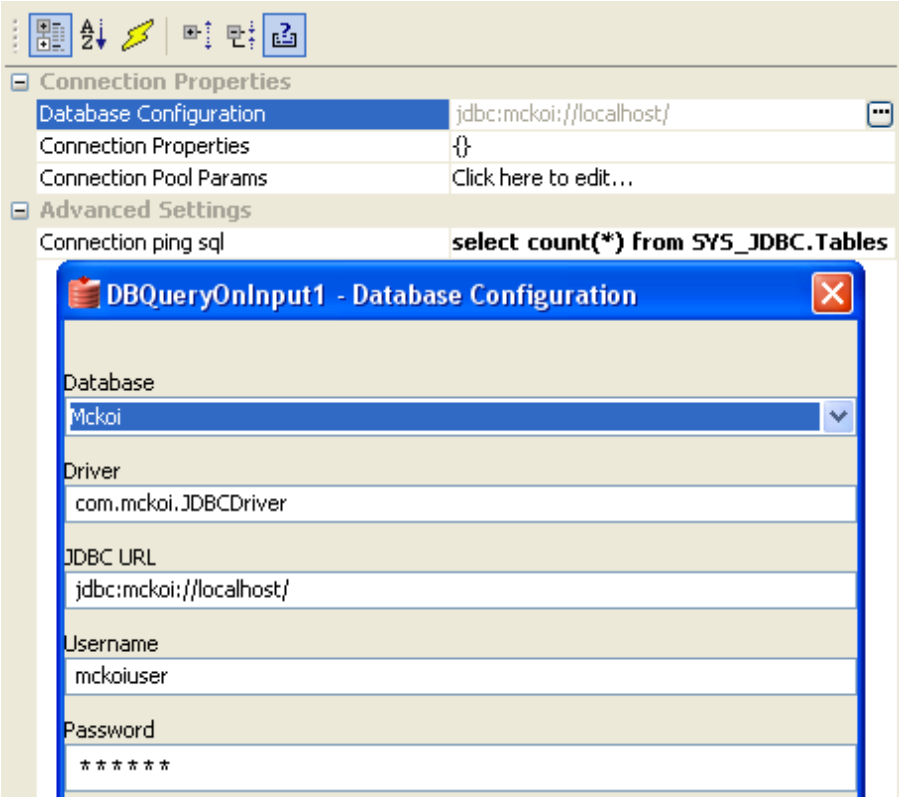


Figure 14: MCF Panel configuration for scenario1

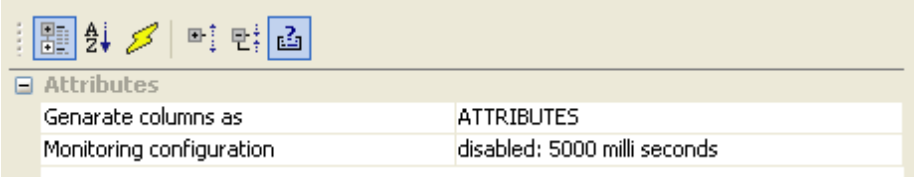


Figure 15: Interactions configuration of scenario1

Use feeder and display components (shown in Figure 16) to create a flow to send sample input and check the response respectively.

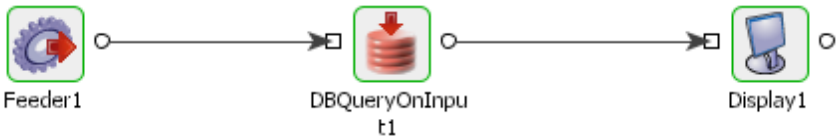


Figure 16: Flow for scenario1

Send input message, shown in Figure 17, from feeder and notice the output similar to the one shown in Figure 18 in display.

```
<ns1:COMPLEX xmlns:ns1="http://www.fiorano.com/fesb/activity/DBQueryOnInput1/In">
   <ns1:query>select * from mckoiuser.LineItems</ns1:query>
</ns1:COMPLEX>
```

Figure 17: Input for scenario1

```
<?xml version="1.0" encoding="UTF-8"?>
<Result xmlns="http://www.fiorano.com/fesb/activity/DBQueryOnInput1/Out">
   <query>select * from mckoiuser.LineItems</query>
   <row resultcount="1">
      <column name="OrderID" type="VARCHAR">1001</column>
      <column name="CustomerID" type="VARCHAR">4001</column>
      <column name="Email" type="VARCHAR">customer1@abc.com</column>
      <column name="Mode" type="VARCHAR">Air</column>
      <column name="Address" type="VARCHAR">CA</column>
      <column name="Country" type="VARCHAR">USA</column>
      <column name="PostalCode" type="VARCHAR">95032</column>
      <column name="PhoneNumber" type="VARCHAR">n/a</column>
      <column name="Product" type="VARCHAR">CISCO ROUTER</column>
      <column name="Price" type="VARCHAR">$450</column>
      <column name="BillAmount" type="VARCHAR">$4500</column>
      <column name="OrderDate" type="VARCHAR">1/1/2004</column>
   </row>
```

Figure 18: Output for scenario1

4.3.3.4.2 Scenario 2

Use same connection configurations as described in **scenario 1** and change the Interactions Configuration as shown in Figure 19.

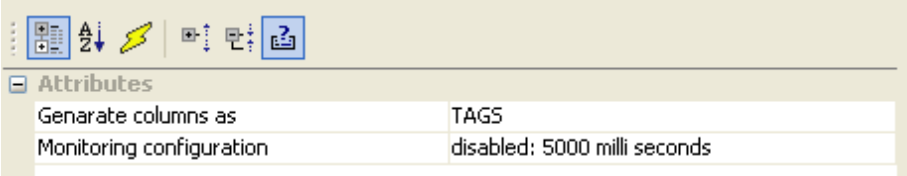


Figure 19: Interaction configuration of scenario2

Repeat the test as described in **scenario 1** with same input and observe the output similar to the one shown in Figure 20.

```
<?xml version="1.0" encoding="UTF-8"?>
<Result xmlns="http://www.fiorano.com/fesb/activity/DBQueryOnInput1/Out">
   <query>select * from mckoiuser.LineItems</query>
   <row resultcount="1">
      <OrderID>1001</OrderID>
      <CustomerID>4001</CustomerID>
      <Email>customer1@abc.com</Email>
      <Mode>Air</Mode>
      <Address>CA</Address>
      <Country>USA</Country>
      <PostalCode>95032</PostalCode>
      <PhoneNumber>n/a</PhoneNumber>
      <Product>CISCO ROUTER</Product>
      <Price>$450</Price>
      <BillAmount>$4500</BillAmount>
      <OrderDate>1/1/2004</OrderDate>
   </row>
```

Figure 20: Output of scenario2

4.3.3.5 Useful Tips

- Only one query can be processed per message. If multiple queries have to be processed, use XMLSplitter to split the message into multiple messages each containing a single query.
- It is recommended NOT to use JDBC-ODBC Bridge driver to connect to any RDBMS in a production environment. Please, use a commercial JDBC driver instead.
- The JDBC drivers or the resources must be directly added onto the JDBC system lib and not as resource to the DBQueryOnInput component itself. To add JDBC drivers to DBQueryOnInput component, please refer to section 3.3.3.3.

4.3.4 DBQuery

The DBQuery component is used to configure simple queries to insert, update, delete or select records from database. The CPS of this component allows designing queries with the application of zero coding effort. However, the SQL mode can also be used to write queries. Syntactical validity can be verified by using the Check Syntax SQL button provided on the SQL configuration panel in the SQL mode.

Points to note:

- Only one query can be configured in the adapter. Please use the DB component if multiple query configurations need to be configured for a single instance.
- Only one query can be processed per message. If multiple queries have to be processed, use XMLSplitter to split the message into multiple messages each containing a single query.
- It is recommended NOT to use JDBC-ODBC Bridge driver to connect to any RDBMS in your production environment. Please use a commercial JDBC driver instead.

- The JDBC drivers or the resources must be directly added onto the JDBC system lib and not as resource to the DBQuery component itself. To add JDBC drivers to DBQuery component, please refer to section 3.3.3.3.

4.3.4.1 Managed Connection Factory Panel

The connection details are configured in the first panel, **Managed Connection Factory** (**MCF**). Figure 1 illustrates the panel with expert properties view enabled.

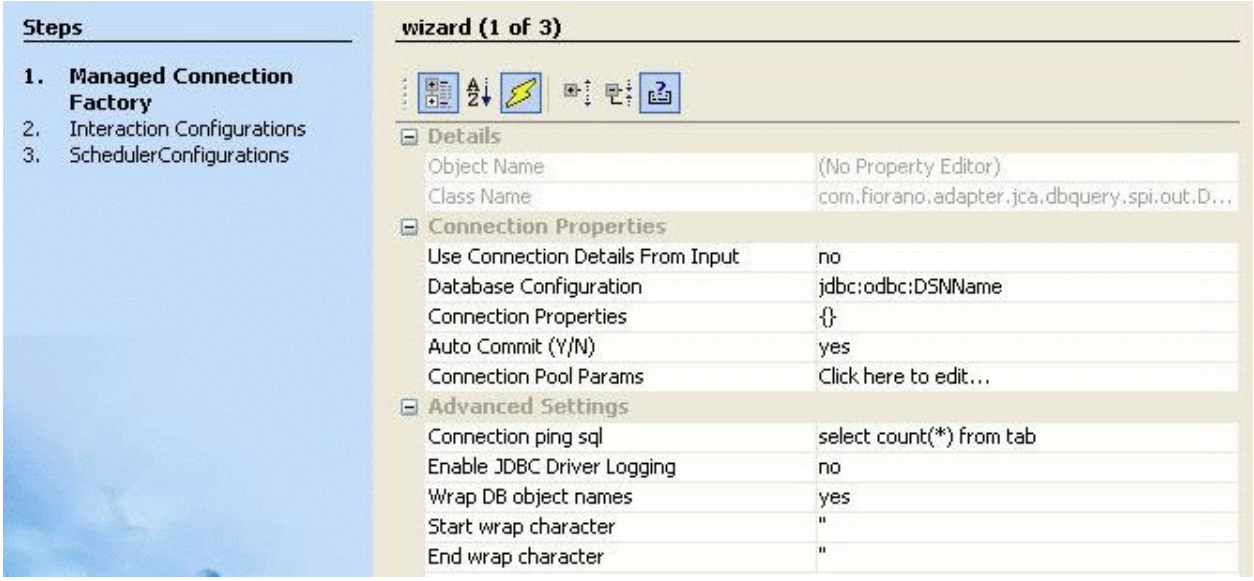


Figure 1: Connection configuration details in MCF panel

4.3.4.1.1 Connection Properties

4.3.4.1.1.1 Use Connection Details From Input

Parameters to create the connection can be specified in the input message when this property is set to true. If this property is selected the validation errors in the managed connection factory panel of the CPS are treated as warnings. So user can bypass this step without giving valid configuration and complete the configuration of the component. If valid properties are not provided even in the input message exception will be thrown at runtime.

4.3.4.1.1.2 Database Configuration

Click ellipsis button to launch **Database Configuration** editor, shown in Figure 2, where details of the database to which the component should connect are configured.

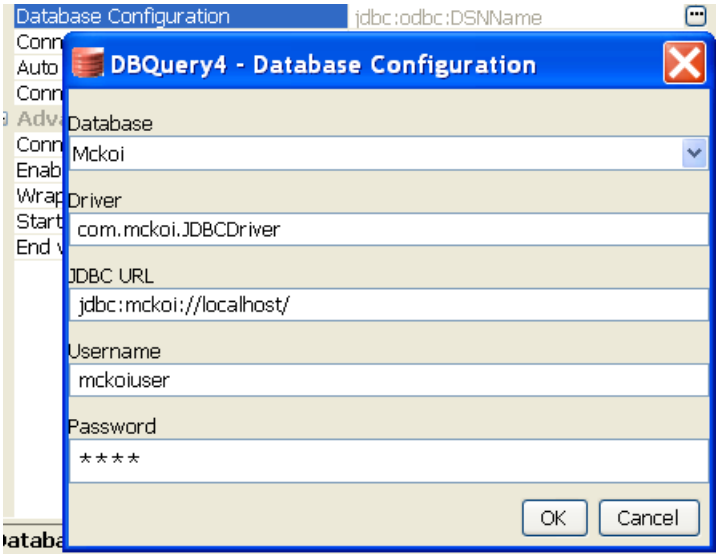


Figure 2: Database Configuration editor with mckoi database details

- **Database**

  This property determines the vendor of the database to which the component has to connect.

  - Vendor name is marked in red color if the default JDBC driver class is not present in component's class path.
  - Even if a particular database vendor name is not present in the drop-down list the component can still connect to the database.
  - If a vendor name is specified in the drop-down list, it only means that vendor specific handling is done. **Example**: vendor specific handling for data types, naming conventions etc
  - To connect to a database from a vendor whose name is not specified in the drop-down list, select Other and provide the correct values for **Driver** and **JDBC URL**.

- **Driver**

  The Driver class name that should be used to connect to the database. On selecting required value for **Database**, Driver value will be populated with standard value (This can be changed to required value based on driver being used).

  Note: The jar file(s) that are part of JDBC client libraries for selected vendor have to be added as resources to **JDBC** system library.

- **JDBC URL**

  This property determines the location at which the required database is running. On selecting required database, **URL** value is populated with standard value (This can be changed to required values based on driver being used).

  Note: The populated value will have place holders which have to be replaced to point to correct database location, for example, In Figure 2 **<hostname>** is replaced with localhost IP, indicating that the database is running on local machine.

- **Username**

  This property determines the username that should be used to connect to the database.

- **Password**

  This property determines the password for the specified user.

4.3.4.1.1.3 Connection Properties

Any driver specific connection properties which may have to be passed while creating a JDBC connection should be provided against Connection Properties (shown in Figure 3). For example, fixedString=true uses FIXED CHAR semantics for string values in oracle.

Note:

- Please refer to documentation of driver that is being used for valid name-values for connection properties.
- Connection properties can be loaded from a properties file using **File...** button. Refer to http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html#load(java.io.InputStream for details on properties file.

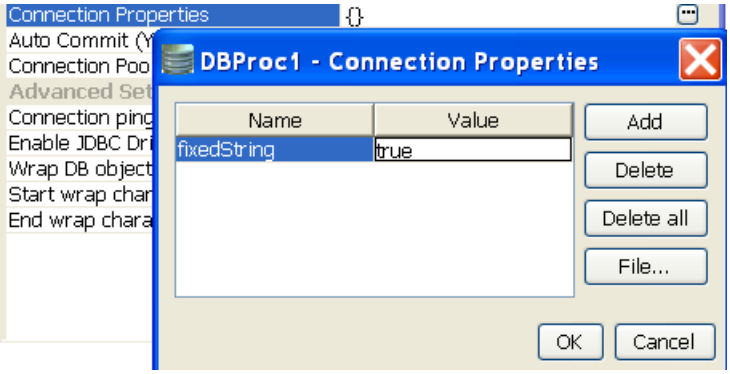


Figure 3: Connection property for oracle

4.3.4.1.1.4 Auto Commit (Y/N)

Commit mode that should be used by the JDBC connection.

- **yes** - Any transactions (queries executed) will be automatically and implicitly committed to the database. This is done even before the response is generated.

- **no** - Any transactions (queries executed) will be committed after the request is processed successfully and response is generated, but before the message is sent out of the component.

4.3.4.1.2 Advanced Settings

4.3.4.1.2.1 Connection ping sql

A SQL statement which is guaranteed to execute without exception, except when connection to database is lost. When a SQL exception occurs on executing a configured query, this SQL statement is executed. If execution of this SQL statement fails as well, then it is assumed that connection to database is lost and appropriate configured action (say, reconnect) is taken.

Example: select * from dual for oracle, select 1 for MS SQL.

4.3.4.1.2.2 Enable JDBC Driver Logging

Value yes implies that logging at the driver level should be enabled. This is used as a debugging option.

4.3.4.1.2.3 Wrap DB object names

When database object names (viz. table names, column names, schema names…) contain spaces, they should be wrapped in database dependent special characters. For example, “ ” for Oracle, [ ] for Microsoft Excel and no wrap characters for MySQL.

Database object names are wrapped as shown below:

Start wrap character + object name + End wrap character

Note: Providing a wrong wrap character may lead to problems

4.3.4.1.2.4 Start wrap character

Character which should be used before the object name.

4.3.4.1.2.5 End wrap character

Character which should be used after the object name.

4.3.4.2 Interaction Configurations Panel

Business logic configuration details are configured in the second panel, **Interaction Configurations**. Figure 4 illustrates the panel with expert properties view enabled.

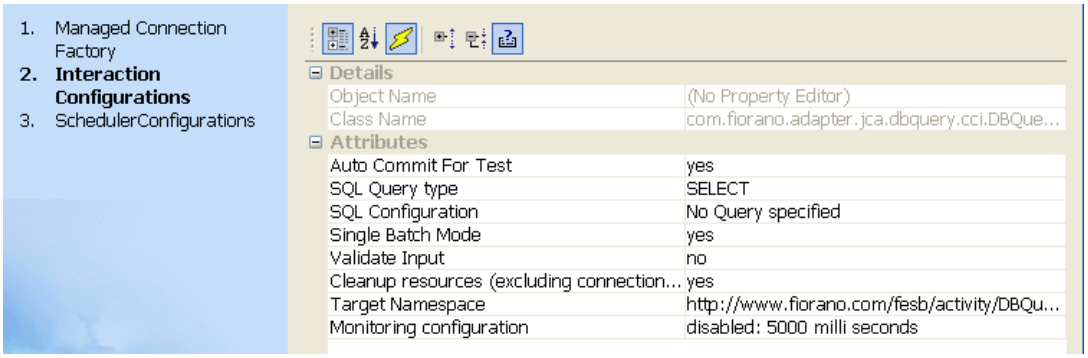


Figure 4: Business in Interaction panel logic configuration

4.3.4.2.1 Attributes

4.3.4.2.1.1 Auto Commit for Test

This property determines whether auto-commit should be enabled when testing from the CPS. Business logic configuration in Interaction configurations panel.

- **yes -** Any transactions (queries executed) will be automatically committed to the database while testing. Performed transactions will have to manually undone
- **no -** Any transactions (queries executed) will be rolled back at the completion of test

This property will override the value provided for property **Auto Commit (Y/N)** in the **MCF** panel.

4.3.4.2.1.2 SQL Query type

This property determines SQL query type to be executed. The user can select one of the SQL query types from SELCET, UPDATE, INSERT and DELETE.

Explanation for types of queries is given in the following table:

Type of query	Explanation
INSERT	Inserts/adds data into database table
UPDATE	Modifies existing data in database table
DELETE	Deletes data from database table
SELECT	Retrieves data from database table

4.3.4.2.1.3 SQL Configuration

Click ellipsis button against property SQL configuration to launch the SQL Configuration wizard which allows configuring queries that have to be executed.

- **Table Selection Dialog**

  Configuring queries requires selecting database object Tables on which actions have to be taken. The Table Selection Dialog does the required thing. This panel is shown in Figure 5.

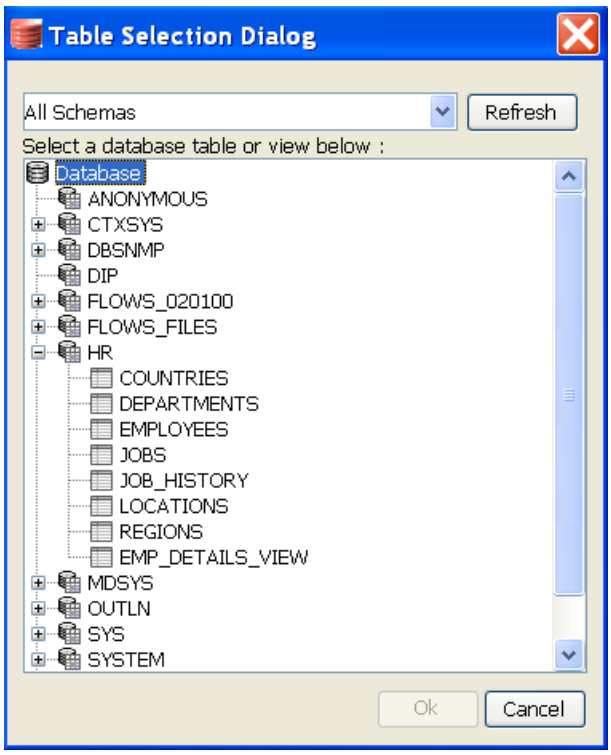


Figure 5: Table Selection Dialog dialog box

Select the Table on which the query should execute. Filter the tables by selecting the required schema and clicking Refresh button.

- **Insert Statement Configuration**

  Click ellipsis button next to **SQL configuration** property after selecting **SQL Query Type** as INSERT will launch the SQL Configuration Wizard which will useful to configure Insert query.

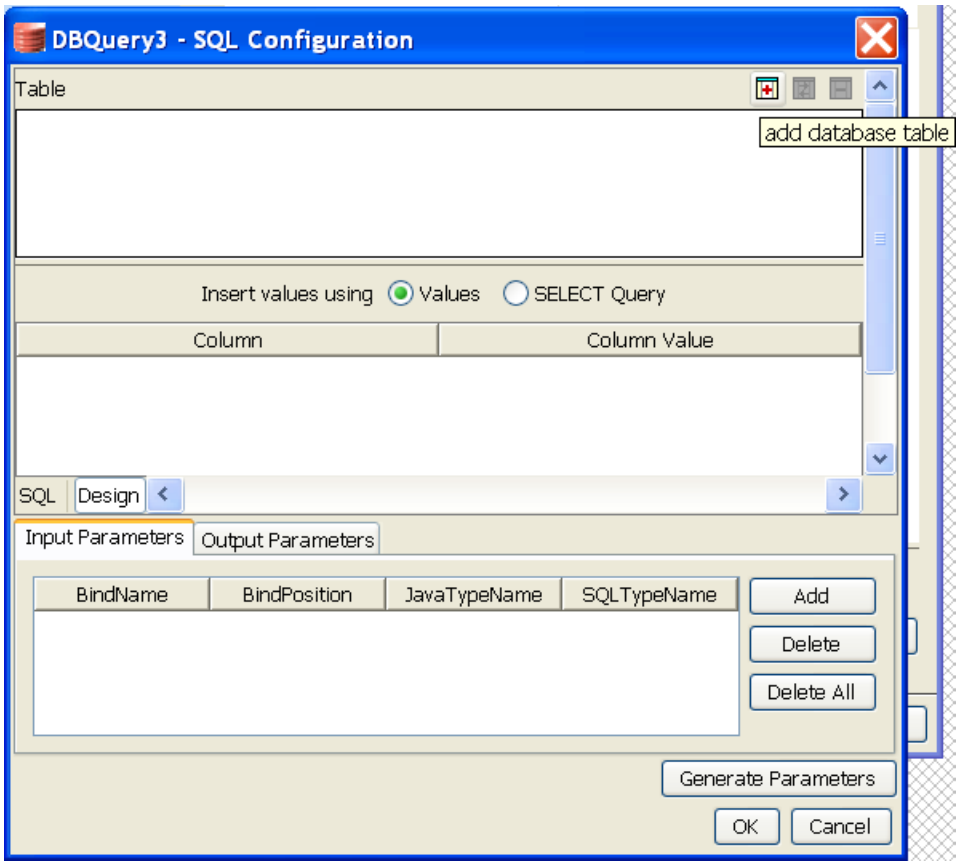


Figure 6: SQL Configuration Dialog for Insert Query

- **Simple Insert Statement**

  Inserts a row in configured table with column values taken from input XML or with constant column values.

  1. Click add database table button to launch Table Selection Dialog dialog box.
  2. Select required table as explained in Table Selection Dialog section. Selected table is added to the easel under Table. Primary key column, if exists, is marked with adjacent to column name.

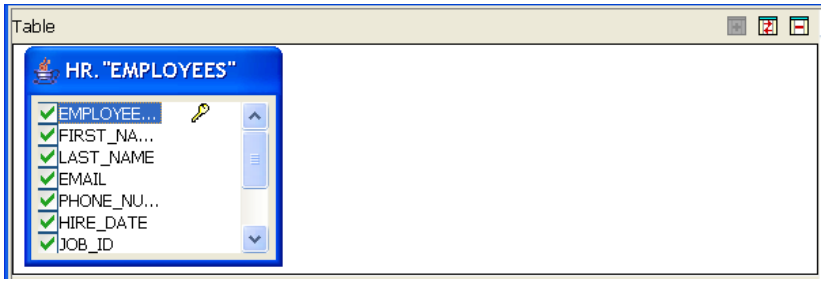


Figure 7: Selected table added to easel

3. Table can be changed by clicking **replace selected table** button and removed by clicking **remove database table** button.
4. If values are never to be inserted into a particular column, then that column can be unchecked (this requires column has a default value or supports null values) as shown in Figure 8.

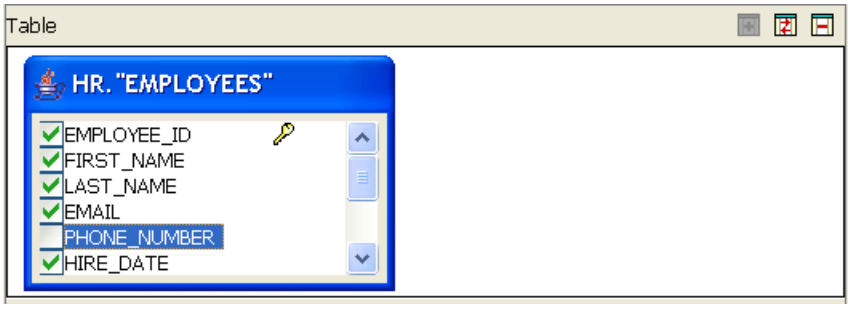


Figure 8: Ignoring column for insertion

5. To insert a constant value for a particular column, specify the required value in the **Column Value** column against the required column name.

   Note:

   - If the value is a string value it should be wrapped in single quotes (' ')
   - ? indicates value will be taken from input or from the output of another query where possible

6. Insert statement is automatically generated and shown in the text editor under SQL Statement in SQL tab. The generated SQL can be validated by pressing (**check syntax**) button.

Note: This feature only checks for invalid tokens, it does not perform a complete syntax check.

- **Enable Reconfiguration**

  Select the check box against this option to reconfigure the query, if it is not selected then the **Design** tab (which is used to configure the query) will not be visible.

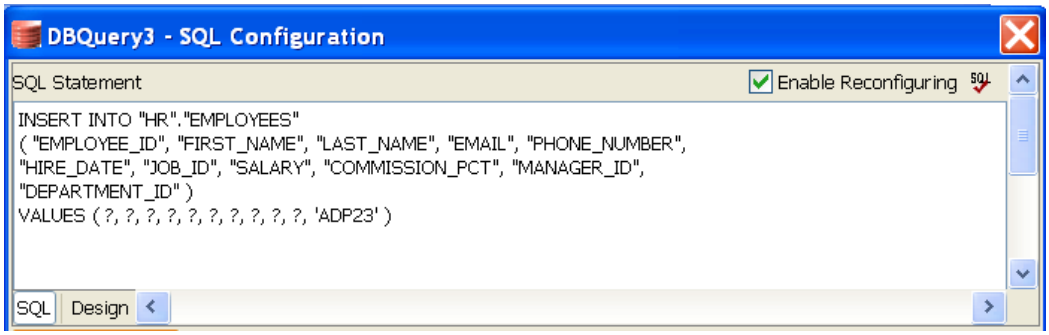


Figure 9: Generated insert query

7. Once the query is configured, the user has to generate the input and output parameters by clicking the **Generate Parameters** button.
8. If the query contains any value which should be taken from input XML then the generated parameters are used to define the input and output port schemas.
9. The **Add** and **Delete** buttons are used to add and delete particular parameters from the list. **Delete All** button used to delete all the parameters from the list.

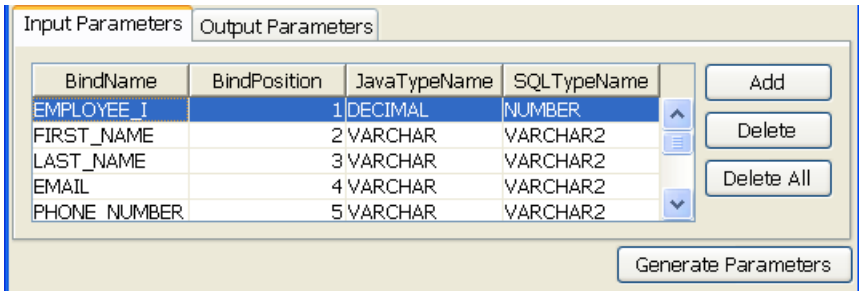


Figure 10: Generated Input/Output Parameters

Each of the columns is explained below.

Column Name	Description
BindName	This value is used to generate the schema for the query. In the Figure 10 value for EMPLOYEE_ID (field name) is changed to EMPLOYEE_I. So the schema generated would contain EMPLOYEE_I as the first element instead of default populated value, EMPLOYEE_ID.
Bind Position	The position in the query where this value is bound to. Note: Do not change this value.
Java TypeName	JDBC type which maps to Data Type.
SQL TypeName	This defines the data type of this column in the database table. This should be correctly defined.

10. Click **Ok** to close the dialog.

- **Insert Statement with select**

Insert rows in configured table by selecting rows from another table.

1. Follow the steps from 1 to 4 as described in section Simple Insert Statement.

2. Select **SELECT Query** option against **Insert values using** as shown in Figure 11.

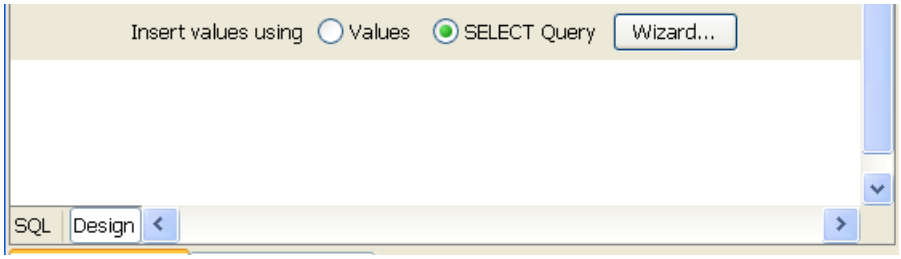


Figure 11: Option to insert values into a table using select query

3. Click **Wizard...** button to launch **Query Builder** to specify the Select query.
4. Follow the steps as described in section Select Statement Configuration.
5. The select query is automatically generated and shown in the text editor under **Insert values using** in **Design** tab as shown in Figure 12.

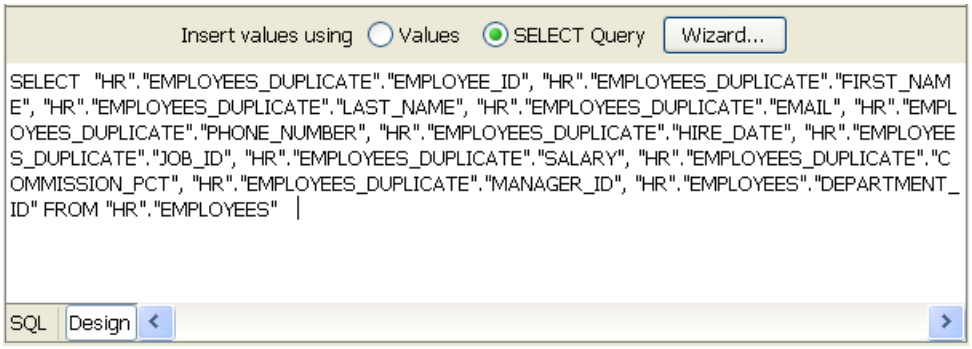


Figure 12: Generated Select query which is used in Insert Query

Insert statement is automatically generated and shown in the text editor under SQL Statement in SQL tab. The generated SQL can be validated by pressing (**check syntax**) button.

Note: This feature only checks for invalid tokens, it does not perform a complete syntax check

- **Enable reconfiguration:**

Select the check box against this option to reconfigure the query, if the option is not selected then the **Design** tab (which is used to configure the query) will not be visible. When a query is reconfigured **Generate Parameters** button should be clicked to generate parameters for the modified query.

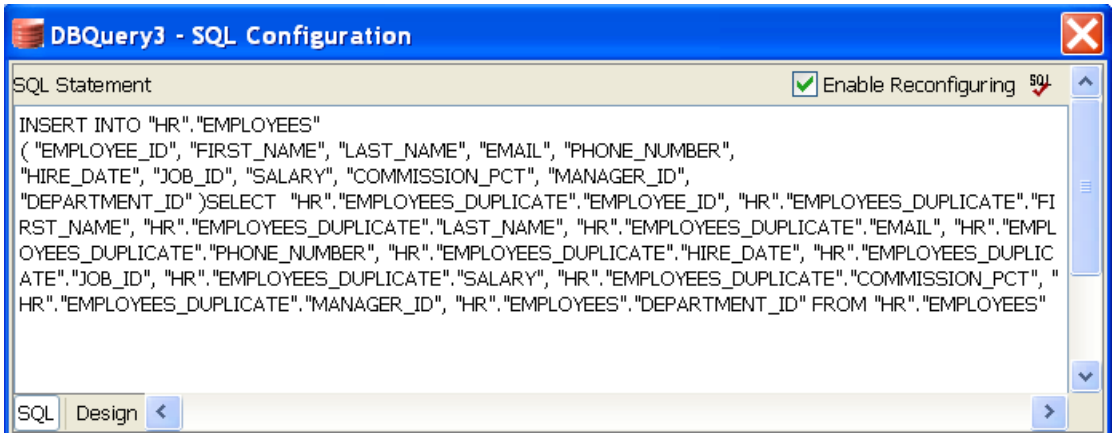


Figure 13: Generated Insert Query

6. Follow the steps from 7 and 8 as described in section Simple Insert Statement to complete the query configuration.

- **Delete Statement Configuration:**

Click ellipsis button next to **SQL configuration** property after selecting **SQL Query Type** as **DELETE** will launch the SQL Configuration Wizard which will useful to configure Insert statement.

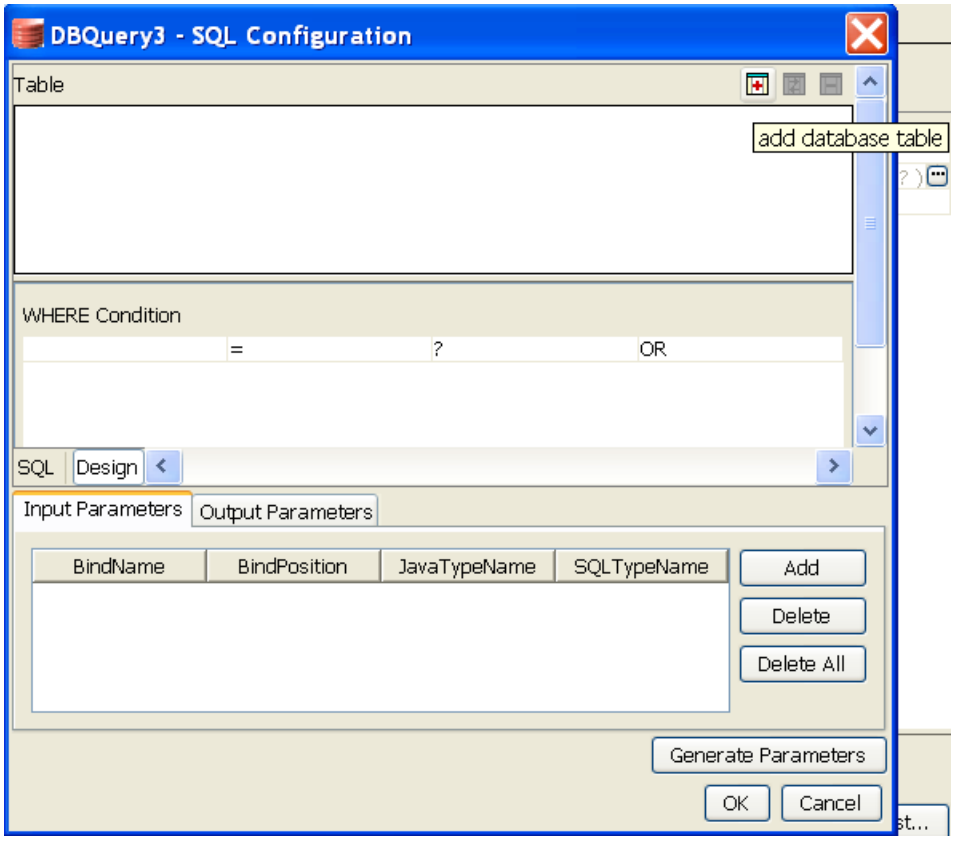


Figure 14: Sql Configuration Dialog for Delete Query

- **Simple Delete Statement:**

  Delete rows satisfying defined condition in configured table, with column values taken from input XML or with constant values.

  - Click **add database table** button to launch **Table Selection Dialog** dialog box**.**
  - Select required table as explained in **Table Selection Dialog** section. Selected table is added to the easel under **Table**

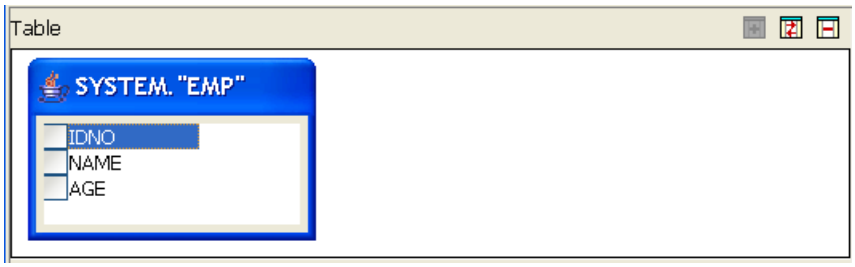


Figure 15: Selected table added to easel

  - Table can be changed by clicking **replace selected table** button and removed by clicking **remove database table** button.

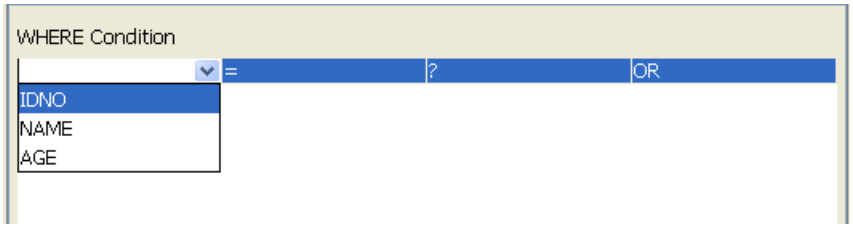


Figure 16: Adding condition on column to WHERE clause

  - Specify condition which should be satisfied for deleting row under **WHERE condition**. Select a column name in the first column on which WHERE condition has to be applied.
  - When selecting multiple columns for where condition, conditions can be combined using **AND** or **OR** under fourth column

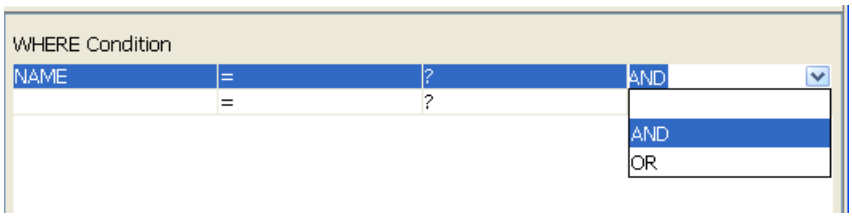


Figure 17: Combining multiple conditions for WHERE clause

  - Operator of choice can be chosen from the drop-down under second column.

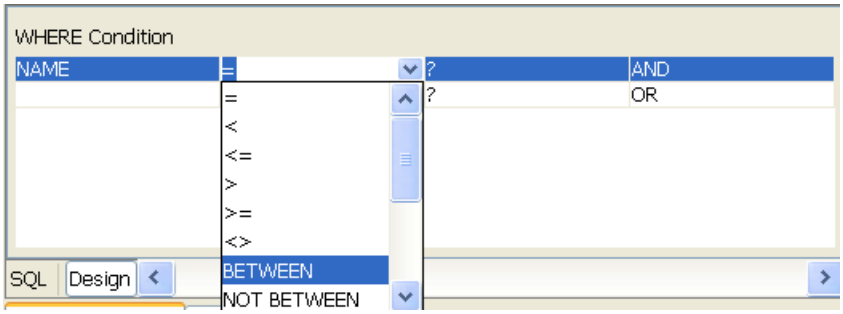


Figure 18: Selecting operator for a condition

- To specify a constant value for WHERE condition on a column, specify the required value in the third column against the required column name in **where** tab

Note:

- If the value is a string value it should be wrapped in single quotes (' ').
- ? indicates value will be taken from input or from the output of another query where possible.

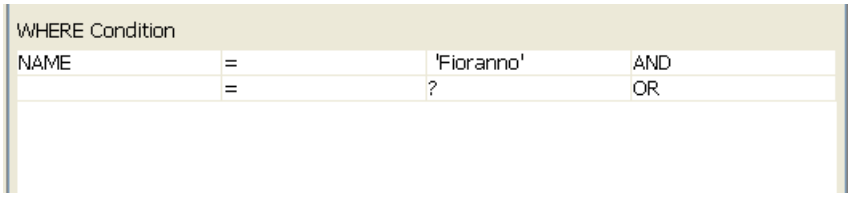


Figure 19: Specifying constant value for a column in condition for WHERE clause

- To specify WHERE condition on a column whose value is equal to value defined in another column, select the required column from drop-down in the third column against the required column name in **where** tab

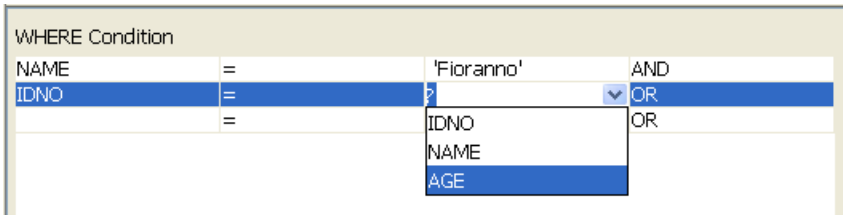


Figure 20: Specifying comparison between columns in condition for WHERE clause

- Insert statement is automatically generated and shown in the text editor under SQL Statement in SQL tab. The generated SQL can be validated by pressing **check syntax** button.

  Note: This feature only checks for invalid tokens, it does not perform a complete syntax check.

- **Enable reconfiguration:** If this option is checked then the query can be re-configured, if the option is unchecked then the **Design** tab (which is used to configure the query) will not be visible.

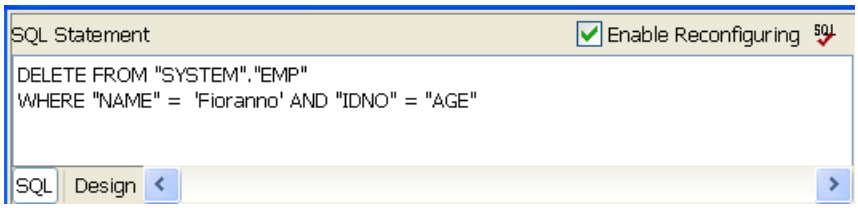


Figure 21: Generated Delete Query

- Follow the steps from 7 and 8 as described in section Simple Insert Statement to complete the query configuration.

- Update Statement Configuration:

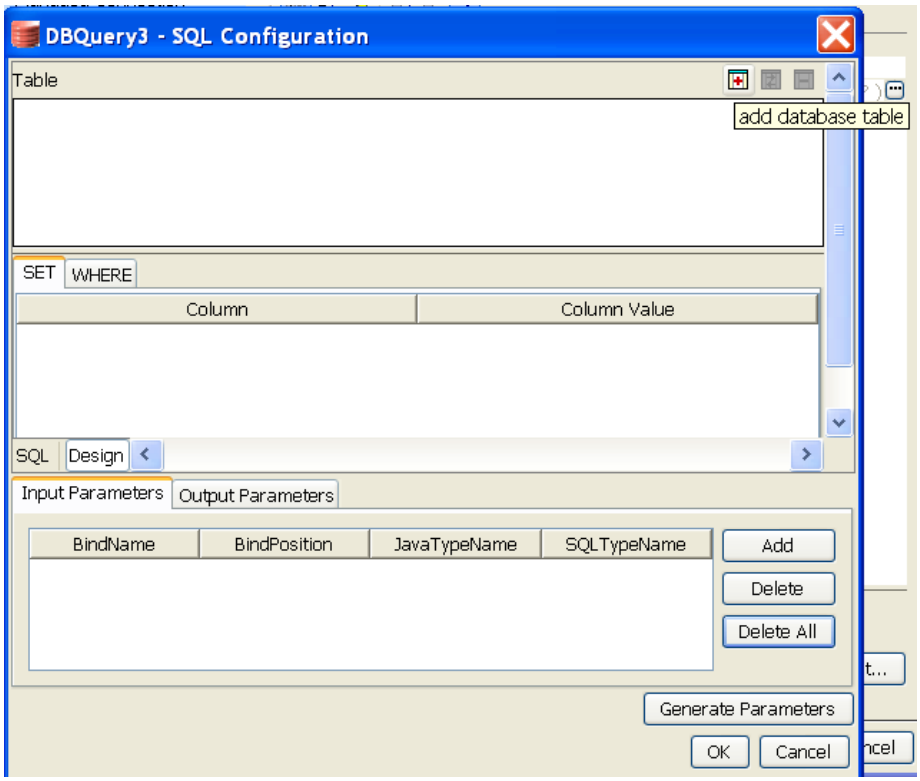


Figure 22: SQL Configuration Dialog for Update Query

Click ellipsis button next to **SQL configuration** property after selecting **SQL Query Type** as UPDATE will launch the SQL Configuration Wizard which will useful to configure update statement.

- **Simple Update Statement**:

  Update rows satisfying defined condition in configured table, with column values taken from input XML or with constant values. Condition values can also be taken from input XML or defined as constant values.

3. Click **add database table** button to launch **Table Selection Dialog** dialog box.
4. Select required table as explained in **Table Selection Dialog** section. Selected table is added to the easel under **Table.**

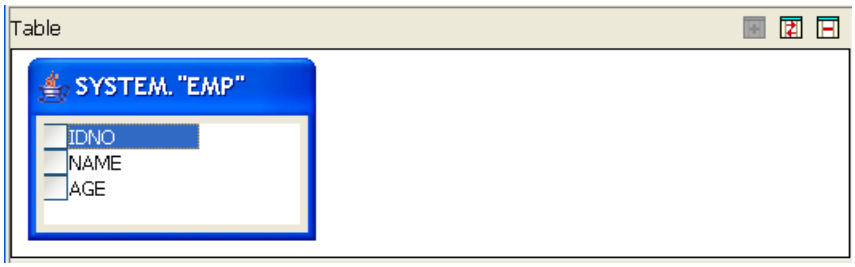


Figure 23: Selected table added to easel

5. Table can be changed by clicking **replace selected table** button and removed by clicking **remove database table** button.
6. Select the columns whose values have to be set. Figure 24 shows that NAME and AGE are selected for update.

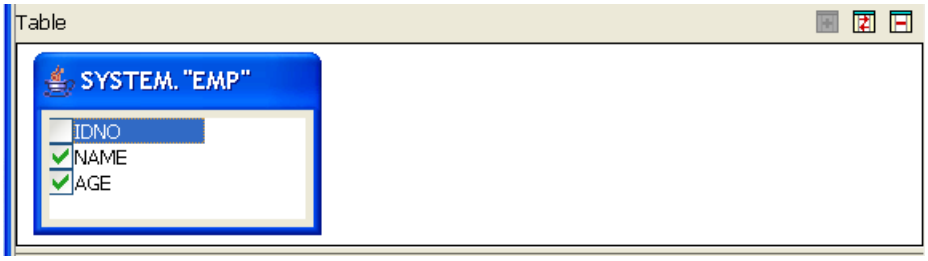


Figure 24: Selecting column for update

7. These selected columns will automatically add under the **SET** tab as shown in Figure 25.

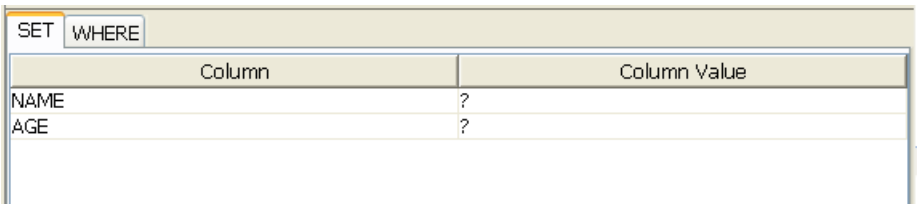


Figure 25: Columns added to SET clause

8. Click **WHERE** tab and select a column name on which **where** condition has to be applied.

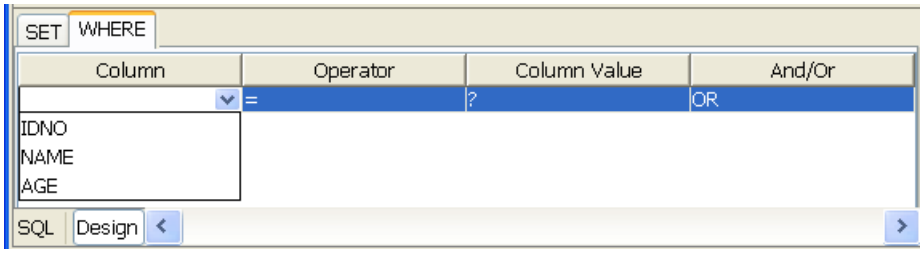


Figure 26: Adding condition on column to WHERE clause

9. When selecting multiple columns for **where** condition, conditions can be combined using **AND** or **OR** under **And/Or** column.

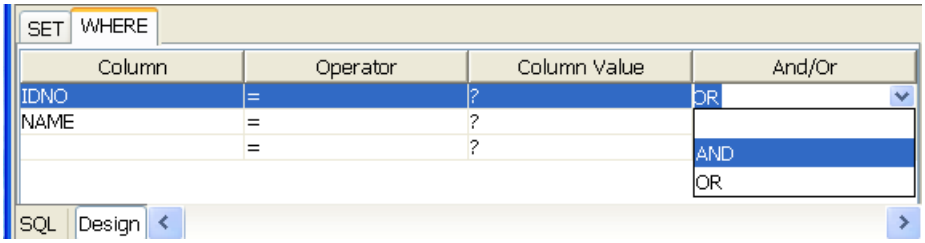


Figure 27: Combining multiple conditions under where clause

10. Operator of choice can be selected from the drop-down menu under **Operator** column.

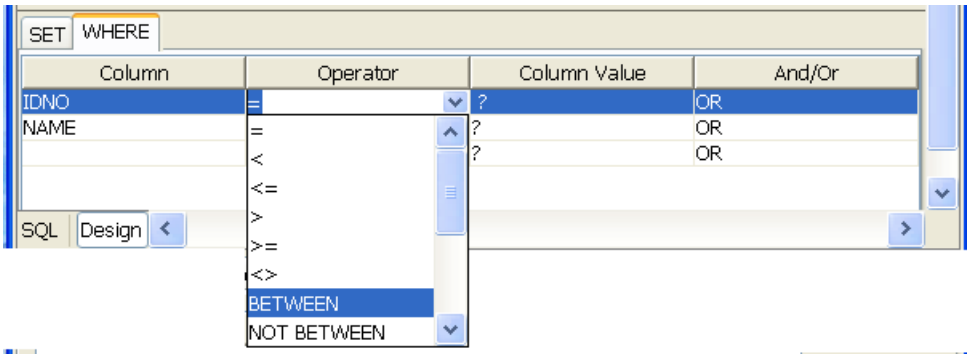


Figure 28: Selecting operator for a condition

11. Constant values can also be set to columns that have to be updated (under **SET** tab) or for values in where condition (under **WHERE** tab).

- To update a column with a constant value, specify the required value in the **Column Value** column against the required column name in **SET** tab.

Note:

- If the value is a string value it should be wrapped in single quotes (**' '**)
- **?** indicates value will be taken from input or from the output of another query where possible.

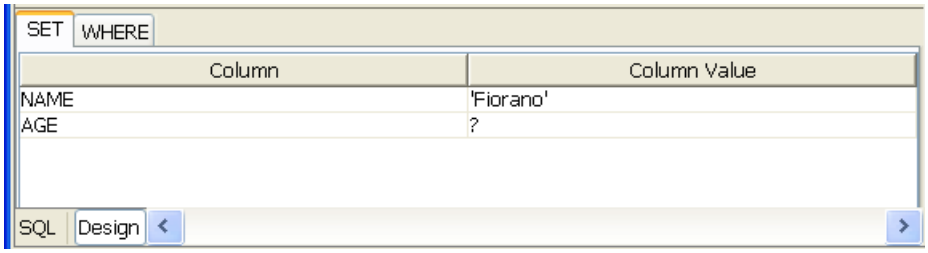


Figure 29: Specifying constant value for a column in SET clause

- To specify a constant value for **where** condition on a column, specify the required value in the **Column Value** column against the required column name in **WHERE** tab.

Note:

- If the value is a string value it should be wrapped in single quotes (' ').

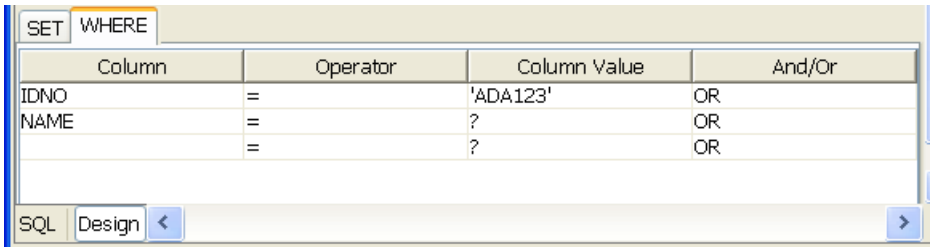


Figure 30: Specifying constant value for a column in condition for WHERE clause

- **?** indicates value will be taken from input or from the output of another query where possible.
- To specify **where** condition on a column whose value is equal to value defined in another column, select the required column from the drop-down menu in the **Column Value** column against the required column name in **WHERE** tab.

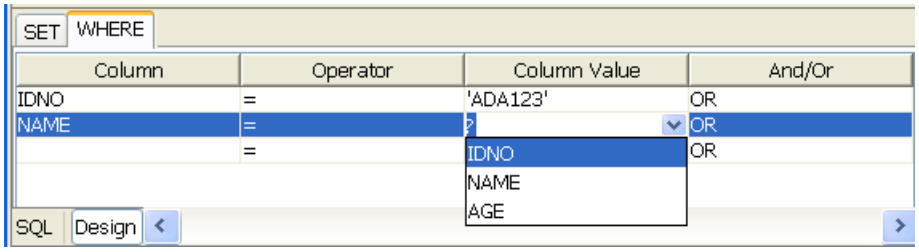


Figure 31: Specifying comparison between columns in condition for WHERE clause

12. Insert statement is automatically generated and shown in the text editor under SQL Statement in SQL tab. The generated SQL can be validated by pressing **check syntax** button.

Note: This feature only checks for invalid tokens, it does not perform a complete syntax check.

Enable reconfiguration: If the check box is selected the query can be re-configured, if the query is uncheck then the **Design** tab (which is used to configure the query) will not be visible.

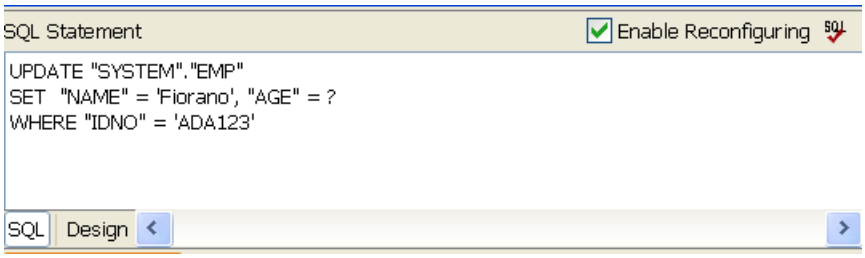


Figure 32: Generated update query

Follow the steps from 7 and 8 as described in section Simple Insert Statement to complete the query configuration.

- **Select Statement Configuration:**

  Click ellipsis button next to **SQL configuration** property after selecting **SQL Query Type** as **SELECT** will launch the SQL Configuration Wizard which will useful to configure Insert statement.

- **Simple Select Statement**

  Retrieves data from all columns or from selected columns in a configured database table.

  1. Click **add database table** button to launch **Table Selection Dialog** dialog box.
  2. Select required table as explained in **Table Selection Dialog** section. Selected table is added to the easel under **Table**

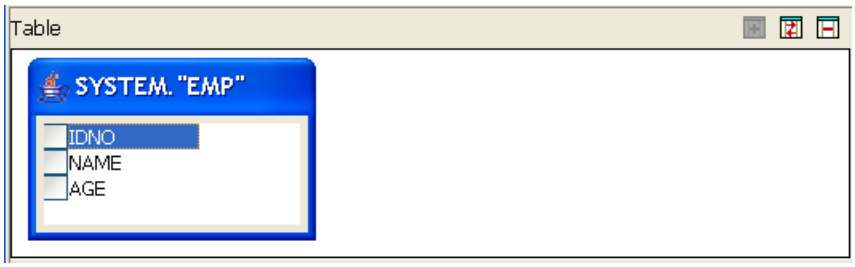


Figure 33: Selected table added to easel

  3. Table can be changed by clicking **replace selected table** button and removed by clicking **remove database table** button.
  4. To retrieve specific columns values from the table, check required columns to build a select query with specific columns. If no column is checked, then **SELECT *** is used. Select the columns in order in which they should appear they should appear in select clause.

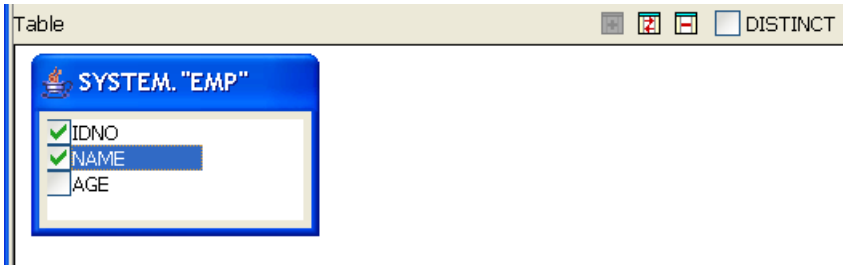


Figure 34: Selecting columns for selection

  5. Selected columns are shown under **Columns** tab. Check/Uncheck the check box in **Output** column against required column name to show/not show the corresponding column in the output XML.

For example, configuration in the following image generates IDNO in the output XML, but does not generate NAME in output XML, though values for both IDNO and NAME are retrieved from the table.

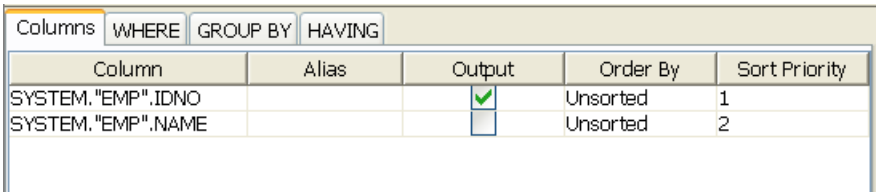


Figure 35: Selecting columns for output XML

6. To define a column alias, provide the alias name under Alias column against required column name. Aliases are useful when the column name is not intuitive or too long. When an alias is specified output XML will contain an element with defined alias name instead of the column name.

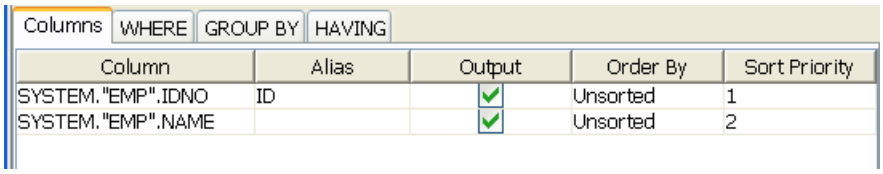


Figure 36: Defining Column Alias

7. To return unique rows check **DISTINCT**.

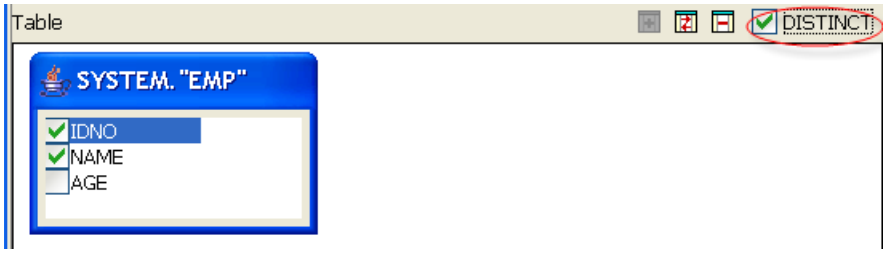


Figure 37: Distinct option to return unique values

8. Select statement is automatically generated and shown in the text editor under SQL Statement in SQL tab. The generated SQL can be validated by pressing (**check syntax**) button.

   Note: This feature only checks for invalid tokens, it does not perform a complete syntax check.

- **Enable reconfiguration:**

  Select the check box against this option to reconfigure the query, if this option is not checked then the Design tab (which is used to configure the query) will not be visible.

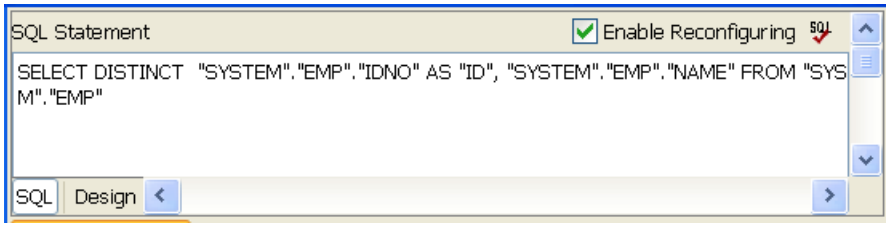


Figure 38: Generated Select Query

9. Once this query is configured, generate the input and output parameters by clicking the **Generate Parameters** button. These generated parameters are used to define the input and output port schema.
10. If the query contains any value which should be taken from input XML then the generated parameters are used to define the input and output port schemas.
11. **Add** and **Delete** buttons are used to add and delete particular parameters from the list. **Delete All** button used to delete all the parameters from the list. The Output parameter list contains only those columns for which the **Output** column is checked under **Columns** tab.

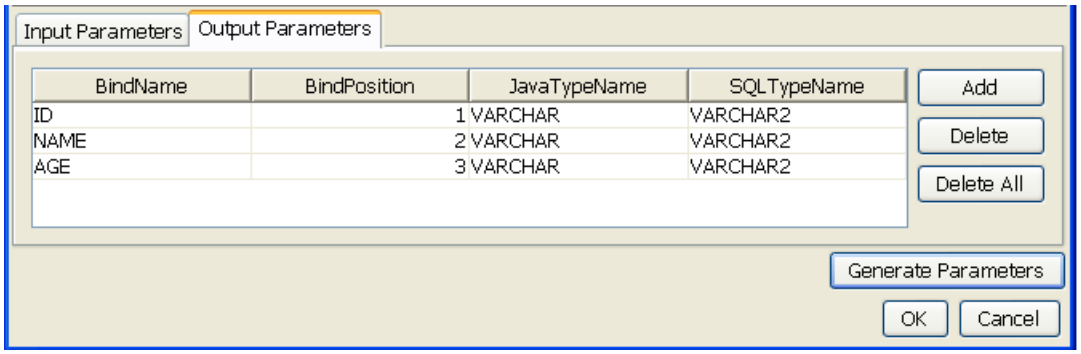


Figure 39: Generated Input/Output Parameters

Each of the columns is explained below.

Column Name	Description
BindName	This value is used to generate the schema for the query. In the Figure 40 value for IDNO (field name) is changed to ID. So the schema generated would contain ID as the first element instead of default populated value, IDNO.
Bind Position	The position in the query where this value is bound to. Note: Do not change this value.
Java TypeName	JDBC type which maps to Data Type.
SQL TypeName	This defines the data type of this column in the database table. This should be correctly defined.

12. Click **Ok** to close the dialog.

- **Select Statement with Filter:**

  Retrieves data from all columns or from selected columns in a configured database table after applying specified conditions. Condition values can be provided from input XML or as constant values.

1. Follow the steps from 1 to 7 as described in the section Simple Select Statement.
2. Click **WHERE** tab and select a **Column** name on which WHERE condition has to be applied.

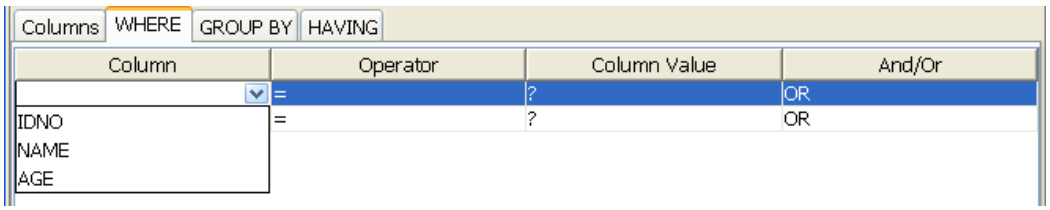


Figure 40: Adding condition on column to WHERE clause

3. When selecting multiple columns for WHERE condition, conditions can be combined using **AND** or **OR** under **And/Or** column.

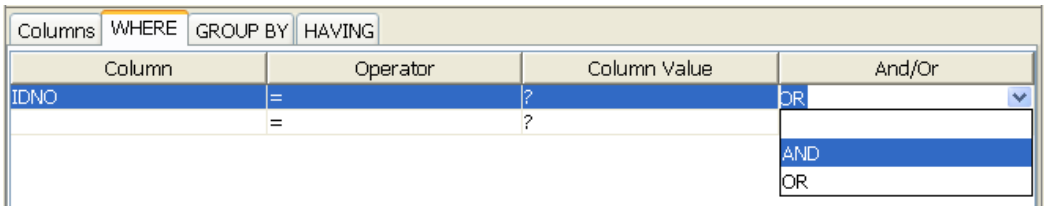


Figure 41: Adding condition on column to WHERE clause

4. Operator of choice can be selected from the drop-down list under **Operator** column.

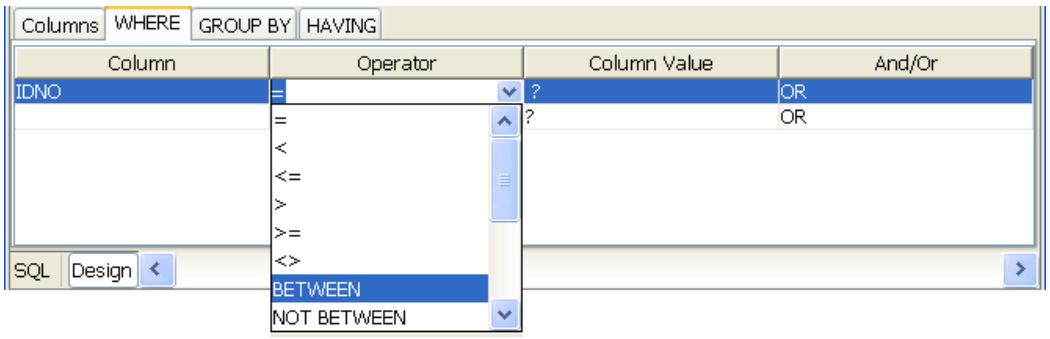


Figure 42: Selecting operator for a condition

5. Constant values can also be set for values in WHERE condition (under **WHERE** tab).
   a. To specify a constant value for WHERE condition on a column, specify the required value in the **Column Value** column against the required column name in **WHERE** tab.

      Notes:

      - If the value is a string value it should be wrapped in single quotes (' ').
      - ? indicates value will be taken from input or from the output of another query where possible.

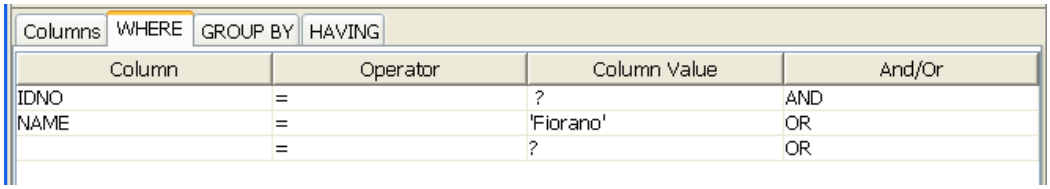


Figure 43: Specifying constant value for a column in SET clause

b. To specify WHERE condition on a **Column** whose value is equal to value defined in another **Column**, select the required **Column** from drop-down list in the **Column Value** against the required column name in **WHERE** tab.

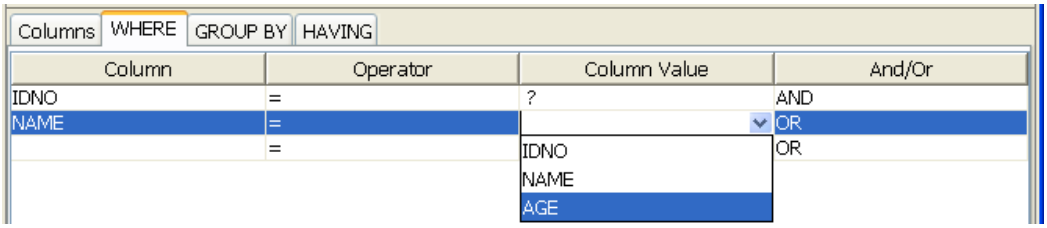


Figure 44: Specifying comparison between columns in condition for WHERE

6. Follow the steps from 8 to 10 as described in the section Simple Select Statement.

- **Select Statement with Sorting:**

  Retrieves sorted data from all columns or from selected columns in a configured database table. Data is sorted in configured order on columns configured for sorting.

  - Follow steps 1 to 7 in the section Simple Select Statement**.**
  - To specify columns which have to be sorted, select the appropriate sort order from drop-down list under **Order By** column in **Columns** tab. **Order By** for each columns has one of the following values:

Order By Value	**Explanation**
Unsorted	Data is not sorted on values in the column, that is, no order by clause will added in the SQL statement.
Ascending	Data is sorted in ascending order on values in the column, that is, order by clause will be added in the SQL statement as ORDER BY <column name> ASC.
Descending	Data is sorted in descending order on values in the column, that is, order by clause will be added in the SQL statement as ORDER BY <column name> DESC.

Default	Data is sorted in default order for order by clause on values in the column, that is, order by clause will be added in the SQL statement as ORDER BY <column name>.

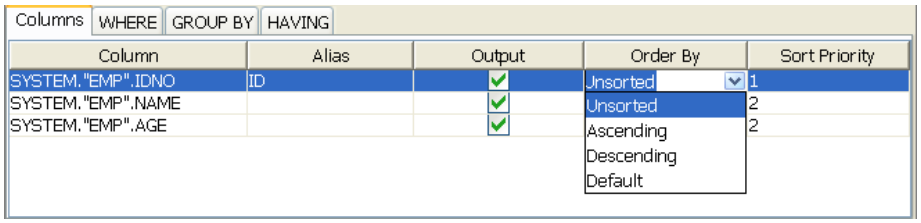


Figure 45: Selecting sorting order for column

- When multiple columns have to be sorted, sorting priority for each column can be set under Sort Priority. Columns are sorted in order of increasing Sort Priority that is column with minimum value for Sort Priority is order first.

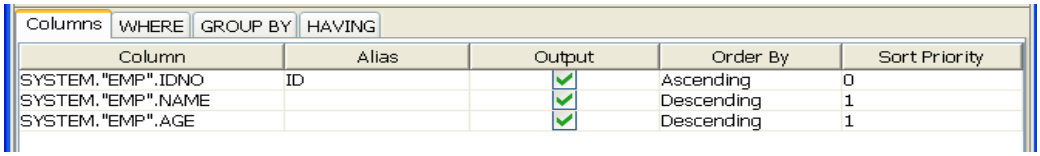


Figure 46: SQL Statement with different columns sorted in different order

When values of Sort Priority for multiple columns are same, columns are sorted in the order in which they appear in select clause.

- Follow the steps from 8 to 10 as described in the section Simple Select Statement.

- **Select Statement with Grouping:**

Retrieves data, after applying grouping conditions, from all columns or from selected columns in a configured database table.

Note: Grouping functions are not provided in query builder. Grouping conditions have to be explicitly added by editing the SQL statement either before closing the SQL Configuration Wizard.

4. Follow steps 1 to 7 in the section Simple Select Statement**.**
5. Click **GROUP BY** tab and check under **Select** against the columns under **Group By** on which group by condition should be applied.

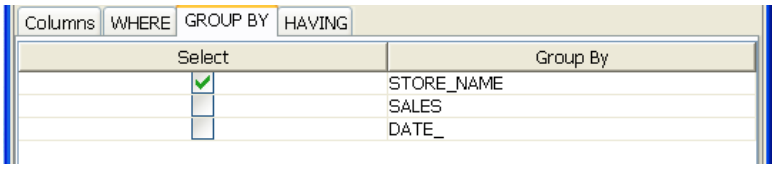


Figure 47: Selecting columns for grouping condition

6. To filter the results click **HAVING** tab and define required conditions. **HAVING** tab has functionality similar to **WHERE** tab (described in Select Statement with filter).

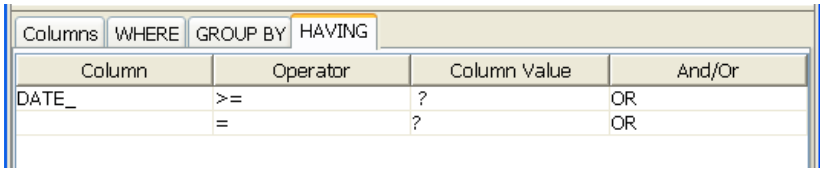


Figure 48: Adding condition to HAVING clause

7. Select required columns under Tables.

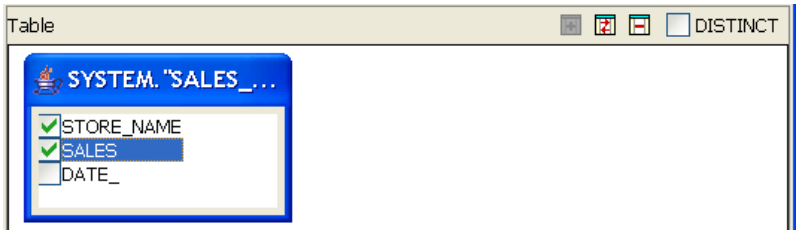


Figure 49: Selecting required columns

8. Now generate the input and output parameters as described in step 9 in section Simple Select Statement.
9. Edit the Select statement is which was shown in the text editor under SQL Statement in SQL tab.

   Note: Editing Select and HAVING clauses should be last action before closing the dialog.

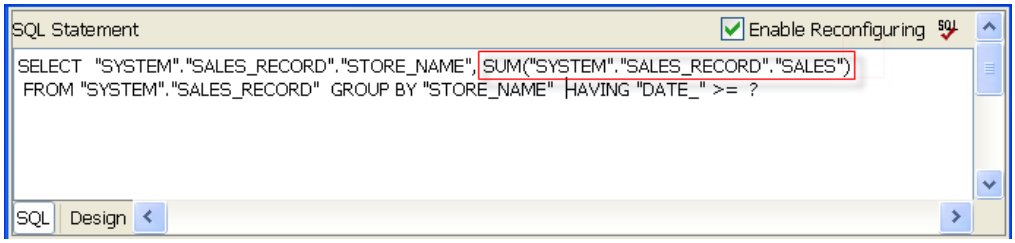


Figure 50: Generated SQL Select Query

10. Click the **Ok** button to close the dialog box.

- **Select Statement with Multiple Tables**

  Retrieves data from all columns or from selected columns from multiple configured database tables. Currently retrieving data from more than one table is not supported.

4.3.4.2.1.4 Single Batch Mode

This option determines whether the component should send entire result of a query as a single message or as multiple messages.

- **Yes:** Complete result of the query from input request is sent out as a single message. If the result set returned is huge then the component can run into memory problems and stop. When this value is selected, property **Batch Size** is hidden.

- **No:** Result of query from input is split and sent out as multiple messages. Number of rows from result to be included in each output message is determined by property **Batch Size**. When this value is selected, property **Batch Size** is shown.

Example: If a query returns 100 rows, and the batch size is set to 10 then 10 outputs will be generated each contains 10 rows.

4.3.4.2.1.5 Batch Size

This property is visible when the value of property **Single Batch Mode** is set as yes. The property determines the number of units of result an output message contains.

Each row in a result set (typically result of a select query) or an update result (result of update, delete, insert operations) is treated as unit of result.

Example: Consider a stored procedure that returns a result of select query followed by three update queries and another select query. Assume first select return 18 rows and second query returns 11 rows. If **Single Batch Mode** is set as no and **Batch Size** is set as 10 then there will be four output messages.

1. first message: first ten rows from first query
2. second message: remaining 8 rows from first query and two update query results
3. third message: third update query result and first nine rows of second select query
4. fourth message: remaining two rows from second query.

4.3.4.2.2 Using Named Configurations

Changes are done in eStudio w.r.t to Named Configurations. With the implementation of Named Configurations MCF panel is changed as shown in Figure below.

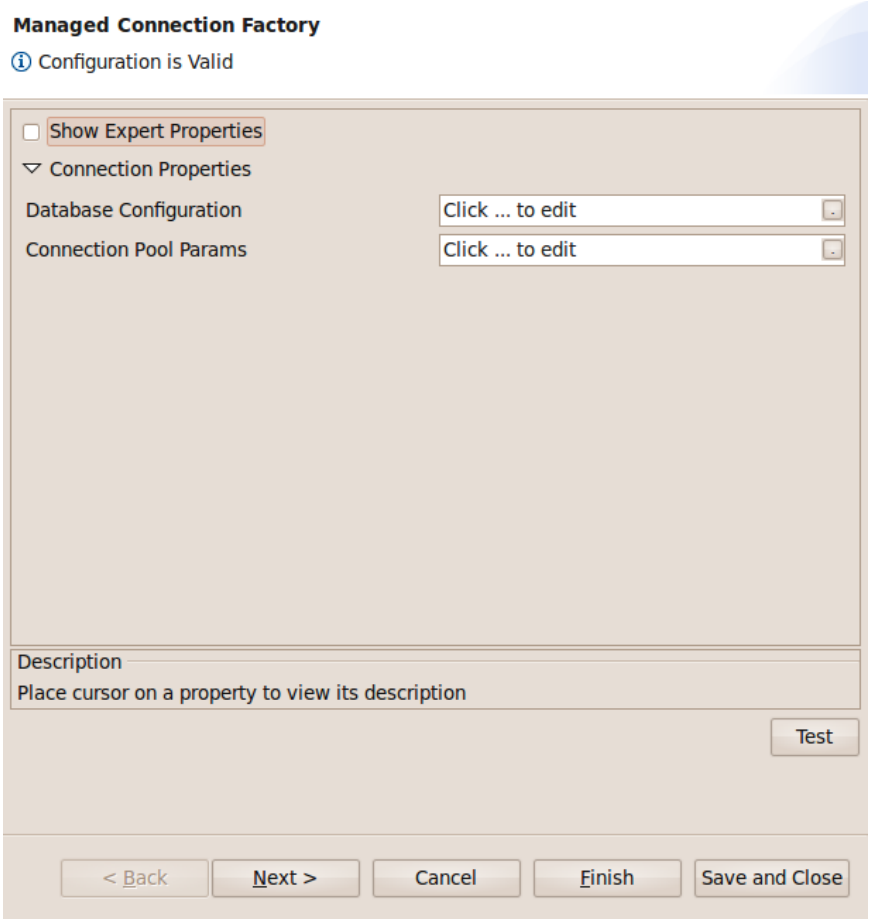


Figure: Modified Managed Connection Factory

Some properties are grouped as one property in the modified configuration panel. Database, Driver, URL, User name, Password, Connection ping sql and Connection Properties which were present previously in the MCF panel are grouped under the name Database Configuration.

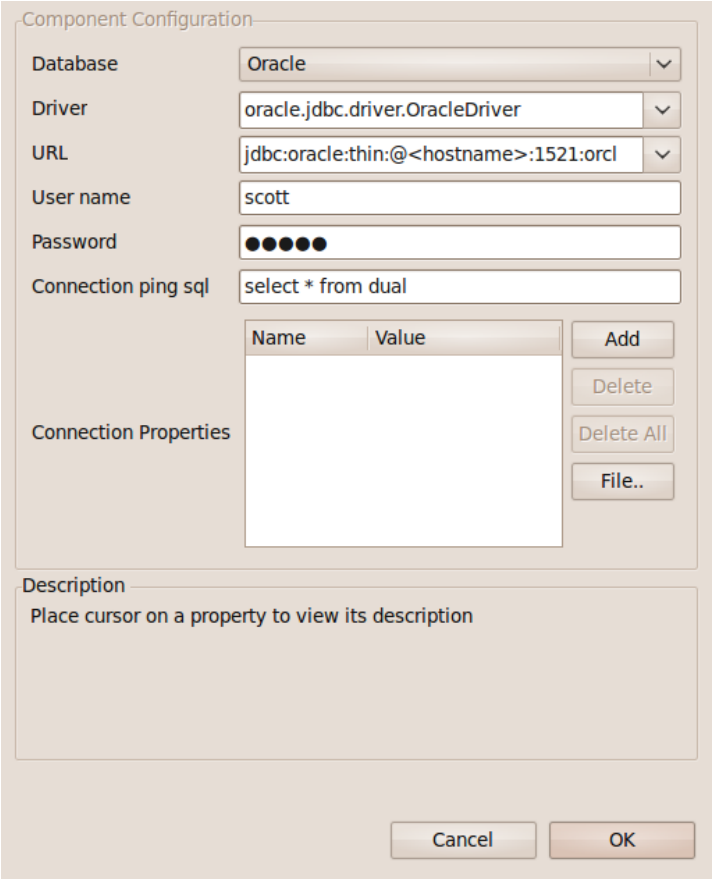


Figure: Database Configuration

4.3.4.3 Input

The input of the component varies with the kind of query configuration. If the query has input parameters which have to be provided dynamically, these parameters will be included in the input schema of the component if **Generate Parameters button** is clicked after configuring SQL.

4.3.4.3.1 SELECT

In case of select query, the input parameters that have to be provided will be added as child elements to the SELECT element. When the query is as shown in Sample Query 1, the input schema will contain the parameters which have to be provided for the query as shown in the Figure 51.

```
SELECT  "MIGWCN"."CNSVID", "MIGWCN"."CNTIME" FROM "PUBLIC"."MIGWCN"
WHERE "CNSVID" = ?
HAVING "CNMSG_" = ?
```

Sample Query 1, Select with Where and Having clauses

XML Item	Value Type
<ns1:SELECT>	
<ns1:CNSVID>	#string
<ns1:CNMSG_>	#string

Figure 51: Input schema – Sample Query 1

4.3.4.3.2 INSERT

In case of insert query, the input parameters that have to be provided will be added as child elements to the INSERT element present in the input port schema of the component. The input schema corresponding to Sample Query 2 is as shown in the Figure 53.

```
INSERT INTO "PUBLIC"."MIGWCN"
( "CNTLC_", "TIF_RECORDID" )
VALUES ( ?, ? )
```

Figure 52: Sample Query Simple Insert

<ns1:SELECT>	
<ns1:CNSVID>	#string
<ns1:CNMSG_>	#string

Figure 53: Input Schema – Sample Query 2

4.3.4.3.3 UPDATE

In case of update query, the input parameters that have to be provided will be added as child elements to the UPDATE element present in the input port schema of the component. The input schema corresponding to Sample Query 3 is as shown in the Figure 55.

```
UPDATE "PUBLIC"."RV_EXITS"
SET  "TRANSACTION_STAMP" = ?
WHERE "FIRMWARE" = ?
```

Figure 54: Sample Query Simple Update

<ns1:UPDATE>	
<ns1:TRANSACTION_STAMP>	#string
<ns1:FIRMWARE>	#string

Figure 55: Input Schema – Sample Query 3

4.3.4.3.4 DELETE

In case of update query, the input parameters that have to be provided will be added as child elements to the DELETE element present in the input port schema of the component. The input schema corresponding to Sample Query 4 is as shown in the Figure 57.

```
DELETE FROM "PUBLIC"."RV_EXITS"
WHERE "TRANSACTION_STAMP" = ? OR "FIRMWARE" = ? OR "ENTRY_TIME" = ? OR
"EXIT_TIME" = ? OR "TRANSACTION_NUMBER" = ? OR "TICKET" = ?
```

Figure 56: Sample Query Delete with Where clause

<ns1:DELETE>	
<ns1:TRANSACTION_STAMP>	#string
<ns1:FIRMWARE>	#string
<ns1:ENTRY_TIME>	#string
<ns1:EXIT_TIME>	#string
<ns1:TRANSACTION_NUMBER>	#string
<ns1:TICKET>	#string

Figure 57: Input Schema - Sample Query 4

When the property Use Connection details from input is chosen, an additional element ConnectionFactorySettings is added to the input schema, as shown in the figure. Properties that are used to create the connection are present under this element.

XML Item	Value Type
<ns1:SELECT>	
<ns1:ConnectionFactorySettings>?	
<ns1:ConnectionProperties>?	
<ns1:Database>?	HSQL | IBM DB2 | Kingbase | Mckoi |
<ns1:Driver>?	#string
<ns1:JDBCURL>?	#string
<ns1:Username>?	#string
<ns1:Password>?	#string
<ns1:AdvancedConnectionProperties>?	#string
<ns1:AutoCommit>?	true | false
<ns1:AdvancedProperties>?	
<ns1:ConnectionPingSQL>?	#string
<ns1:EnableJDBCLogging>?	true | false

Figure 58: Input schema with ConnectionFactorySettings

4.3.4.4 Output

The output schema is auto generated based on the configuration provided. An element Result will be used to represent the result of the query configured. In case of SELECT query, an element Row with **zero-many** cardinality will be added to this element. Each Row element represents a single entry in the result set obtained. If **Generate Parameters button** is clicked after configuring SQL different elements corresponding to columns in the result set will be added as child elements to the row elements. The sample output for the Sample Query 1 is as shown in Figure 58.

<ns1:Result>	
<ns1:Row>*	
<ns1:CNSVID>	#string
<ns1:CNTIME>	#string

Figure 58: Output Schema – Sample Query 1

In case of UPDATE, DELETE, INSERT statements an element UpdateCount will be added as child to the Result element which holds the number of rows updated as a result of execution of the query. The output schema for queries 2, 3 and 4 is as shown in Figure 59.

```
<ns1:Result>
    <ns1:UpdateCount>          #string
```

Figure 59: Output Schema – Sample Queries 2,3,4

4.3.4.5 Functional Demonstration

4.3.4.5.1 Scenario 1

Execution of a select query

Start mckoiDB present at %FIORANO_HOME%\esb\samples\mckoiDB by executing CreateMckoiDB.bat and RunMckoi.bat. Configure the DBQuery component as shown in Figure 2 and configure a select statement as described in Simple Select Statement section. Use feeder and display components (shown in Figure 60) to create a flow to send sample input and check the response respectively. DBQuery component configures to a select query. As shown in sample input (Figure 61) DBQuery takes the `CUSTOMER` column value and returns that particular row. The selected row will be organized as XML and it will be returned to the output port as shown in sample output (Figure 62).

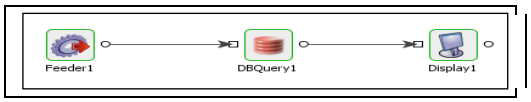


Figure 60: Flow for Scenario 1

Send input message, shown in Figure 60, from feeder and notice the output similar to the one shown in Figure 61 in display.

```
<ns1:SELECT xmlns:ns1="http://www.fiorano.com/fesb/activity/DBQueryl/In">
    <ns1:CUSTOMER>CUSTOMER</ns1:CUSTOMER>
</ns1:SELECT>
```

Figure 61: Sample Input for Scenario 1

```
<?xml version="1.0" encoding="UTF-8"?>
<Result xmlns="http://www.fiorano.com/fesb/activity/DBQueryl/Out">
    <Row>
        <CUSTOMER>CUSTOMER</CUSTOMER>
        <DEADLINE>2007-10-12 00:00:00.0</DEADLINE>
        <SCREENINGFREQUENCY>10</SCREENINGFREQUENCY>
        <PRIORITY>80</PRIORITY>
        <VERTICAL>VERTICAL</VERTICAL>
        <DURATION>10</DURATION>
        <RELEASEDATE>2007-10-12 00:00:00.0</RELEASEDATE>
        <TIF_RECORDID>1</TIF_RECORDID>
        <TIF_STATUS>TIF_STATUS</TIF_STATUS>
    </Row>
</Result>
```

Figure 62: Sample Output for Scenario1

4.3.4.6 Useful Tips

- Only one query can be configured in the adapter. Please use the DB component if multiple query configurations need to be configured for a single instance.
- Only one query can be processed per message. If multiple queries have to be processed, use XMLSplitter to split the message into multiple messages each containing a single query.
- It is recommended NOT to use JDBC-ODBC Bridge driver to connect to any RDBMS in your production environment. Please use a commercial JDBC driver instead.
- The JDBC drivers or the resources must be directly added onto the JDBC system lib and not as resource to the DBQuery component itself. To add JDBC drivers to DBQuery component, please refer to section 3.3.3.3.

4.4 Error

The Error category consists of ExceptionListener component. The following section describes this component.

4.4.1 Exception Listener

The Exception Listener component listens for exceptions from components running in a Fiorano network. It connects to the configured Enterprise Server and obtains details of all running Peer Servers whose names match the regular expression provided in the configuration. It then subscribes for messages on all topics, whose names match the regular expression in configuration, for each Peer Server identified. With the default values for regular expressions the component listens for messages on topics whose names end with ON_EXCEPTION on all Peer Servers. It sends out messages received from configured topics on its output port. Messages sent out from the component have an additional string property ESBX_SYSTEM_SOURCE_TOPIC_NAME that contains the name of topic to which the message was actually sent.

4.4.1.1 Configuration and Testing

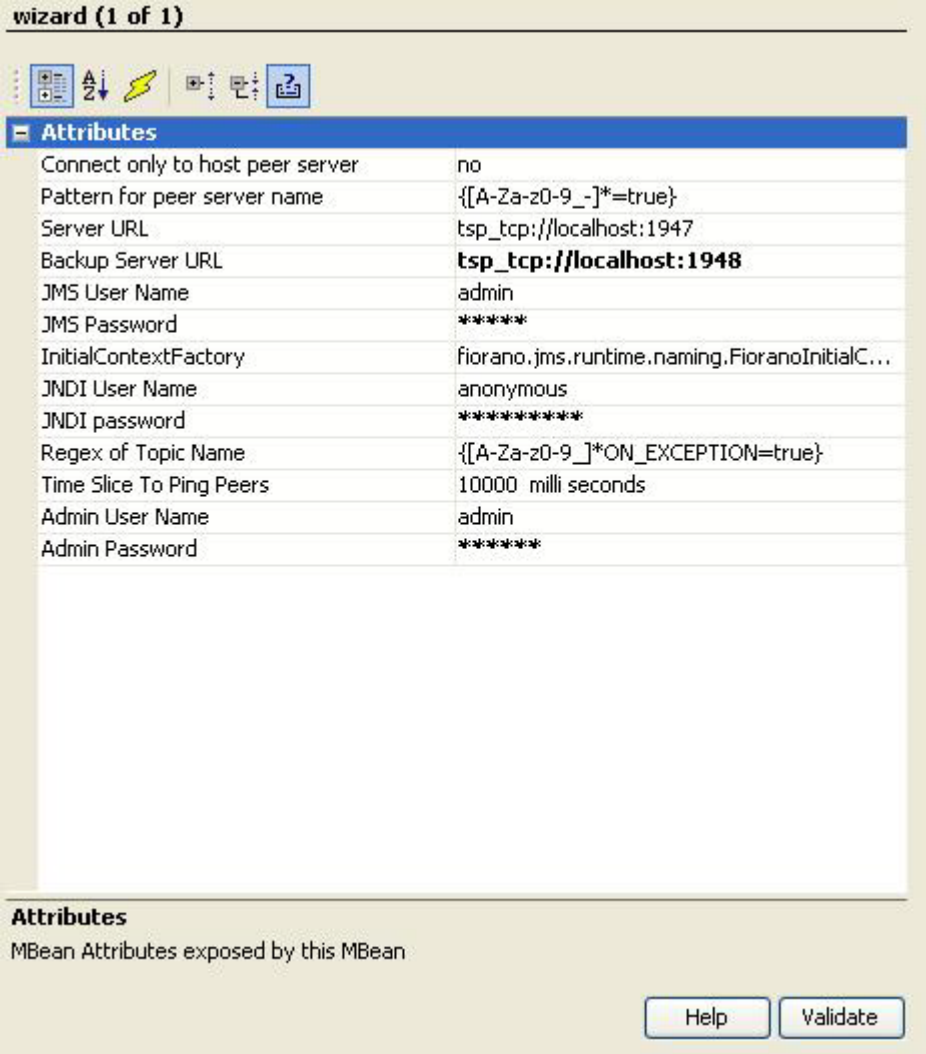


Figure 1: Configurations of Exception Listener

4.4.1.1.1 Attributes

4.4.1.1.1.1 Connect only to host peer server

This property determines whether the component should listen for messages only from topics on the Peer Server on which the component is running.

- **Yes:** The component listens for messages only from the configured topics on the Peer Server on which this component is running. When this value is selected neither Enterprise Server connection is needed to fetch the details of Peer Servers nor the regular expression for matching the Peer Server name and hence, properties **Pattern for peer server name**, **Server URL**, **Back Up Server URL**, **JMS User Name**, **JMS Password** are hidden.
- **No**: The component listens for messages from the configured topics on configured Peer Servers. When this value is selected an Enterprise Server connection to fetch the details of Peer Servers and the regular expression for filtering Peer Servers are needed and hence, properties **Pattern for peer server name**, **Server URL**, **Backup Server URL**, **JMS User Name**, **JMS Password** are shown.

4.4.1.1.1.2 Pattern for peer server name

This property determines the Peer Servers from the Fiorano network that have to be monitored. The Fiorano network from which Peer Servers are filtered is determined by the Enterprise Server details provided against properties **Server URL** and **Backup Server URL.**

Click the ellipsis button to specify the pattern matching condition(s). A dialog box appears containing a table with two fields:

- **Name**

  This column contains the regular expression that should be used to match the Peer Server name. Regular expressions should follow the syntax described java.util.regex.Pattern. Please check the following link for details http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html

- **Value**

  The value in this column determines whether Peer Servers whose names match regular expression defined in **Name** column should be monitored or not. If this column value is, ignoring case, true then the Peer Servers whose names match the regular expression is selected. If the value is false, Peer Servers whose names match the regular expression is not selected.

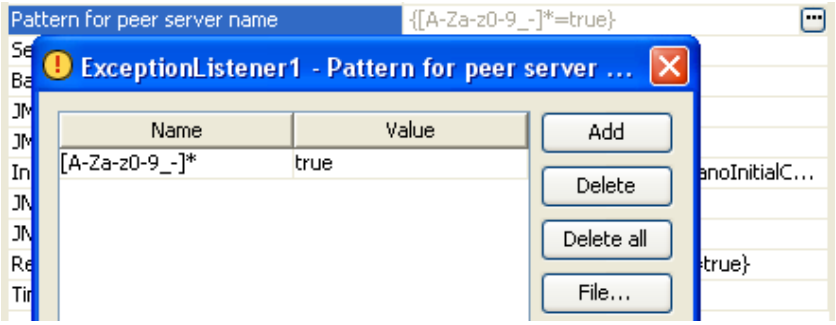


Figure 2: Defining regular expression for filtering peer servers

If no patterns are defined, then no Peer Server will be selected and the component does not listen for any messages. When there are multiple pattern matches defined for each Peer Server name in Fiorano network, validation against all pattern matching conditions is performed. If even a single pattern matching condition is not satisfied, the Peer Server will not be selected.

Patterns can be loaded from a properties file using **File...** button. Click the following link http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html#load(java.io.InputStream for details on properties file.

Example:

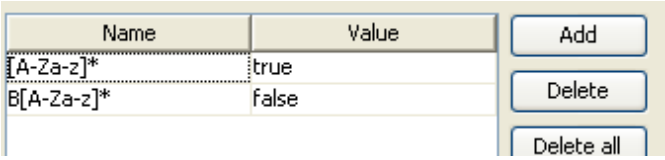


Figure 3: Sample pattern matching condition

Pattern matching condition in Figure 3 matches and selects all Peer Servers whose name has English alphabets only, but does not start with B.

4.4.1.1.1.3 Server URL

The URL of Enterprise Server to which the Peer Servers that have to be monitored are connected. For more information, refer to section Connection to Enterprise Server.

4.4.1.1.1.4 Back Up Server URL

The alternate URL that should be tried for connecting to the Enterprise Server if the Enterprise Server cannot be connected to using the URL mentioned against property **Server URL**. For more information, refer to section Connection to Enterprise Server.

Note: In case of Enterprise Servers in HA mode, this should point to Secondary Server URL if the primary is set against **Server URL** property and vice-versa.

4.4.1.1.1.5 JMS User Name

The user name to connect to the Enterprise Server. For more information, refer to section Connection to Enterprise Server.

4.4.1.1.1.6 JMS Password

The password to connect to the Enterprise Server. For more information, refer to section Connection to Enterprise Server.

4.4.1.1.1.7 InitialContextFactory

InitialContextFactory implementation class name which can be used to create Initial Context that is used to lookup Peer Server's administration object. This is required to find the list of topics created on Peer Server.

Note: This property need not be changed normally.

4.4.1.1.1.8 JNDI User Name

The user name required to create Initial Context for looking up objects on Peer Server. This user should be present on each of the Peer Servers and have permission to perform lookup for JMS objects.

4.4.1.1.1.9 JNDI Password

The password for user provided against property **JNDI User Name** required to create Initial Context for looking up objects on Peer Server.

4.4.1.1.1.10 Regex of Topic Name

This property determines the topics on Peer Servers from the Fiorano network that have to be monitored. Refer help for property **Pattern for peer server name** to specify pattern matching conditions for topic names.

4.4.1.1.1.11 Time Slice To Ping Peers

This property determines the time interval in milliseconds after which the component has to periodically poll for changes in Peer Servers and topics. At every poll interval, the component does the following:

- Fetches all Peer Servers in Fiorano network or just the Peer Server on which the component is running based on property **Connect only to host peer server.**
- For each Peer Server name that matches the pattern matching conditions defined, checks if the Peer Server is available in network.
    - If the Peer Server is not available in the network but the component has created and cached a connection to the Peer Server during previous polls, then the connection is discarded.
    - If the Peer Server is available in the network, but component has not created and cached a connection to the Peer Server during previous polls, then a new connection is made and cached.
- For each Peer Server that is available in network and connected to the component, all the topics are listed.
- If the topic matches the pattern matching conditions defined and is not in the list of topics to which the component subscribes, a subscription to the topic is made and a message listener is set.

4.4.1.1.1.12 Admin User Name

The user name required to create subscribers on topics on specified peer servers. This user should be present in admin user group.

4.4.1.1.1.13 Admin Password

The password for user provided against property **Admin User Name** required to subscribe to topics on specified peer servers.

4.4.1.1.2 Connection to Enterprise Server

A connection to Enterprise Server is required only if property **Connect only to host peer server** is set to yes. The connection is made during the component launch using the properties **Server URL**, **Back Up Server URL**, **JMS User Name,** and **JMS Password.** If the Enterprise Server connection cannot be successfully made during the component launch, the component is automatically stopped and the error is logged in the error logs of the component.

Whenever, the component tries to check for newly added Peer Servers or topics which match configured criteria based on property **Time Slice To Ping Peers** it validates connection with Enterprise Server and if the connection is found invalid, it tries to reconnect using the configured Enterprise Server details.

4.4.1.2 Functional Demonstration

4.4.1.2.1 Scenario 1

The Regular Expression can be specified to match or differ based on the configuration. To allow topics whose name matches with the regular expression, the value column for that Regular Expression must be set as true. To allow topics whose name does not match with a regular expression, the value column for that regular expression must be set as false.

To make the exception listener listen to the error ports of all the Event Processes other than itself, the regular expression list can be specified as shown in Figure 4.

Name	Value
[A-Za-z0-9_]*ON_EXCEPTION	true
<APP_GUID>[A-Za-z0-9_]*	false

Figure 4: Configuration to listen on exception ports of only one event process

Similarly, regular expressions can be used to match the peers on which the exception listener has to subscribe to.

4.4.1.2.2 Scenario 2

Configure the Exception Listener as described in Configuration and Testing section; configure a CBR with any schema. Use feeder to send an improper message to the CBR and display component to check the output message send by Exception Listener (which is picked from the exception port of CBR) on its output port.

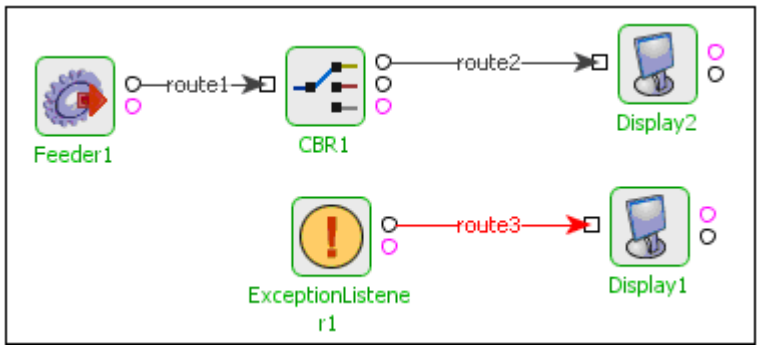


Figure 5: Event process for scenario 2

Sample Input

```
<shiporder orderid="orderid">
          <orderperson>orderperson</orderperson>
          <shipto>
                  <name>name</name>
                  <address>address</address>
                  <city>city</city>
                  <country>country</country>
         </shipto>
         <item>
                  <title>title</title>
                  <note>note</note>
                  <quantity>204</quantity>
                  <pricedf>sadsd</price>
         </item>
</shiporder>
```

Figure 6: Sample Input

Sample Output

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:Error xmlns:ns1="http://www.fiorano.com/fesb/activity/fault">
         <errorCode/>
         <errorMessage>error_processing_messagenet.sf.saxon.trans.DynamicError:
         net.sf.saxon.trans.DynamicError: org.xml.sax.SAXParseException: The
         element type "pricedf" must be terminated by the matching end-tag
         "&lt;/pricedf&gt;".</errorMessage>
         <errorDetail/>
</ns1:Error>
```

Figure 7: Sample Output

4.4.1.3 Use Case Scenario

In a sales force integration scenario, exception listener component listens for exceptions which might occur at any step of the process.

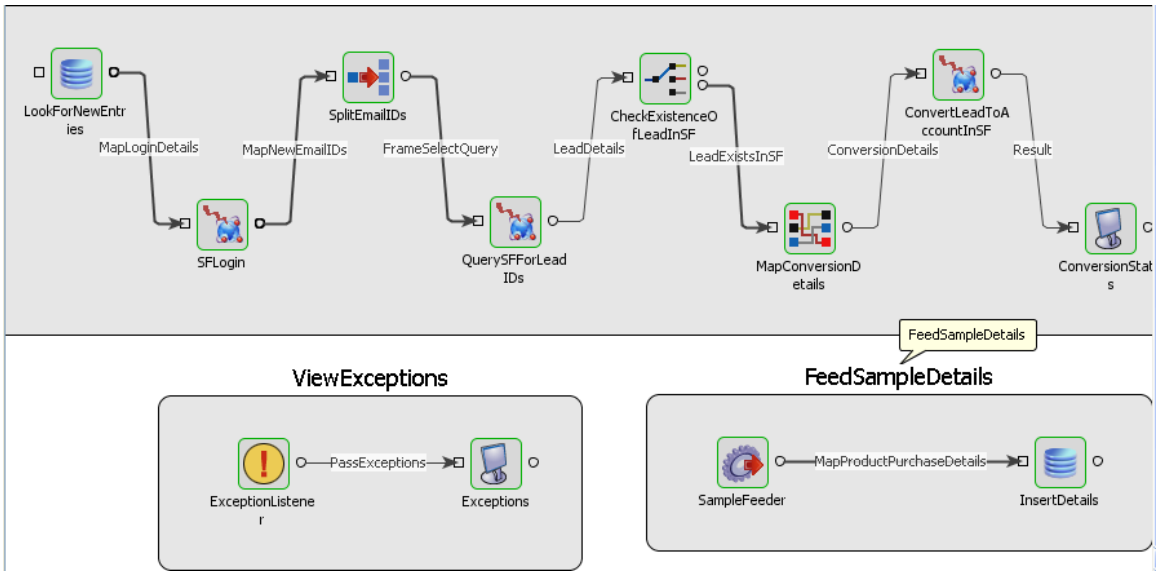


Figure 8: Salesforce Integration scenario

The Event Process demonstrating this scenario is bundled with the installer.

Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Studio.

4.4.1.4 Useful Tips

- This component does not listen for specified topics, if the Enterprise Server is down.
- This component listens only to topics that match the regular expression list provided in the CPS.
- Regular expressions are case sensitive.
- [A-Za-z0-9_]* can be used to match any valid character sequence.
- The regular expression for a string with prefix as PREFIX is PREFIX[A-Za-z0-9_]*
- The regular expression for a string with suffix as SUFFIX is [A-Za-z0-9_]*SUFFIX.

4.5 File

File Reader component reads file from the file system and send their contents to the output port. The source file can either be unstructured (plain) text or binary.

File Reader is capable of handling:

Text/Flat files

Text files may be read from a specified file in an unstructured fashion. Unstructured text in the files to be transferred is read as it is and sends on the output port of the FileReader component.

Note: The unstructured plain text needs to be transferred into its corresponding XML using the Text → XML component**.**

Binary Files

Binary file contents are read as bytes of data from the source file and are sent in chunks or bundles to the output port of the FileReader component based on the configuration properties of this component.

File Reader uses core Java APIs to read the files.

4.5.1 File Reader

The FileReader component reads files from the file system and sends their contents to the output port. The source file can either be text or binary.

Text file

Text file may be read from a specified file in an unstructured fashion and the content is sent in a single message.

Binary file

Binary file contents are read as bytes of data from the source file and are sent in chunks or bundles to the output port of the FileReader component based on the configuration properties of this component.

Points to note

- The component runs on the peer server and therefore the file paths and directories mentioned in the CPS should be valid on the machine where the peer server is running. If the component fails over to another peer server, ensure that the machine on which the secondary peer server is running must have the same path available.
- The unstructured plain text can be transformed into its corresponding XML using the Text2XML component.
- Number of outgoing messages for an input binary file = ceil (Size of File/ Chunk Size).

4.5.1.1 Interaction Configurations

Business logic configuration details are configured in the **Interaction Configurations** panel. Figure 1 below illustrates the panel with expert properties view enabled.

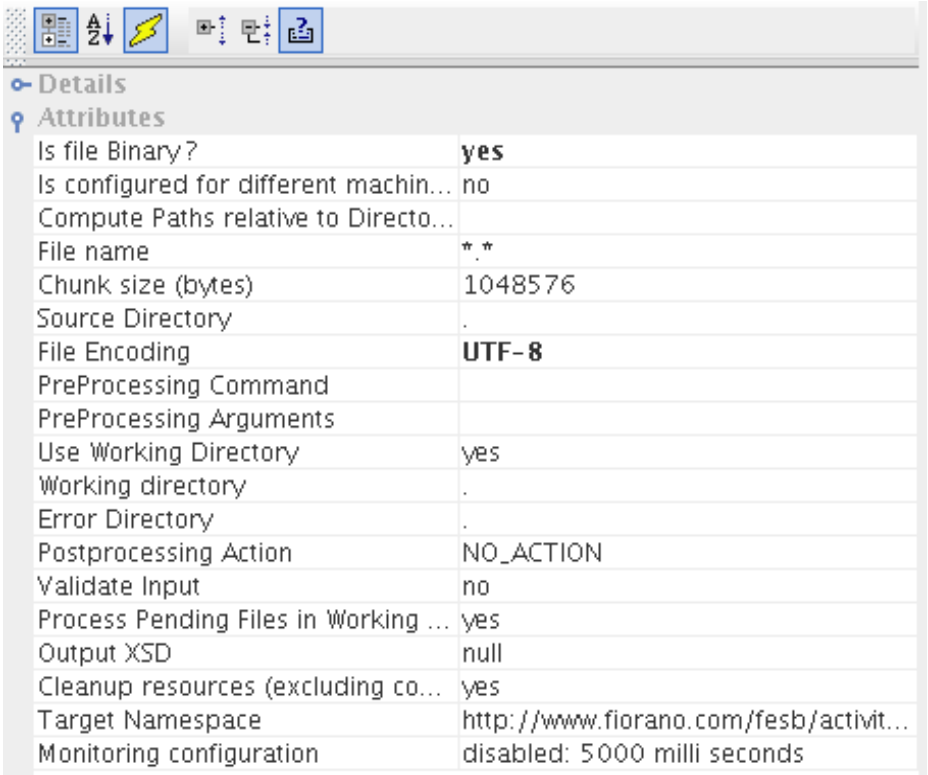


Figure 1: Interaction Configurations

4.5.1.1.1 Attributes

4.5.1.1.1.1 Is file Binary?

The property is used to specify if the input file which is being read is binary or a flat file.

- **Yes:** The contents of the input binary file are read as binary data and are sent to the output port in chunks whose size, in bytes, is specified through the property **Chunk size**.
- **No:** The contents of the target file are read in an unstructured fashion and the content is sent to the output port as a single message.

4.5.1.1.1.2 Chunk size (bytes)

When the input file being read is binary, user can choose to receive the contents read from the file in chunks of binary data at the output ports. The size of these chunks (in bytes) can be specified in this property. When the chunk size is specified as 0 bytes, the whole file is read in a single run.

Note: This property is visible only when property Is file Binary? is set to Yes.

The number of output messages received = ceil (Size of File / Chunk size). The last output message received can be identified by the value of the property COMPLETE in its message headers. Refer Table 1 for information on message headers.

4.5.1.1.1.3 Is configured for different machine?

Specifies whether the Peer Server on which the component is to be launched and Fiorano Studio are running on the same machine or on different machines. This helps the component to determine the type of dialog to be shown while providing the paths of Source Directory, Working Directory and Error Directory. When both the Peer Server and the Studio are running on the same machine, the paths to the above specified directories can be chosen from a file dialog with the directory structure of the current machine. Otherwise, a text editor will be shown where the paths of Source/Working/Error directories need to be specified.

- **Yes:** If the Peer Server on which the component is to be launched and Fiorano Studio are running on different machines.
- **No:** If Peer Server and Fiorano Studio are running on the same machine.

Note: If **Yes** is specified, a text editor is shown to set the path of file and if **No** is specified, a file dialog is shown with directory structure to set the file path as shown in Figures 2 and 3 respectively.

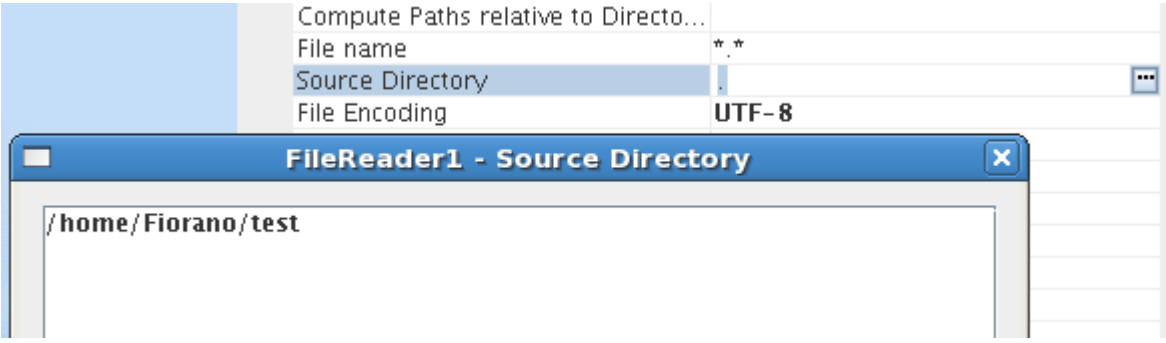


Figure 2: Specifying directory path using Text Editor

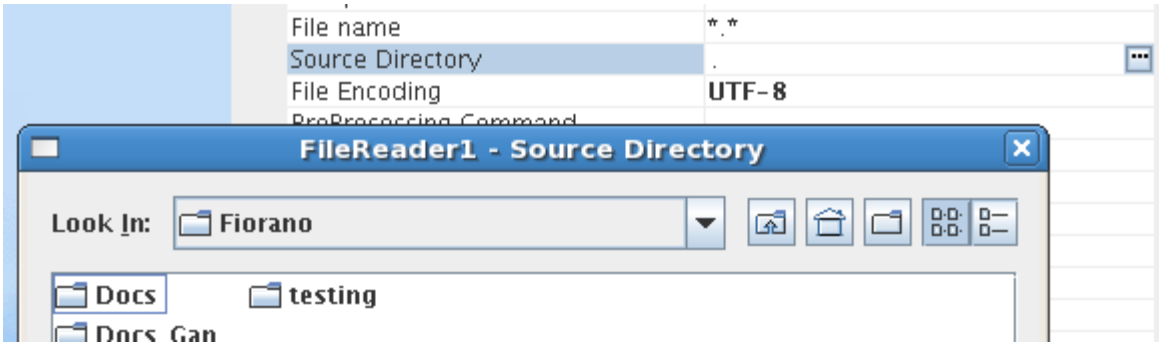


Figure 3: Choosing directory path using File Dialog

4.5.1.1.1.4 Compute Paths relative to Directory

The path of the directory relative to which the paths of Source Directory, Working Directory, Error Directory and Postprocessing Directory are calculated. By default, this points to the FIORANO_HOME directory. If the paths specified for Source/Working/Error/Postprocessing directories are not absolute, their paths are calculated relative to the directory specified here.

Note: If the path specified for Source Directory/Working Directory/Error Directory/Postprocessing directory is absolute, the path specified for Compute Paths relative to directory will not be used in the computation of the path for that particular directory.

4.5.1.1.1.5 File name

The name of the file to be read. A pattern of file names can also be provided using wild character *. Multiple patterns are not allowed. All the files in the Source Directory are checked against this pattern and are suitably processed.

Example: *.txt includes all the files with a .txt extension.

S*.* would include Sample.txt, Service.doc, but not SampleFile

Note:

- Only a single pattern of names can be specified. Multiple formats are not supported while specifying the File name
- When the component is not in scheduling mode, the file name can be specified in the input message to the component and the name specified in the input message overrides the file name (if any) provided during the configuration.
- When a pattern of file names is specified, there is no guarantee that the matching files will be processed in any specific order.

4.5.1.1.1.6 Source Directory

The directory which holds the file(s) to be read has to be specified in Source Directory. All the files in this directory whose names match the pattern specified for the File name property will be processed. The files present in the sub-directories are not considered.

An absolute path or a path relative to the directory specified in the Compute Paths relative to Directory can be provided.

If `Is configured on different machine?` is set to **No**, clicking the ellipses button opens a file dialog as shown in Figure 3, where the directory can be chosen from the file system. Otherwise a text editor pops up where the path of the directory needs to be specified as shown in Figure 2.

The path provided here should point to an existing directory.

Note:

- The directory specified in Compute Paths relative to Directory property will be used in computing the path only if the path specified here is not absolute.
- The FileReader throws an exception if the specified directory does not exist.
- If the component is not configured in scheduling mode, the Source Directory can be specified in the input message to the component and the directory specified during configuration, if any, is overridden by the one provided in the input message.

4.5.1.1.1.7 File Encoding

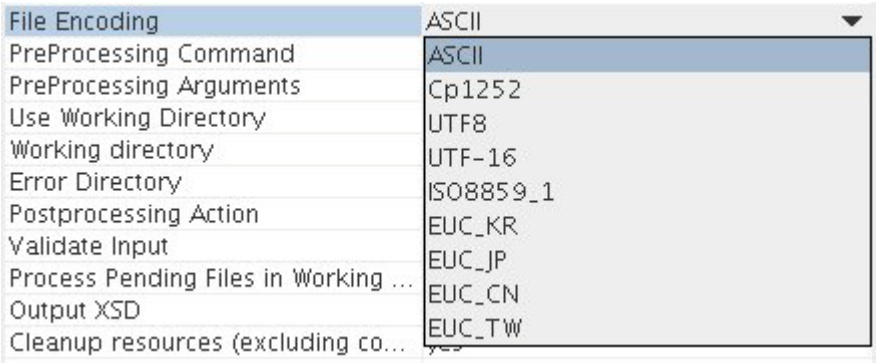


Figure 4: Different type of File Encoding

The encoding to be used while reading the file. Figure 4 shows all the encodings that can be used.

- **ASCII**

  A coding standard used to represent plain text. It is based on English Alphabetical order.

- **Cp1252**

  This is a character encoding of the Latin alphabet

- **UTF8**

  A variable-length character encoding for Unicode

- **UTF-16**

  This too is a variable-length character encoding for Unicode. The encoding form maps each character to a sequence of 16-bit words

- **ISO8859_1**

  ISO 8859-1, more formally cited as ISO/IEC 8859-1 is part 1 of ISO/IEC 8859, a standard character encoding of the Latin alphabet

- **EUC_KR**
- **EUC_JP**
- **EUC_CN**
- **EUC_TW**

EUC_KR, EUC_JP, EUC_CN, EUC_TW are multi-byte character encoding systems used for Korean, Japanese, Simplified Chinese, and Traditional Chinese languages respectively.

Note: Reading UTF files with a byte order mark (BOM) attached to the beginning of the file may not give the desired result.

4.5.1.1.1.8 PreProcessing Command

Script or Command that is to be executed before the processing on file starts. A Command can be entered in the text area provided against this property in the CPS. To provide a script file, the file dialog which is shown by clicking the ellipses button ⋯ can be used.

By default, the component appends the absolute path of the file that is currently taken up for processing to this script / command, that is, the absolute path of the file would be the first argument to this script / command. More arguments for this command could be specified using the property PreProcessing Arguments.

The final command formed by the FileReader would be:

```
<PreProcessing Command> + <Absolute path of the file taken up for processing> +
<PreProcessing Arguments>.
```

4.5.1.1.1.9 PreProcessing Arguments

Arguments that are passed to preprocessing script or command. As mentioned in the PreProcessing Command section, the component, by default, appends the absolute path of the file that is currently taken up for processing to the PreProcessing Command. Any other arguments that need to be passed to the PreProcessing Command can be provided here.

The use of PreProcessing Commands and Arguments is explained in this Sample Scenario

Sample scenario:

Copying all the files present in Error directory to a backup location before the processing on a file starts.

Solution:

A batch file copyerrors.bat with content copy C:\FileReader\ErrorDir %2 is written and is placed in C:\. The path of this batch file is specified for PreProcessing Command. The backup location (C:\ProcessingFailures) is specified as the value for PreProcessing Arguments.

Let, C:\test.txt be the file picked up for processing. With this configuration, the command formed by FileReader would be C:\copyerrors.bat C:\test.txt C:\ProcessingFailures. The copy command executed finally would be copy C:\FileReader\ErrorDir C:\ProcessingFailures which will move all the files present in C:\FileReader\ErrorDir to the backup location C:\ProcessingFailures.

4.5.1.1.1.10 Use Working Directory

Specify if the working directory is to used.

- **Yes:** The files will be moved from the Source Directory to Working directory while FileReader processes the file. The component needs write/modify permissions on the Source Directory to be able to use a working directory. If such permissions are not available, set this property to No.
- **No:** Files won't be moved to the Working directory.

Note: The property Working Directory becomes visible only when Use Working Directory is set to Yes.

4.5.1.1.1.11 Working Directory

The path of the directory which is to be used for intermediate processing of files. If preprocessing actions are specified, the working directory will be used while processing them. If Is configured on different machine? is set to No, clicking the ellipses button will open a file dialog as shown in Figure 3, where the directory can be chosen from the file system. Otherwise a text editor pops up where the path of the directory needs to be specified as shown in Figure 2.

Note:

- This property is visible only when **Use Working Directory** is set to Yes.
- Either an absolute path or a path relative to the directory specified in the **Compute Paths relative to Directory** can be provided.
- If this directory doesn't exist, FileReader creates it while processing the input message.
- FileReader requires write permissions on Working and Error Directories

4.5.1.1.1.12 Error Directory

Path of the directory which should hold the files whose processing has not been successful.

If Is configured on different machine? is set to No, clicking the ellipses button will open a file dialog as shown in Figure 3, where the directory can be chosen from the file system. Otherwise a text editor pops up where the path of the directory needs to be specified as shown in Figure 2.

Note:

- Either an absolute path or a path relative to the directory specified in the Compute Paths relative to Directory can be provided.
- If this directory does not exist, FileReader creates it while processing the input message.
- FileReader requires write permissions on Working and Error Directories.

4.5.1.1.1.13 Postprocessing Action

Action to be taken on the file after it is read successfully. Figure 5 shows all the Postprocessing Actions that are allowed.

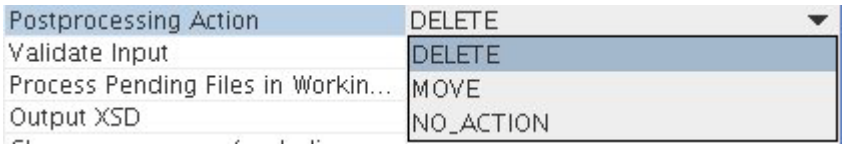


Figure 5: Postprocessing Actions

- **DELETE**

  Delete the file after reading it successfully.

- **MOVE**

Move the file to a different location (specified by the property **Postprocessing Directory**)

Note: When MOVE is selected as the Postprocessing Action, four other properties become visible (Shown in the Figure 6)

Postprocessing Action	MOVE
Postprocessing Directory	.
Append timestamp ?	yes
Timestamp format	ddMMyyyy_HHmmssSSS
Append counter ?	no

Figure 6: MOVE action

- **NO_ACTION**

  Take no action on the file.

4.5.1.1.1.14 Postprocessing Directory

The directory to which files are to be moved when they are read successfully, when MOVE is selected as the Postprocessing Action.

If Is configured on different machine? is set to No, clicking the ellipses(...) button will open a file dialog, as shown in Figure 3, where the directory can be chosen from the file system. Otherwise a text editor pops up where the path of the directory needs to be typed in as shown in Figure 2.

Note: This option is visible only when MOVE is selected as the Postprocessing Action.

4.5.1.1.1.15 Append timestamp?

Specifies if a time stamp has to be appended to the file names after they have been moved to the Postprocessing Directory.

- **Yes:** FileReader adds a time stamp whose format is provided through the Timestamp format property and a counter (if Append counter? is set to Yes).
- **No:** No timestamp is added to the files that have been moved to the Postprocessing directory.

Note: This option is visible only when MOVE is selected as the Postprocessing Action.

4.5.1.1.1.16 Timestamp format

The format of the time stamp to be appended to the file name can be specified here. The descriptions of the symbols that can be used in the time stamp formats are depicted in Figure 7.

Symbol	Meaning	Presentation	Example
G	era designator	(Text)	AD
y	year	(Number)	1996
M	month in year	(Text & Number)	July & 07
d	day in month	(Number)	10
h	hour in am/pm (1~12)	(Number)	12
H	hour in day (0~23)	(Number)	0
m	minute in hour	(Number)	30
s	second in minute	(Number)	55
S	millisecond	(Number)	978
E	day in week	(Text)	Tuesday
D	day in year	(Number)	189
F	day of week in month	(Number)	2 (2nd Wed in July)
w	week in year	(Number)	27
W	week in month	(Number)	2
a	am/pm marker	(Text)	PM
k	hour in day (1~24)	(Number)	24
K	hour in am/pm (0~11)	(Number)	0
z	time zone	(Text)	Pacific Standard Time
'	escape for text	(Delimiter)	
''	single quote	(Literal)	'

Figure 7: Symbols used in Timestamp format

Example: ddMMyyyy_HHmm

Note:

- This property is visible ONLY when the **Append timestamp** is set to Yes.
- Avoid using slashes ('/' or '\') in the time stamp format as they can be mis-interpreted as File Separators and can lead to confusion.
- Special characters that are not allowed in file names should not be included in the timestamp format (This can be platform specific).

4.5.1.1.1.17 Append counter?

- **Yes:** A counter is appended to the file name of each processed file in addition to the time stamp. Appending counter to file names ensures that no two files in the Postprocessing directory will have same name. The name of the file would look like `<filename>_<time stamp>_<counter>`.
- **No:** No counter is added to the files that have been moved to the Postprocessing directory.

Note: This property is visible ONLY when the Append timestamp is set to Yes.

4.5.1.1.1.18 Validate Input

If set to Yes, the input request sent to the FileReader is validated against the input port XSD of the component.

4.5.1.1.1.19 Process Pending files in Working Directory

Specify if the pending files present in the Working directory are to be processed. When this property is enabled, for every input request to read a file from a specific directory, the file is searched in Working directory in addition to the specified directory. If the component is in scheduling mode, enabling this property processes the files in the Working directory as well.

Note:

- If a file with same name exists in both the Source directory and Working directory, the file in the Source directory is processed.
- Setting this property to Yes will not have any effect if **Use Working Directory** is NO.

4.5.1.1.1.20 Output XSD

This property is used to set the schema of the output message. If the file content is expected to be an XML, setting its schema on the output port using the Output XSD can be useful for applying transformations on the output message. The XSD can be provided using the Schema Editor as shown in the Figure 8.

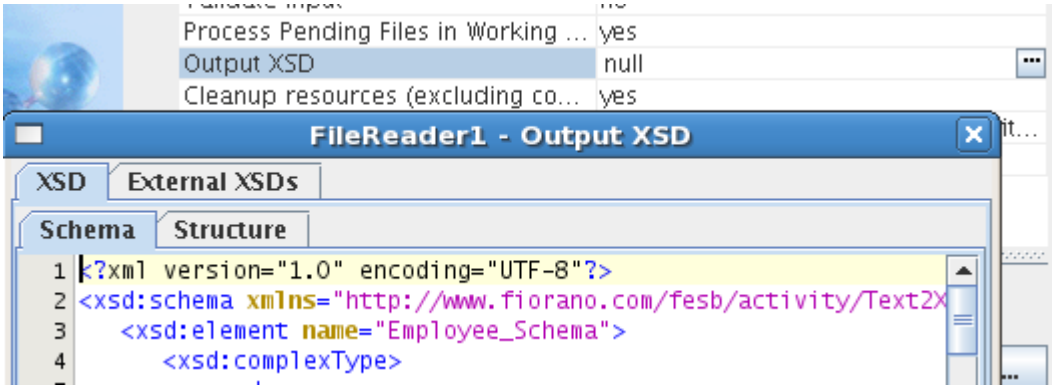


Figure 8: Schema editor to provide Output XSD

4.5.1.1.1.21 Header properties

Table 1 shows the descriptions of header properties set by the component on the output message when Flat/Binary files are processed.

Type of the file processed	Header property	Description
Flat/Binary	FileName	Name of the file being read.
	FilePath	Path of the directory which holds the source file.
	Size	The size of the file being read.

Type of the file processed	Header property	Description
	START_EVENT	An output message with this property set to true determines that the message is the first record in the set of responses generated for an input message. Note: This property appears only on the first record in the set of responses.
	CLOSE_EVENT	An output message with this property set to true determines that the message is the last record in the set of responses generated for an input message. Note: This property appears only on the last record in the set of responses.
	RECORD_INDEX	A value n for this property indicates that this is the nth response generated for an input message.
Flat	FullName	Absolute path of the processed file.
	ReadAccess	Determines if the processed file is readable.
	WriteAccess	Determines if the processed file is writable.
	Type	File / Directory.
Binary	NEW	An output message with this property set to true determines that this is the first chunk of the binary file being read.
	COMPLETE	An output message with this property set to true determines that this is the last chunk of the binary file being read.
	START_INDEX	Determines the offset of first byte of the current chunk read.
	END_INDEX	Determines the offset of last byte of the current chunk read.

Table 1: Header Properties

4.5.1.1.2 Using Named Configurations

Changes are done in eStudio wrt to Named Configurations. With the implementation of Named Configurations, Interaction Configurations panel is changed as shown in Figure below.

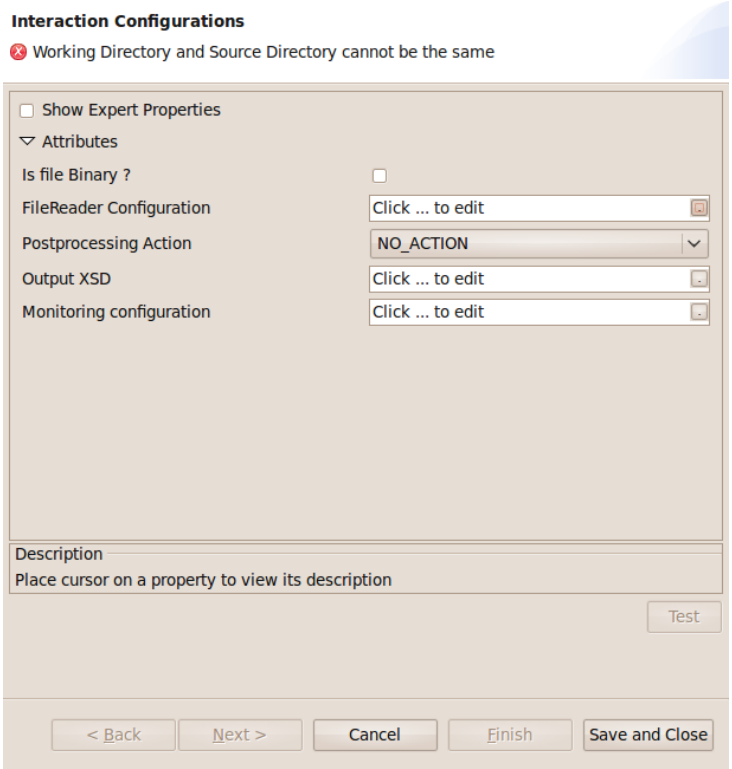


Figure: Modified Interaction Configurations panel

Some properties are grouped as one property in the modified configuration panel. Is configured for different machine , Compute Paths relative to Directory, File Name, Source Directory, Use Working Directory, Working Directory, Process Pending Files in Working Dir and Error Directory which were present previously in the Interaction Configurations panel are grouped under the name File Reader Configuration.

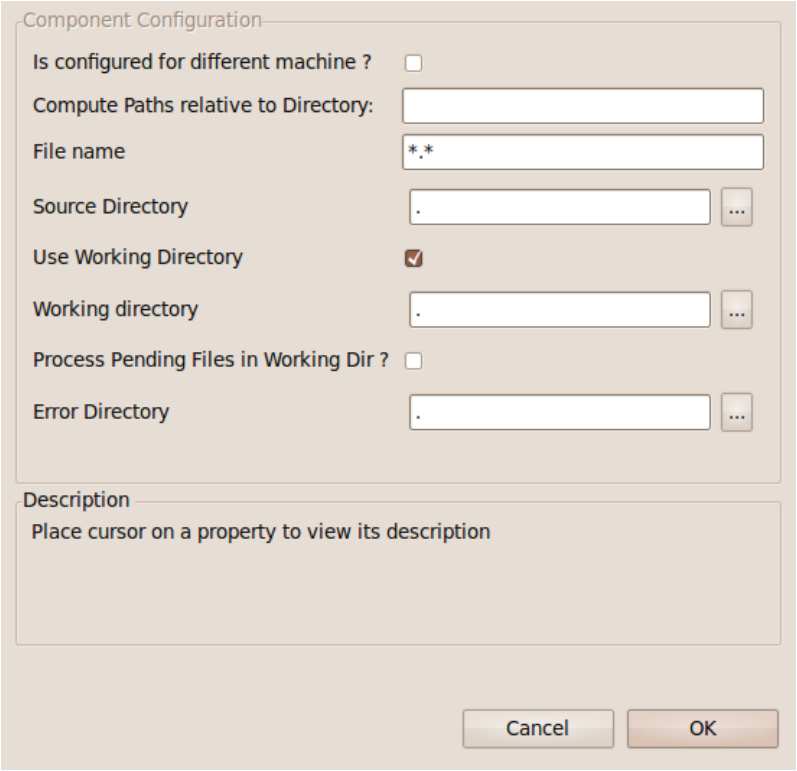


Figure: File Reader Configuration panel

4.5.1.2 Input and Output

4.5.1.2.1 Input

When FileReader is not in scheduling mode, messages can be sent onto the input port of the component specifying the file to be read and the location of the file. The schema of the input XML message is shown in Figure 10.

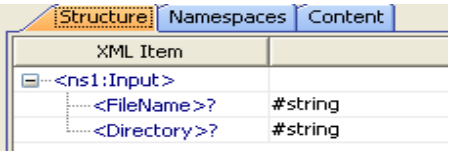


Figure 9: Schema of the input message

- FileName is the name of the file which is to be read.
- Directory is the location of the file.

Note: If the values for FileName and Directory are not specified in the input message, the values configured for the CPS properties File Name and Source Directory are used.

4.5.1.2.2 Output

The output schema depends on the configuration of property Output XSD. Schema provided for this property is directly set as schema on output port. For more information, please refer to Output XSD in *Interaction Configuration* section.

4.5.1.3 Testing the Interaction Configurations

Interaction configurations can be tested from the CPS by clicking the **Test** button. Figure 10 and Figure 11 show the sample input and the corresponding output respectively.

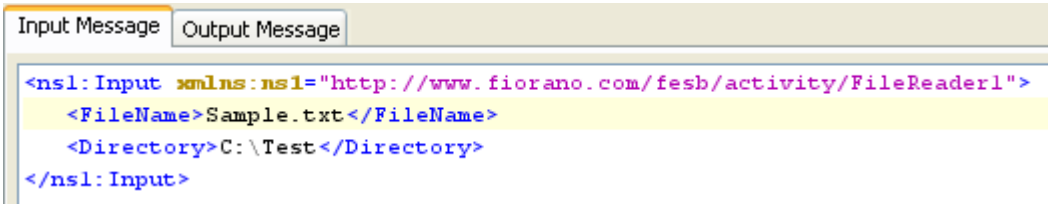


Figure 10: Sample input

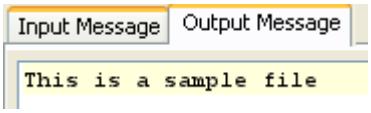


Figure 11: Output produced for sample input shown in Figure 10

4.5.1.4 Functional Demonstration

4.5.1.4.1 Scenario 1

Reading simple text files and displaying the contents.

Configure the FileReader as described in *Configuration and Testing* section and use feeder and display component to send sample input and check the response respectively.

Figure 12: Demonstrating scenario 1 with sample input and output

Input Message:

```
<ns1:Input xmlns:ns1="http://www.fiorano.com/fesb/activity/FileReader1">
 <FileName>Input.txt</FileName>
 <Directory>C:\Read</Directory>
</ns1:Input>
```

Output Message:

This is a sample file (Contents of the input file).

4.5.1.5 Use Case Scenario

In a revenue control packet scenario transaction files are read and then transformed.

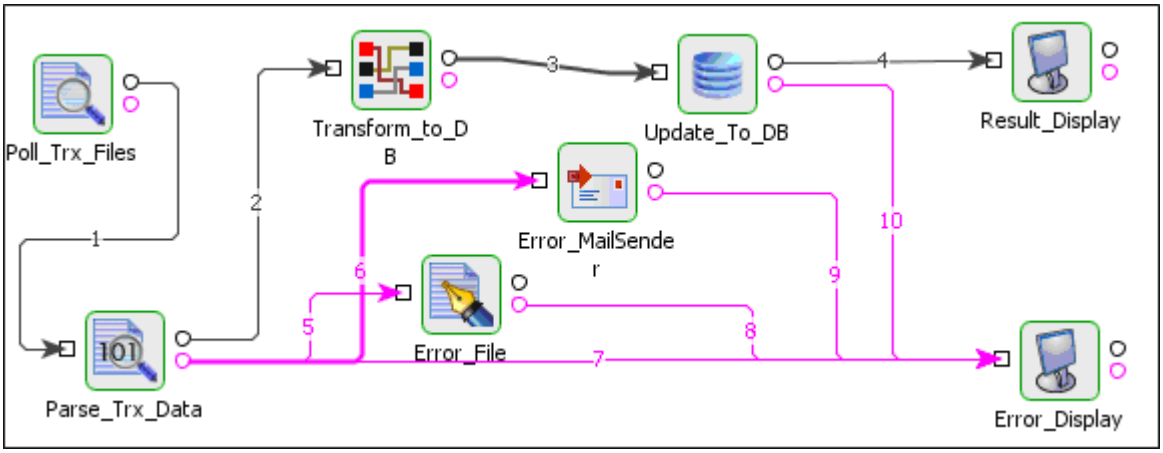


Figure 13: Revenue Control Packet Scenario

The event process demonstrating this scenario is bundled with the installer.

Documentation of the scenario and instructions to run the flow can be found in the Help tab of flow when open in Studio.

4.5.1.6 Useful Tips

Best Practices to read a file from a network drive using the FileReader component:

- To access a file present in a network drive, FileReader component needs full permissions on that directory. Please enable the option "Allow Network Users to change my files" while sharing a directory.
- If you do not have permissions to change the files, then in the File Reader Custom Property Sheet you need to set "Use Working Directory" to "No".
- In case running peer server as Windows/Linux service, it is possible that the network drive is not mounted by the time the peer server has started the file components. In such a case, making the peer service dependent on the service that is mounting the Network Drive will help.

4.5.2 File Writer

The FileWriter component writes the received data from its input port to the specified output file. The received data can either be plain text or binary data.

File Writer is capable of handling:

- **Text files**

  Text content from input message is written to the configured file in text format.

- **Binary Files**

  Binary content from the input message is written to the configured file in binary format.

File Writer uses core Java APIs to write the files.

Points to note

- The component runs on the Peer Server and therefore the file paths and directories mentioned in the CPS should be valid on the machine where the Peer Server is running. If the component fails over to another peer, ensure that the machine on which the secondary Peer Server is running does have the same path available.
- The received XML equivalent of the unstructured plain text needs to be transformed to its original format using the XML -> Flat component.
- FileWriter closes the file and moves it to the target directory on receiving a message having property **COMPLETE** whose value is set to true. This is useful even when you use **Append if Exists** output mode for the FileWriter component and you would like to close the file on the occurrence of this event.
- The FileWriter component uses the JMS Header filename to get the file name from the input message.

4.5.2.1 Interaction Configurations

Business logic configuration details are configured in the second panel, **Interaction Configurations**. Figure 1 illustrates the panel with expert properties view enabled.

Attributes	
Is input Binary ?	**yes**
Is Input Base64 Encoded?	no
Compute Paths relative to Directo...	
Is Configured for Different Machi...	no
File name	hello.txt
Use file details from message he...	no
Target Directory	/home/ramesh/target
Append timestamp ?	no
Append counter ?	no
File Encoding	**UTF-8**
Working directory	/home/ramesh/work
Error Directory	/home/ramesh/err
PostProcessing command	
PostProcessing Arguments	
Time interval for PostProcessing ...	**-1**
Return data on Output ?	no
Validate Input	no
Target Namespace	http://www.fiorano.com/fesb/activit...
Monitoring configuration	disabled: 5000 milli seconds

Figure 1: Interaction Configurations.

4.5.2.1.1 Attributes

4.5.2.1.1.1 Is input Binary?

Specifies whether the input received on the input port is binary.

- **yes** - Writes the data as bytes into the target file.
- **no** - Writes text to the specified target file in an unstructured fashion.

Note: The Output message sent to the output port depends on the configuration provided here. When set to **y**es, it hides the property **Output Mode**.

4.5.2.1.1.2 Is Input Base64 Encoded ?

Option is visible only when the input is selected as binary. Specifies whether the input text received on the input port is Base64 encoded or not.

- **yes** - Writes the data decoded from base64 encoded input text as bytes into the target file.
- **no** - Writes input text to the specified target file with no decoding similar to simple binary data processing.

Note: Details regarding whether the input text is Base64 encoded or not can be overridden with java String property named "isBase64Encoded" in message header while sending input to the Filewriter. If value of "isBase64Encoded" is given as 'true' or 'yes', component will treat the input text as Base64 encoded and assign the corresponding decoding processor.

If value of "isBase64Encoded" is given as 'false' or 'no', input text is considered as normal binary data with no decoding steps involved. The value of "isBase64Encoded" property will be given priority over the value set in cps and if no value is set for the string property variable, then Base64 encoding details provided in cps will be taken into consideration.

4.5.2.1.1.3 Compute Paths relative to Directory

The path of the directory relative to which the paths of Target Directory, Working Directory, and Error Directory are computed. By default, this points to the *FIORANO_HOME* directory. If the paths specified for Target/Working/Error Directories are not absolute, their paths are calculated relative to the directory specified here.

Note: If the path specified for Target Directory/Working Directory/Error Directory is absolute, the path specified for Compute Paths relative to Directory will not be used in the computation of the directory paths.

4.5.2.1.1.4 Is Configured for Different Machine?

Specifies whether the Peer Server on which the component is to be launched and Fiorano Studio are running on the same machine or on different machines. This helps the component to determine the type of dialog to be shown while providing the paths of Source Directory, Working Directory, and Error Directory. When both the Peer Servers and the Fiorano Studio is running on the same machine, the paths to the above specified directories can be chosen from a file dialog with the directory structure of the current machine as shown in Figure 2. Otherwise, a text editor will be shown where the paths of Source/Working/Error directories need to be typed in as shown in Figure 3.

- **yes** -

If the Peer Server on which the component is to be launched and Fiorano Studio are running on different machines.

- **no** -

  If the Peer Server and Fiorano Studio are on the same machine.

  When this property is set to no, paths of directories can be chosen from the file dialog. When the property is set to yes, the paths of directories should be manually specified in the Text Editor as shown in Figure 2 and Figure 3.

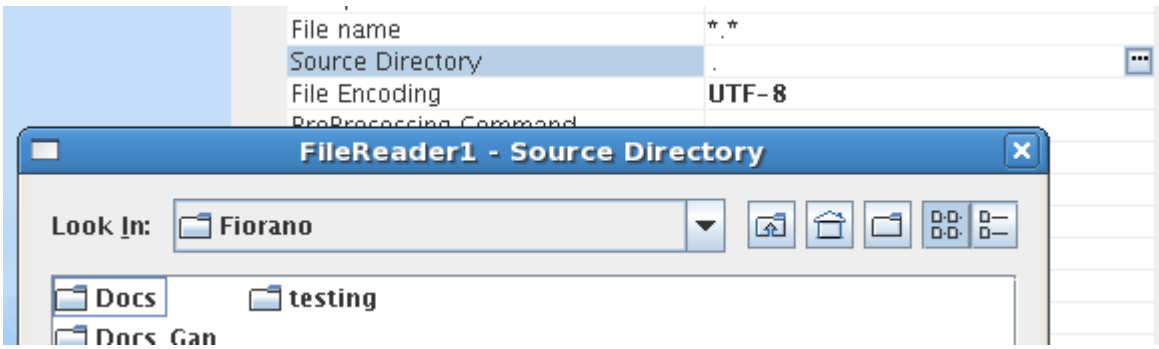


Figure 2: Choosing directory path using File Dialog

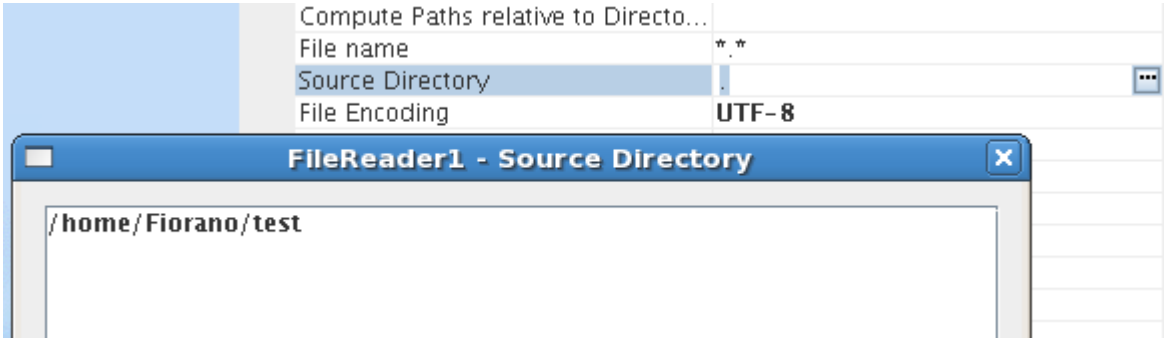


Figure 3: Specifying directory path using Text Editor

4.5.2.1.1.5 File name

The name of the file to which the input data has to be written is specified here. If the file with this name does not exist, a new file is created.

Note:

- If the property Use file details from message headers is set to Yes, then this name would be used only if the FileName property is not present in the header of the incoming message.
- When no file name is specified both in the CPS and the message header, FileWriter throws an exception.

4.5.2.1.1.6 Use file details from message headers

Specify if the file details are to be taken from the header of the input message.

- **yes** -

The name of the target file would be the value of the header property FileName present on the input message and the target directory would be the value of the header property directory.

- **no** -

  When the property is set to no, the value specified for the CPS property **File name** is used.

  Note: If **Use file details from message headers** is yes and if the property FileName is not found in the message headers, the **File name** from the CPS is used. Similarly, the directory specified in the CPS (Target Directory) is used as the target directory if the property directory is not found in the message headers.

4.5.2.1.1.7 Target Directory

The path of the directory where the target file (with the data received on the input port) is created. An absolute path of the target directory or a path relative to the directory specified in **Compute Paths relative to Directory** can be specified here. While specifying a directory as the Target Directory, make sure that you have permissions to create/modify files present in this directory.

If **Is configured for different machine?** is set to **no**, clicking the ellipses button will open a file dialog, as shown in Figure 21, where the directory can be chosen from the file system. Otherwise a text editor pops up where the path of the directory needs to be specified as shown in Figure 22.

Note:

- If this folder does not exist, FileWriter creates it while processing the input message.
- If the property **Use file details from message headers** is set to yes, the directory specified here would be used only if the directory property is not present in the incoming message headers. FileWriter throws an exception if no directory is specified both in the CPS and in the message headers.

4.5.2.1.1.8 Append timestamp?

Specifies whether a timestamp is to be appended to the name of the target file.

Note: When **Is input binary?** is set to No, this property appears only when **Output mode** is set to New file for each message.

4.5.2.1.1.9 Timestamp format

Specify the format in which the timestamp is to be appended to the name of the target file. Figure 23 shows the descriptions for the symbols that could be used in the Timestamp format.

Symbol	Meaning	Presentation	Example
G	era designator	(Text)	AD
y	year	(Number)	1996
M	month in year	(Text & Number)	July & 07
d	day in month	(Number)	10
h	hour in am/pm (1~12)	(Number)	12
H	hour in day (0~23)	(Number)	0
m	minute in hour	(Number)	30
s	second in minute	(Number)	55
S	millisecond	(Number)	978
E	day in week	(Text)	Tuesday
D	day in year	(Number)	189
F	day of week in month	(Number)	2 (2nd Wed in July)
w	week in year	(Number)	27
W	week in month	(Number)	2
a	am/pm marker	(Text)	PM
k	hour in day (1~24)	(Number)	24
K	hour in am/pm (0~11)	(Number)	0
z	time zone	(Text)	Pacific Standard Time
'	escape for text	(Delimiter)	
''	single quote	(Literal)	'

Figure 4: Symbols used in Timestamp format.

Example: If Test.txt is the filename to which the data is to be written, the target file created could be Test_02042008_183950765.txt when the default format (ddMMyyyy_HHmmssSSS) is used.

Note: This property is visible only when **Append timestamp?** is set to Yes.

4.5.2.1.1.10 Append Counter?

- **yes** -

  Appends a counter to the name of each file processed by the FileWriter in addition to the timestamp. The name of the file would look like <filename>_<time stamp>_<counter>. Appending counter to file names ensures that no two files created can have same name.

- **no** -

  No counter is appended to the target file processed by the component FileWriter.

  Example: If Test.txt is the filename to which the data is to be written, the target file created could be Test_02042008_183950765_0.txt when **Append Counter?** is set to **yes**.

  Note: This property is visible only when **Append timestamp?** is set to yes.

4.5.2.1.1.11 Output Mode

Specify the way in which the output is to be created with the received data. Figure 5 shows the output modes that can be used.

Note: This property is visible only when **Is input Binary?** is set to No.

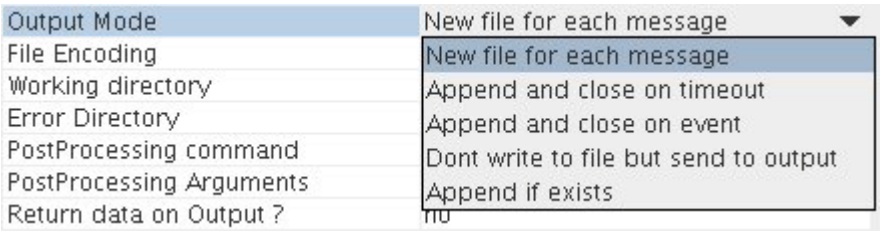


Figure 5: Output Mode

- **New file for each message**

  This option tells the FileWriter to create a new file for each input message that has been processed. When this output mode is being used, timestamp and counter can be appended to the filename using the properties **Append timestamp?** and **Append counter?**. The **Append timestamp?** option will be visible only when this option is selected as the **Output Mode**.

  Note: If the same File name is specified for two different input messages and no time stamp or counter is appended, the existing file is overwritten.

- **Append and close on timeout**

  This output mode is used for appending the content of all messages received till timeout, to a single file. During this time interval, the file is present in Working directory. When the timeout occurs, the file is closed and is moved to Target directory. Selecting this option will show the property **File Timeout (min)** which can be used to set the timeout. The property **File Timeou**t is visible only when Output Mode is set to **Append and close on timeout.**

  Note:

  - Only one file holds the content of all messages received during the configured time interval. The name of this file would be the value specified for the Header property FileName on the first message received in this time interval, if **Use file details from message header?** is set to yes. If not, the file name would be the value of the property **File Name** specified in CPS.
  - If, for any reason, the component is stopped/relaunched before the specified timeout, further messages will be processed into a new file.

- **Append and close on event**

  This output mode is used for appending the content of all received messages to a single file until CLOSE_EVENT occurs. When a message with the header property CLOSE_EVENT set to true is received, the content of this message is appended to the file, the file is closed and is moved from Working directory to Target directory.

  Note:

  - The file will continue to be in the working directory till it receives a closing event.

- Only one file holds the content of all messages received until CLOSE_EVENT occurs. The name of this file would be the value specified for the Header property FileName on the first message received, if **Use file details from message header?** is set to yes. Otherwise, the file name would be the value of **File Name** specified in CPS.

- **Do not write to file but send to output**

  The received data is sent to the output port instead of writing into a file. Figure 6 and Figure 7 show the snapshots of the text and binary data sent to output port.

  Note: If **Is Input Binary?** has to be set to yes for treating the input as binary.

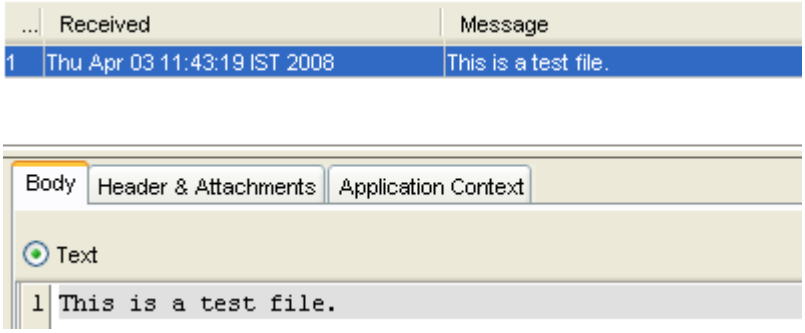


Figure 6: Text data received on output port.

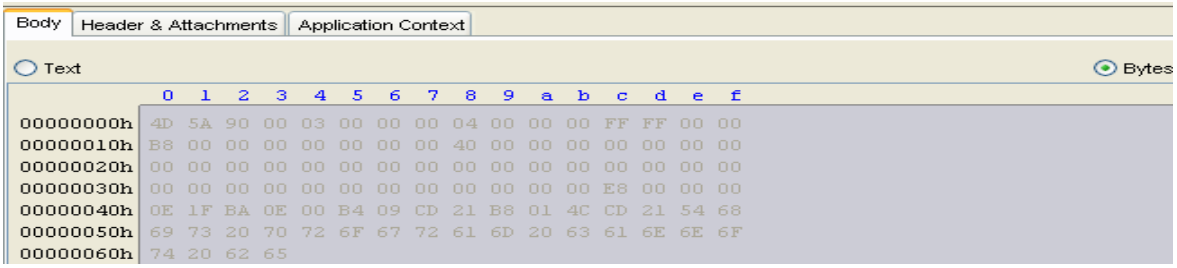


Figure 7: Binary data received on output port.

Note: Selecting Do not write to file but send to output will hide the property **Return data on Output?** Since, the output will definitely be sent to the output.

- **Append if exists**

  If the file to which the data is to be written already exists, data is appended to the content in the existing file. The file being processed will be moved to the Working directory and content received in the input message will be appended to the existing file until FileWriter receives a message with a property COMPLETE set to true. (JMS function setStringProperty() can be used to set the value of a JMS property on a message).

  Note: Until FileWriter receives a message with this property COMPLETE set to true, it keeps the file in Working directory.

4.5.2.1.1.12 File Timeout (min)

The time for which the component has to wait before closing and moving the file from Working directory to the Target directory is specified using this property. This is visible only when Append and close on timeout is selected as the **Output Mode**.

Note: File timeout cannot be a fraction. Hence, the minimum timeout that can be specified is 1 minute.

4.5.2.1.1.13 File Encoding

The encoding to be used while writing the file. The following encodings can be used. Figure 8 shows all the encodings that can be used.

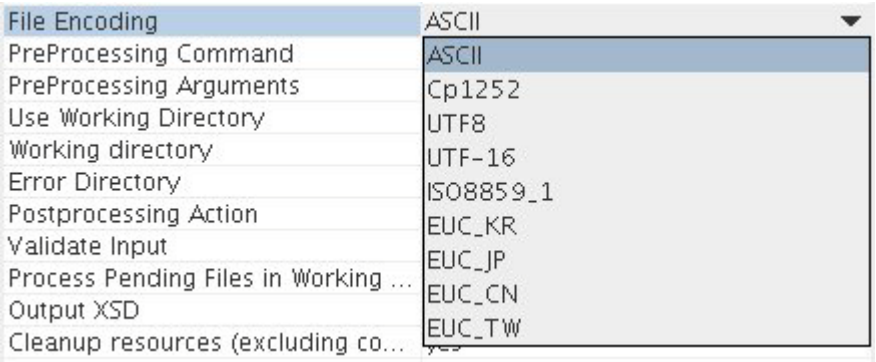


Figure 8: Different types of File Encoding

- **ASCII -**

  A coding standard used to represent plain text. It is based on the order of English Alphabet

- **Cp1252 -**

  This is a character encoding of the Latin alphabet.

- **UTF8 -**

  A variable-length character encoding for Unicode.

- **UTF-16 -**

  This is a variable-length character encoding for Unicode. The encoding form maps each character to a sequence of 16-bit words.

- **ISO8859_1 -**

  ISO 8859-1, more formally cited as ISO/IEC 8859-1 is part 1 of ISO/IEC 8859, a standard character encoding of the Latin alphabet.

- **EUC_KR**
- **EUC_JP**
- **EUC_CN**
- **EUC_TW**

  EUC_KR, EUC_JP, EUC_CN, EUC_TW are multi-byte character encoding systems used for Korean, Japanese, Simplified Chinese, and Traditional Chinese languages respectively.

4.5.2.1.1.14 Working directory

The path of the directory which should hold the files during intermediate processing. Either an absolute path or a path relative to the directory specified in the **Compute Paths relative to Directory** can be provided.

If **Is configured for different machine?** is set to no, clicking the ellipses button will open a file dialog, as shown in Figure 2, where the directory can be chosen from the file system. Otherwise, a text editor pops up where the path of the directory needs to be typed in. Shown in Figure 3.

Note:

- When **Output mode** is Append and close on timeout or Append and close on event, Working directory holds the files until the timeout or CLOSE_EVENT occurs.
- If this directory does not exist, FileWriter creates it while processing the input message.
- FileWriter requires write permissions on Working and Error directories.

4.5.2.1.1.15 Error directory

Path of the directory which should hold the files for which the processing has not been successful.

Note:

- Either an absolute path or a path relative to the directory specified in the **Compute Paths relative to Directory** can be provided.
- If this directory does not exist, FileWriter creates it while processing the input message.
- For any file, when the processing is successfully finished, FileWriter moves the file from Working directory to Target directory. If the processing is not successful, the file gets retained in the Working directory itself. When FileWriter receives data which is to be written to the file that already exists in the Working directory, FileWriter moves the existing file in the Working directory to Error directory and then creates a new file with the input data. Every file that moves to the Error directory will have appended to its name, the time (in milliseconds) at which it is moved to the Error directory.
- FileWriter requires write permissions on Working and Error Directories.

4.5.2.1.1.16 PostProcessing Command

Script or a Command that is to be executed after the file processing is finished. A command can be specified by simply typing it in the text area provided against this property. To provide a script file, the file dialog which is shown by clicking the ellipses button can be used.

By default, the component appends the absolute path of the target file to this script/command that is. the absolute path of the target file would be the first argument to this script/command. More arguments for this command could be specified using the property **PostProcessing Arguments**. The final command formed by the FileWriter would be <PostProcessing Command> + <Absolute path of the target file> + <PostProcessing Arguments>.

4.5.2.1.1.17 PostProcessing Arguments

Arguments that are passed to postprocessing script or command.

4.5.2.1.2 Using Named Configurations

Changes are done in eStudio wrt to Named Configurations. With the implementation of Named Configurations, Interaction Configurations panel is changed as shown in Figure below.

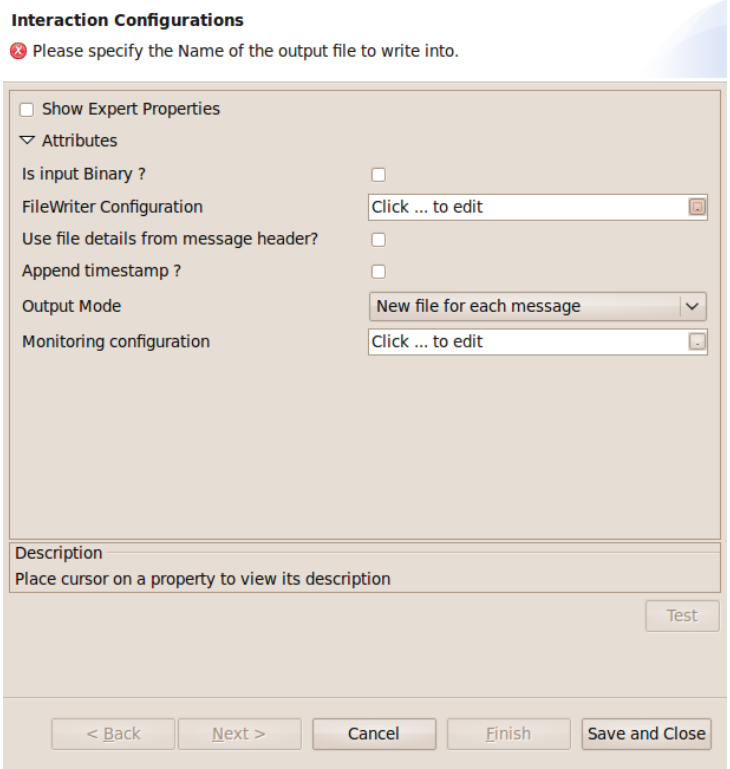


Figure: Modified Interaction Configurations panel

Some properties are grouped as one property in the modified configuration panel. Is configured for different machine, Compute Paths relative to Directory, File Name, Target Directory, Working Directory and Error Directory which were present previously in the Interaction Configurations panel are grouped under the name File Writer Configuration.

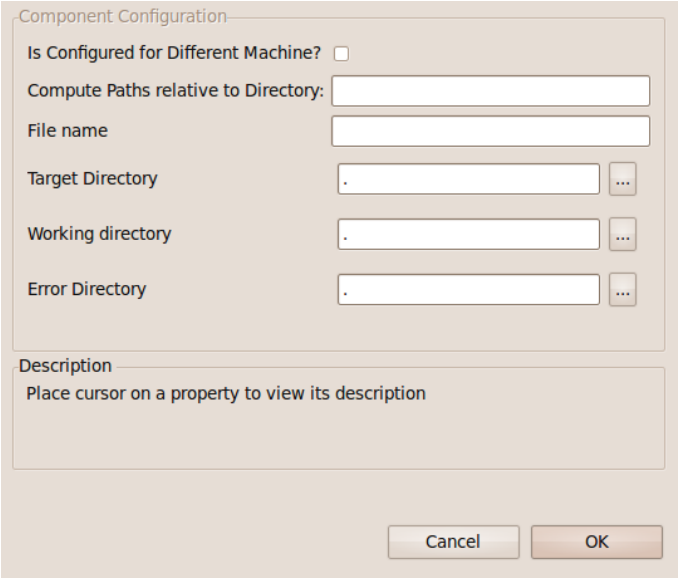


Figure: File Writer Configuration panel

4.5.2.2 Sample Scenario

Copying all the files present in Error directory to a backup location after the file is processed.

Solution:

A batch file copyerrors.bat with content copy C:\FileWriter\ErrorDir %2 is written and is placed in C:\. The path of this batch file is specified for the property **PostProcessing Command**. The backup location (C:\ProcessingFailures) is specified as the value for **PostProcessing Arguments.**

Let C:\FileWriter\TargetDir\test.txt be the target file. With this configuration, the command formed by FileReader would be C:\copyerrors.bat C:\FileWriter\TargetDir\test.txt C:\ProcessingFailures. The copy command executed finally would be copy C:\FileWriter\ErrorDir C:\ProcessingFailures which moves all the files in C:\FileWriter\ErrorDir to the backup location C:\ProcessingFailures.

Return data on Output?

Determines if the data written to the file is to be sent onto the output port instead of sending out an XML message containing the details of the file to which the content has been written. Figure 6 and Figure 7 show sample views of the text and binary data sent on the output port. As shown in the figures, setting this property to Yes will send the data in the output message. However, the file details can still be fetched from the message headers of the output message. Refer Table 1 for the description of header properties.

Note: This property is not visible when **Output mode** is set to Don't write to file but send to output.

Header Properties

Table 1 shows the descriptions of header properties set by the component on the output message when binary/flat content is processed.

Type of data received on input port	**Header Property**	**Description**
Flat/Binary	FileName	Name of the file to which the data is being written.
	FilePath	Path of the directory which holds the destination file.
	FullName	Absolute path of the destination file. For a binary file, this property appears only on the message which represents the last chunk of the file.
	Size	Determines the final size of the destination file.
	ReadAccess/ WriteAccess	Determines if the destination file is readable/writable. For a binary file, these properties appear only on the message which represents the last chunk of the file.
	START_EVENT	Value of the header property START_EVENT present on the input message to the component. An input message with this property set indicates that this is the first message whose content is to be written to the file represented by the property FileName.
	RECORD_INDEX	Value of the header property RECORD_INDEX present on the input message to the component. A value 'n' for this property indicates that this is the nth message whose content is to be appended to the file represented by the property FileName
	CLOSE_EVENT	Value of the header property CLOSE_EVENT present on the input message to the component. An input message with this property set indicates that this is the last message whose content is to be appended to the file represented by the property FileName.
Flat	Type	File/Directory

Type of data received on input port	Header Property	Description
Binary	NEW	Value of the header property NEW present on the input message to the component. An input message with this property set to true indicates that this message holds the first chunk of data to be written to the file represented by the property FileName
	COMPLETE	Value of the header property COMPLETE present on the input message to the component. An input message with this property set to true indicates that this message holds the last chunk of data to be appended to the file represented by the property FileName
	START_INDEX	Value of the header property START_INDEX present on the input message to the component. This value represents the offset of first byte of the chunk which has been currently written to the destination file.
	END_INDEX	Value of the header property END_INDEX present on the input message to the component. This value represents the offset of last byte of the chunk which has been currently written to the destination file.

Table 1: Header Properties

4.5.2.3 Input and Output

4.5.2.3.1 Input

No schema is set on the input port. If any XML message has to be written into a file then the XML message is directly sent as input message.

4.5.2.3.2 Output

When a file is written successfully, FileWriter sends out the file information onto the output port. Figure 9 shows the schema of the output XML message. Table 2 shows the description of the schema elements.

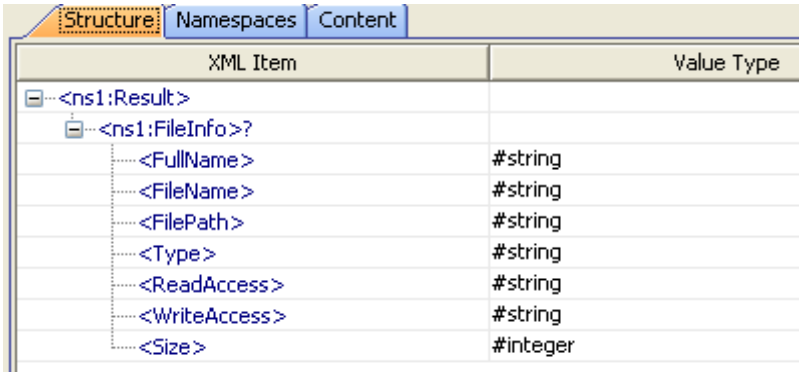


Figure 9: Schema on the output port

Schema Element	**Description**
FullName	Path of the file to which the data has been written
FileName	Name of the file to which the data has been written
FilePath	Path of the directory which holds the output file
Type	The type of output
ReadAccess	Specifies if the output file has read access
WriteAccess	Specifies if the output file has write access
Size	Size of the output file in bytes

Table 2: Description of Output schema elements

4.5.2.4 Testing the Interaction Configurations

Interaction configurations can be tested from the CPS by clicking the Test button. Figure 10 and Figure 11 shows the sample input and the corresponding output message respectively.

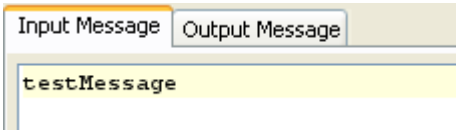


Figure 10: Sample input

Input Message | Output Message

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:Result xmlns:ns1="http://www.fiorano.com/fesb/activity/FileWriter1">
   <ns1:FileInfo>
      <FullName>C:\FileWriter\TargetDir\Test_03042008_171633859_0.txt</FullName>
      <FileName>Test_03042008_171633859_0.txt</FileName>
      <FilePath>C:\FileWriter\TargetDir</FilePath>
      <Type>File</Type>
      <ReadAccess>true</ReadAccess>
      <WriteAccess>true</WriteAccess>
      <Size>11</Size>
   </ns1:FileInfo>
</ns1:Result>
```

Figure 11: Output generated for input shown in Figure 10.

4.5.2.5 Functional Demonstration

4.5.2.5.1 Scenario 1

Writing the input message into to a file and displaying the response contents.

Configure the FileWriter as shown in Figure 12.

Attributes	
Is input Binary?	no
Compute Paths relative to Directo...	
Is Configured for Different Machi...	no
File name	out.txt
Use file details from message he...	no
Target Directory	/home/geetha/backup/Target
Append timestamp ?	**yes**
Timestamp format	ddMMyyyy_HHmmssSSS
Append counter ?	**yes**
Output Mode	New file for each message
File Encoding	**UTF-8**
Working directory	/home/geetha/backup/working
Error Directory	/home/geetha/backup/error
PostProcessing command	
PostProcessing Arguments	
Return data on Output ?	no
Validate Input	no
Target Namespace	http://www.fiorano.com/fesb/activit.
Monitoring configuration	disabled: 5000 milli seconds

Figure 12: Sample Configuration of FileWriter

Use feeder and display component to send sample input and check the response.

Figure 13: Demonstrating scenario 1 with sample input and output

Input Message

```
Input Text
```

Output Message

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:Result xmlns:ns1="http://www.fiorano.com/fesb/activity/FileWriter1">
   <ns1:FileInfo>
      <FullName>/home/geetha/backup/Target/out_10032009_142102262_0.txt</FullName>
      <FileName>out_10032009_142102262_0.txt</FileName>
      <FilePath>/home/geetha/backup/Target</FilePath>
      <Type>File</Type>
      <ReadAccess>true</ReadAccess>
      <WriteAccess>true</WriteAccess>
      <Size>10</Size>
   </ns1:FileInfo>
</ns1:Result>
```

4.5.2.6 Use Case Scenario

In a revenue control packet scenario if an error occurs while parsing the transaction data, error message is written to file.

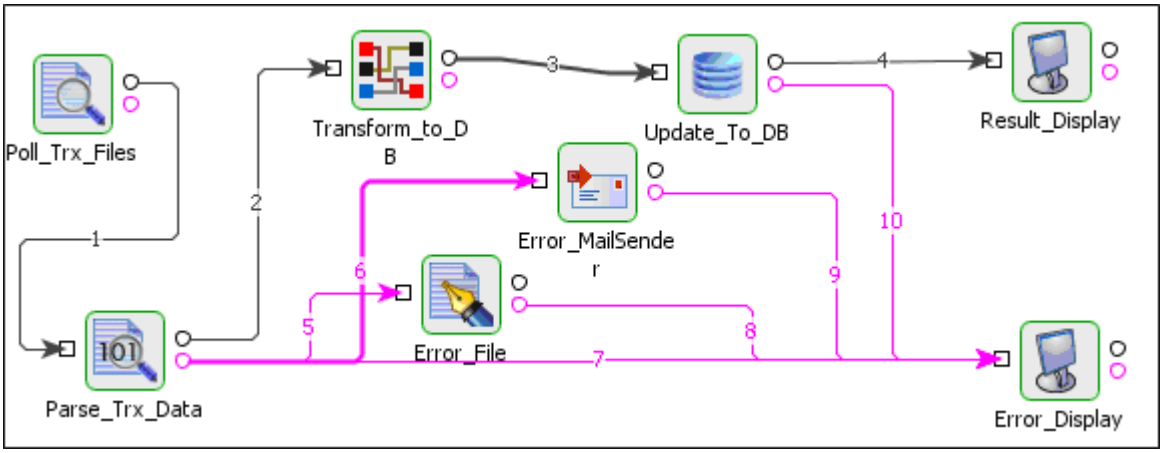


Figure 14: Revenue Control Packet Scenario

The Event Process demonstrating this scenario is bundled with the installer.

Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Studio.

4.5.3 File Transmitter

The FileTransmitter component reads files from the file system and sends their contents to the output port. Data from the source file is read as bytes and is sent to the output port as chunks. The component provides flexible monitoring capabilities and ensures reliable data transfer.

Note: FileTransmitter and FileReceiver components work together as a unit.These components should not be decoupled.

4.5.3.1 Configuration and Testing

The Configuration property sheet of File Transmitter is shown in Figure 4.262.

Chunk Size (in bytes)	100000
Source directory	.
Start timeout	30000
Packets per update	100
Status on Percentage Increase	50
Status on Packets Transmitted Count	1000
Status on Delay Interval	60000
Monitoring configuration	disabled: 5000 milli seconds
Delay before retransmission	60000

Figure 4.262: Sample FileTransmitter Configuration

The table below provides description for the properties in the CPS.

Property	Description
Chunk size	Number of bytes of the source file to be sent in each packet.
Source Directory	The directory from where the FileTransmitter picks the files to be transmitted.
Start timeout	Transmitter sends a Start packet to know the existence of receiver(s). This timeout is the time (in milliseconds) to wait before resending a Start packet.
Packets per update	Number of packets to be sent before saving the transfer state to disk.
Status on Percentage Increase	Maximum increase in percentage completion before the FileTransmitter sends another status report.
Status on Packets Transmitted Count	Maximum increase in the number of chunks sent before the FileTransmitter sends another status report.
Status on Delay Interval	Maximum delay, in milliseconds, before the FileTransmitter sends another status report.
Delay before retransmission	Delay, in milliseconds, after which the FileTransmitter calculates the file status from the receiver and sends retransmissions.

The configuration can be validated using the **Validate** button. Note that this button doesn't check the existence of the source directory. A sample result of the **Validate** operation is shown in Figure 4.263.

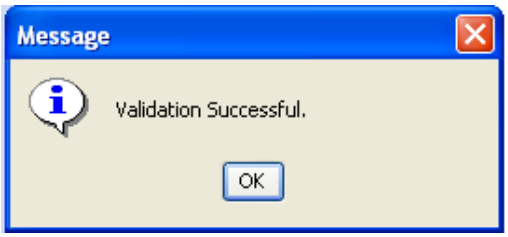


Figure 4.263: Validating the configuration

Input Ports

- Command - Accepts the commands.
- Acknowledgement - Receives acknowledgements from the connected FileReceiver component.

Output Ports

- Data – Sends file data
- Status – Sends the status of file(s) being transmitted.

Commands to FileTransmitter

Send, Stop, Pause, Resume

1. Send

   Syntax:

   ```
   Send <sourceFileName> [o/O] [DestinationDirectory]
   ```

   Notes:

   1. [o/O] represents that the file should be overwritten.
   2. Overwrite bit and Destination directory are optional. Absence of Destination directory puts the file in the default destination directory configured in FileReceiver.
   3. In case a Transmitter is transmitting files to a Receiver present on a different type of Operating system, specifying an absolute path for the DestinationDirectory may not work. In such situations, relative path names should be specified (which is appended to the Destination directory configured in Receiver).

   Examples:

   Send Sample.txt o destDir\

   Send Sample1.txt O destDir\Sample2.txt

   Send Sample1.txt destDir\subDir\Sample2.txt

2. Stop

   Syntax:

```
Stop <sourceFileName>
```

Note: Aborts the transfer of the file specified and sends a kill packet to the connected File receiver which in turn deletes the file being transfered.

Example:

Stop Sample1.txt

3. Pause

   Syntax:

   ```
   Pause
   ```

   Note: Puts the FileTransmitter into the paused state, which suspends file transfers until the Resume command is entered.

   Example:

   ```
   Pause
   ```

4. Resume

   Syntax:

   ```
   Resume
   ```

   Note: Take the FileTransmitter out of the paused state, the file transfers resumes.

   Example:

   Resume

5. Status

   Syntax:

   ```
   Status
   ```

   Note: This command sends out:

   a. The state of the FileTransmitter (Paused or Running).
   b. Names of files for which the transfer has to be resumed (to send the missing packets).
   c. Names of files that are yet to be transmitted.

4.5.3.2 Functional Demonstration

Figure 4.264 shows the event process where a FileTransmitter accepts commands from the Feeder and transmits the files to FileReceiver upon request.

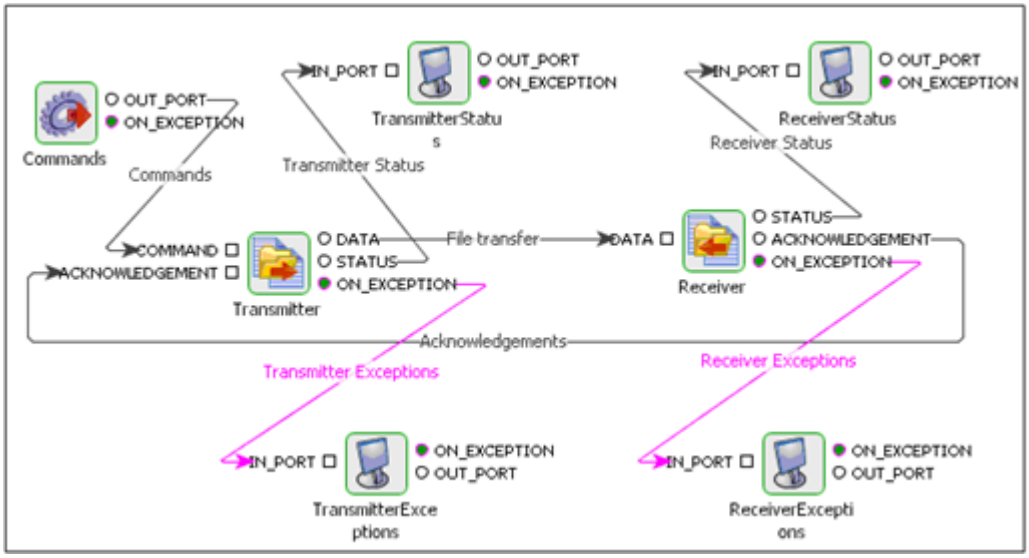


Figure 4.264: Event process showing the File Transfer components

4.5.3.2.1 Scenario 1

Transmitting a file.

Sample Input

The Figure 4.265 shows the sample input from the Feeder.

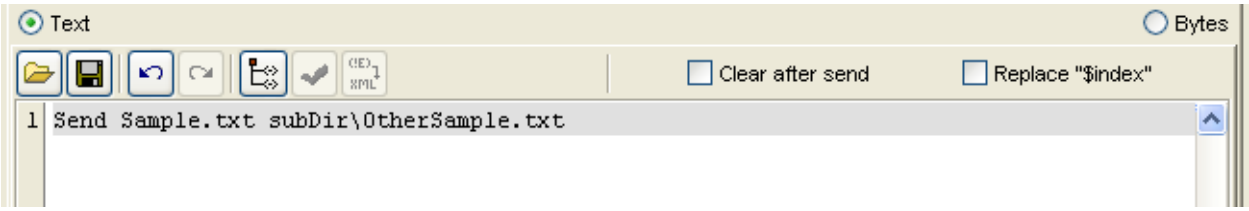


Figure 4.265: Sample input from the Feeder

Sample Output

The Figure 4.266 shows the status messages sent by the FileTransmitter component.

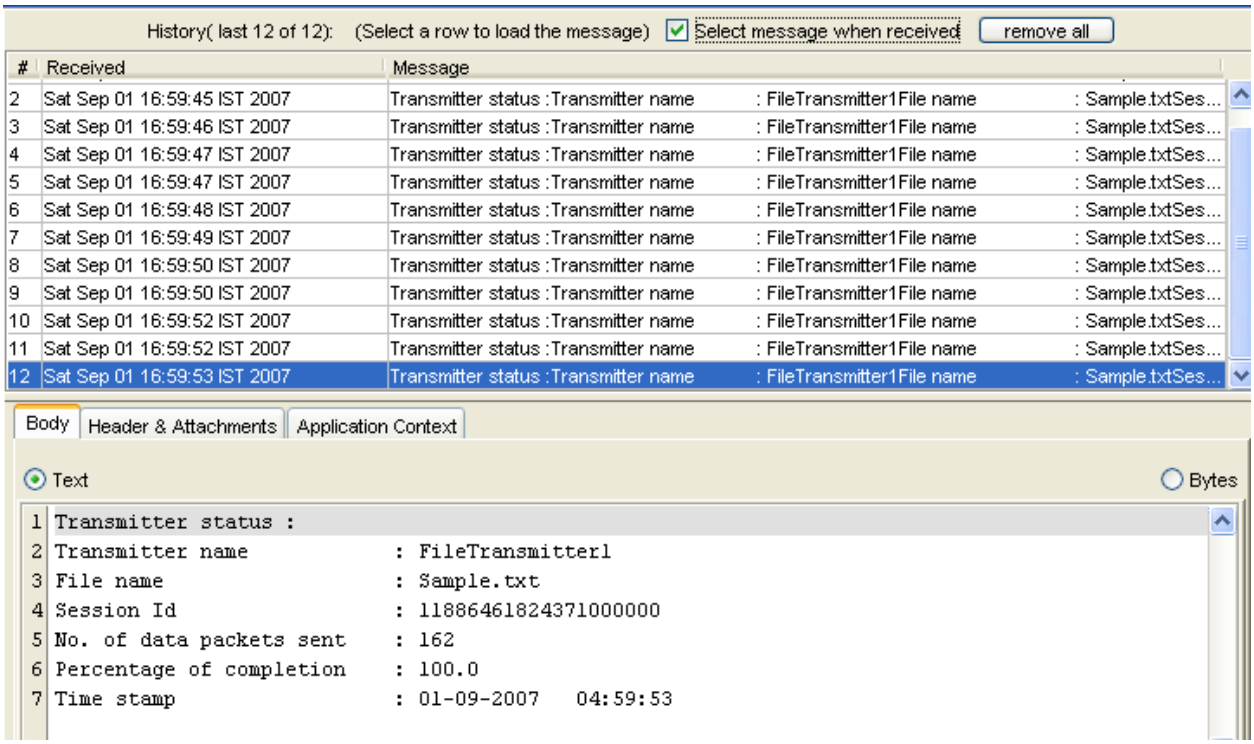


Figure 4.266: Status messages sent by the FileTransmitter component

4.5.3.2.2 Scenario 2

Requesting a report on the current state of FileTransmitter. In this scenario, 1000 send requests are being sent as shown below and then a Status command is sent as shown in the Sample Input to know the files which need data to be resent, the files that are yet to be sent and the current state of transfer (paused/in progress).

Figure 4.267: Information dialog box

Sample Input

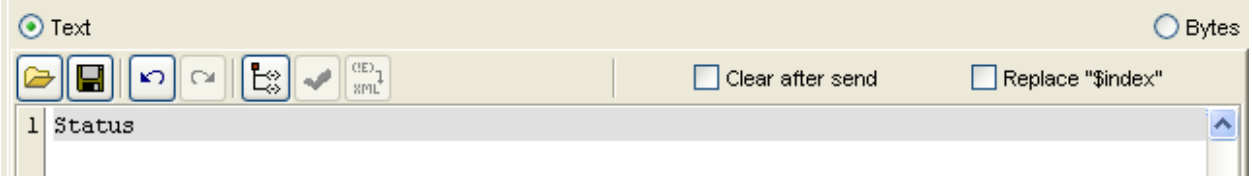


Figure 4.268: Sample input

Sample Output

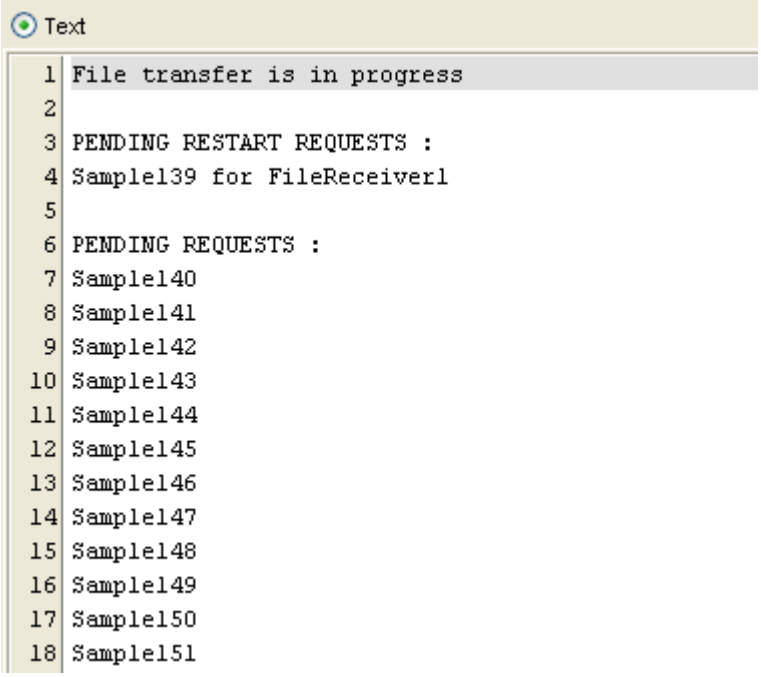


Figure 4.269: Sample output

4.5.3.3 Useful Tips

1. Make sure that the three properties Status on Percentage Increase, Status on Packets Transmitted Count, and Status on Delay Interval are configured depending on the requirement. If all the three are set to small values, the component sends many status messages.
2. If file transfer is across internet, the property **Start timeout** needs to be tuned appropriately to a higher value in order to avoid FileTransmitter sending multiple Start packets unnecessarily.
3. **Note**: the component's cache must always be set to yes, otherwise the component will not be able to resume a transfer on restart.

4.5.4 File Receiver

The FileReceiver component writes the data received on its input port to the output file specified by the user.The component provides flexible monitoring capabilities and robust acknowledgement mechanism.

Note: FileTransmitter and FileReceiver components work together as a unit. These components should not be decoupled.

4.5.4.1 Configuration and Testing

The Configuration property sheet of File Receiver is shown in Figure 4.270.

Attributes	
Destination directory	.
Abort on restart	yes
Restart timeout	60000
Abort timeout	3600000
Number of packets to receive before ac...	10
Status on Percentage Increase	50
Status on Packets Received Count	1000
Status Delay Interval	60000

Figure 4.270: Sample FileReceiver Configuration

The table below provides description for the properties in the CPS.

Property	Description
Destination Directory	Directory where the files being received are to be put by default (in case the 'Send' command doesn't specify the destination directory). Relative destination path names specified in the 'Send' command would be evaluated under this directory.
Abort on restart	'Yes' aborts all file transfers in progress when the component restarts.
Restart timeout	Time (in milliseconds) to wait before resending the receipt status of a particular file to the Transmitter.
Abort timeout	Time (in milliseconds) to wait before sending an Abort packet to the Transmitter to abort transfer of a particular file.
Number of packets to receive before acknowledging	Number of packets to receive before sending the chunk receipt status to the corresponding Transmitter.
Status on Percentage Increase	Maximum increase in percentage completion before the FileReceiver sends another status report.
Status on Packets Received Count	Maximum increase in the number of chunks received before the FileReceiver sends another status report.
Status on Delay Interval	Maximum delay, in milliseconds, before the FileReceiver sends another status report.

The configuration can be validated using the 'Validate' button.Note that this button doesn't check the existence of the destination directory. A sample result of the 'Validate' operation is shown in Figure 4.271.

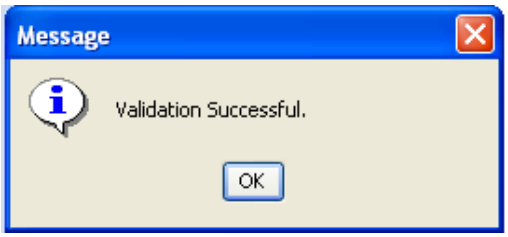


Figure 4.271: Validating the configuration

Input Ports:

- Data – Receives file data.

Output Ports:

- Status – Sends the status of file(s) being received.
- Acknowledgement – Sends acknowledgements to the File Transmitter(s).

4.5.4.1.1 Using Named Configurations

Changes are done in eStudio w.r.t to Named Configurations. The property Destination Directory which was previously present in Component Configuration Panel is now modified as Destination Configuration to use Named Configurations as shown in the Figure below:

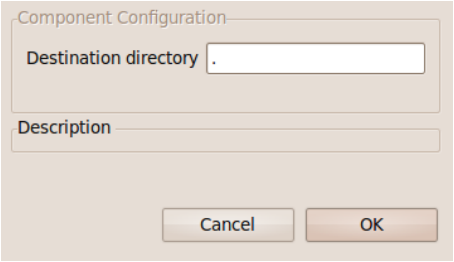


Figure: Destination Configuration

4.5.4.2 Functional Demonstration

Figure 4.272 shows the event process where a FileReceiver receives files transmitted by FileTransmitter.

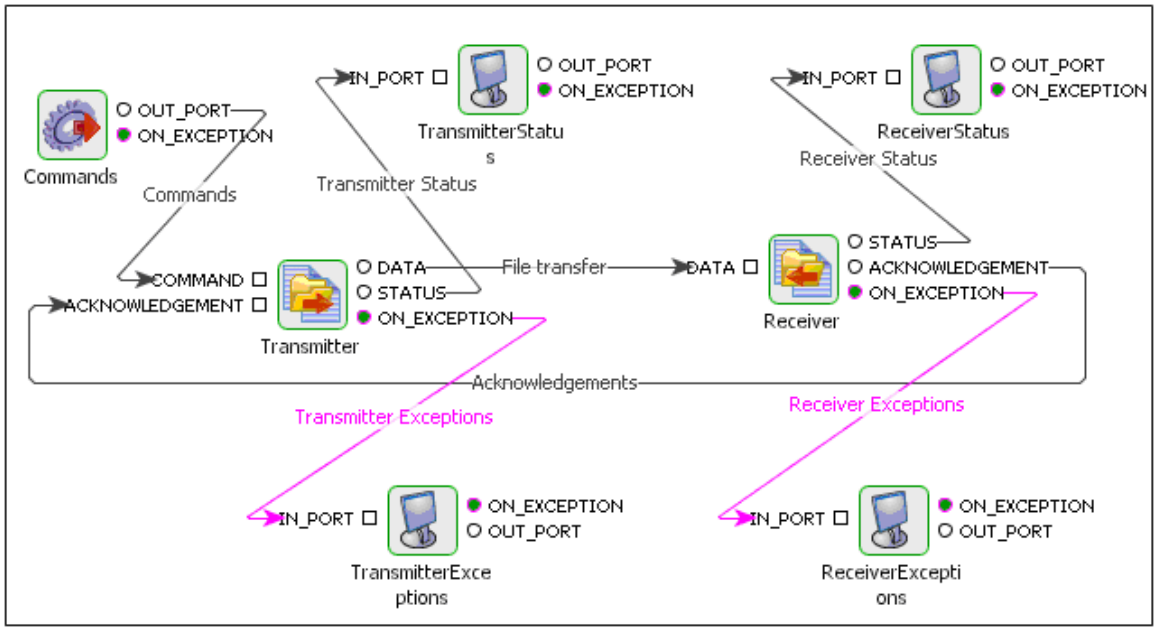


Figure 4.272: Event process showing the File Transfer components

4.5.4.2.1 Scenario 1

FileReceiver receiving a file transmitted by the FileTransmitter.

Sample Input

The below shown input is sent to the FileTransmitter from the Feeder.

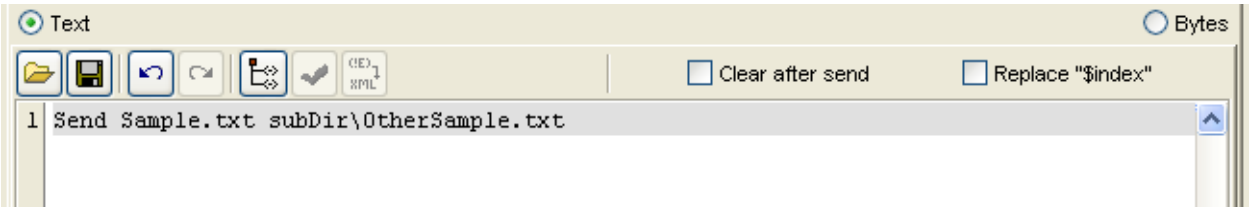


Figure 4.273: Input from Feeeder component

Sample Output

The below shown figure shows the status messages sent by the FileReceiver component.

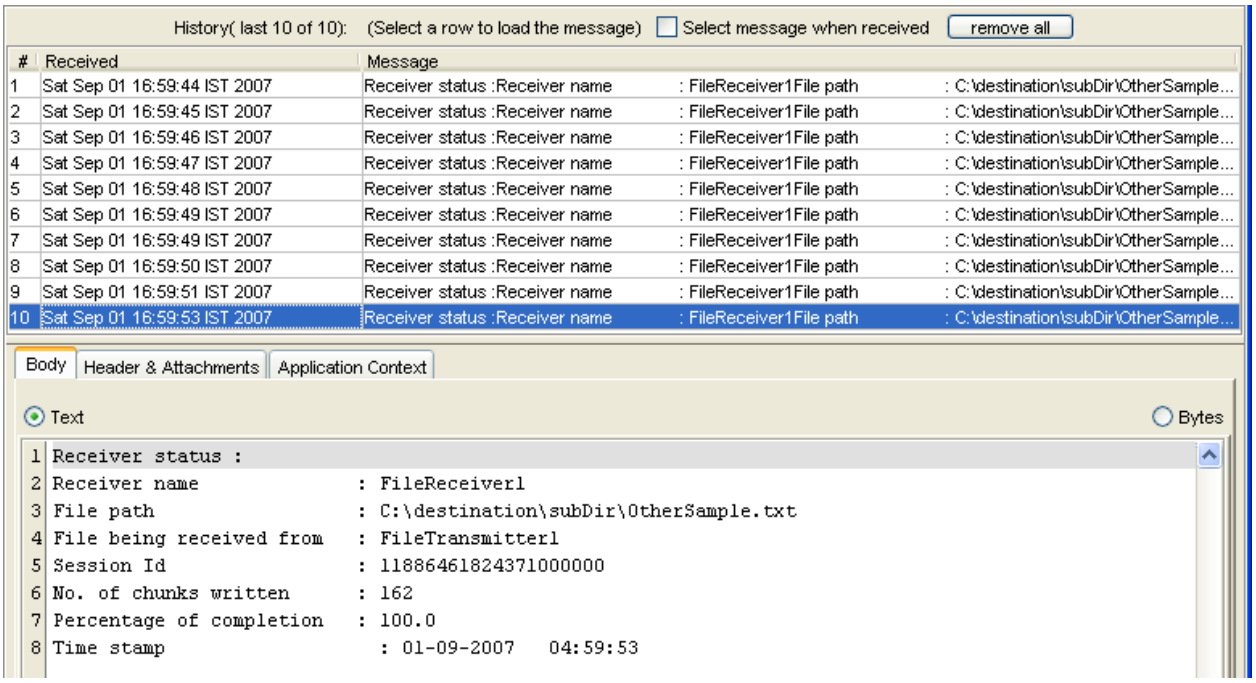


Figure 4.274: Status message from FileReceiver component

4.5.4.3 Useful Tips

- Make sure that the three properties Status on Percentage Increase, Status on Packets Transmitted Count, and Status on Delay Interval are configured depending on the requirement. If all the three are set to small values, the component sends many status messages.
- If file transfer is across internet, the property **Restart timeout** needs to be tuned appropriately to a higher value in order to avoid FileReceiver sending additional acknowledgements unnecessarily.
- **Note**: the component's cache must always be set to yes, otherwise the component will not be able to resume a transfer on restart.

4.6 Flow

The Flow category consists of components like Aggregator, CBR, Cache, DistributionService, Join, Sleep, Timer, WorkList, WorkListManager, XMLSplitter, and XMLVerification. The following section describes each component.

4.6.1 Aggregator

This component collects and aggregates messages received on its IN_PORT based on a specified Completeness Condition. The collected messages are then forwarded as an aggregated bundle of messages to a component connected to its OUT_PORT.

The Aggregator is a special message filter that receives a stream of messages and identifies the messages that are correlated. Once a complete set of messages have been received, the Aggregator collects information from each of these message and publishes a single, aggregated message to the output channel for further processing.

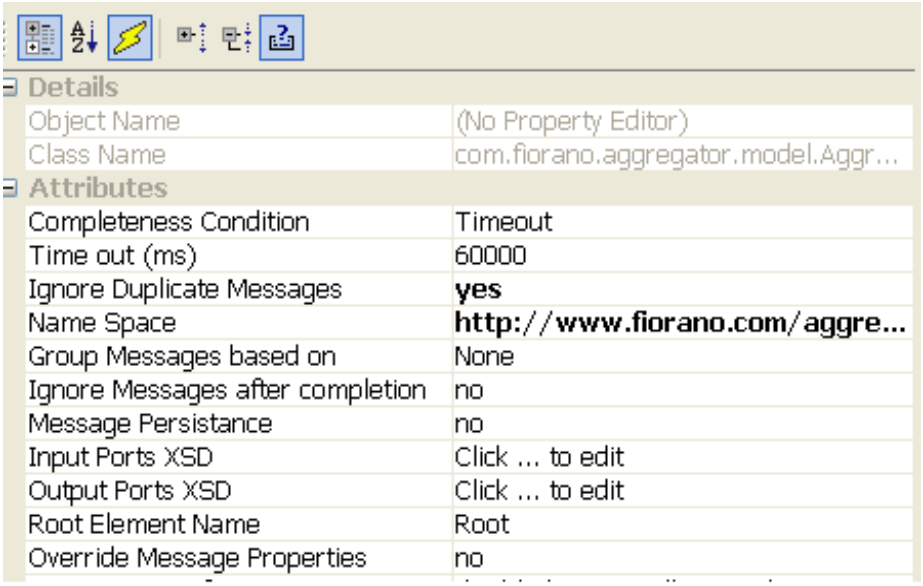
The Aggregator is a stateful component unlike other simple routing components like Content Based Router (CBR) which are generally stateless. Stateless components process incoming messages one by one and are not required to maintain any information between messages.

The component also has an option to persist the message into a RDBMS.

Note: The component ignores the non-XML messages (like text messages) send to its input port. The input is expected to be a valid XML message.

4.6.1.1 Configuration and Testing

The configuration of Aggregator is defined as shown in Figure 1.



Details	
Object Name	(No Property Editor)
Class Name	com.fiorano.aggregator.model.Aggr...
Attributes	
Completeness Condition	Timeout
Time out (ms)	60000
Ignore Duplicate Messages	**yes**
Name Space	**http://www.fiorano.com/aggre...**
Group Messages based on	None
Ignore Messages after completion	no
Message Persistance	no
Input Ports XSD	Click ... to edit
Output Ports XSD	Click ... to edit
Root Element Name	Root
Override Message Properties	no

Figure 4.275: Default Configuration of Aggregator

4.6.1.1.1 Attributes

4.6.1.1.1.1 Completeness Condition

Aggregator starts aggregating the received messages when the completeness condition is satisfied and sends the aggregated message on to the output port. Completeness condition can be configured to one of the following values:

- **Timeout with Override**

  Messages are aggregated after a specified **Time out (ms)** or after configured number of messages as specified by **Completeness Message Count** are received, which ever happens earlier.

  Example 1: If **Time out (ms)** is 5000 ms and **Completeness Message Count** is 3, aggregator waits for 5000 milliseconds before aggregating. However, if 3 messages are received before 5000 milliseconds, then it will immediately aggregate them and send out. Similarly, if only 2 messages are received in 5000 ms then it aggregates those 2 messages and send out.

  Example 2: When messages with different correlation IDs are received on the input port of the component and **Correlation id** is the condition specified for the property **Group Message based on**, the following procedure is followed to aggregate those messages.

  Let Mn(Cn)@Nsecs denote "nth Message with Correlation ID C has a value n arriving at Nth Second"

  and aggregator is configured as Aggregate after time out with 25 seconds based on correlation ID.

  M1(C1) @0s, M2C2@10s, M3C1@20s, M4C2@30s, M5C1@40s are received on input port of Aggregator.

  @0 seconds, M1 is received, timer T1 is set for 25s for correlation ID 1.

  @10 seconds, M2 is received with correlation ID 2, a new timer T2 starts now for 25s for correlation ID 2.

  @20 seconds, M3 is received, since 25s have not passed after M1 (which has same correlation ID as M3) is received, it is stored.

  @25 seconds, timer for correlation ID 1(T1) goes off and messages M1 and M3 are aggregated and sent out without waiting for M5.

  @30 seconds, M4 is received and stored along with M2 since, they have same correlation ID.

  @35 seconds, T2 times out (25 seconds after M2 has arrived) M2 and M4 are aggregated and sent out.

- **Timeout**

  Messages belonging to the same group are aggregated after time specified by the property **Time out (ms)**. The grouping is done based on the property **Group Message based on.**

- **Wait for 'N' Messages**

  Messages belonging to the same group are aggregated after receiving number of messages as specified by property **Completeness Message Count**. The grouping is done based on the property **Group Message based on.**

- **Dynamic Number of Messages**

  Number of messages to be aggregated is dynamic (that is, changes with time during the lifespan of the business application). This number depends on properties **Aggregation Condition** and **Completeness Message Xpath/Property Name containing completeness condition**.

- **Timeout with Dynamic number of messages**

  Messages are aggregated after a time specified by property **Time out (ms)** or after dynamically decided number of messages are received, which ever happens earlier. This number depends on properties **Aggregation Condition** and **Completeness Message XPath/Property Name containing completeness condition**.

4.6.1.1.1.2 Timeout (ms)

Time (in milliseconds) for which the Aggregator waits, before it aggregates the messages in Cache. It should not be set to '0 (zero). This property is visible when we set **Completeness Condition** to Timeout or Timeout with Dynamic number of messages.

Completeness Message Count

This property specifies the number of messages satisfying the grouping condition after which messages should be aggregated. Default value 0 (zero) indicates infinite number of messages. This property is visible only when we set **Completeness Condition** to Timeout with Override or Wait for 'N' Messages.

Apply completeness on all messages

- **No**

  If a message with a particular correlation ID satisfies the completeness condition then the cached messages having the same correlation ID will be aggregated and sent out.

- **Yes**

  If a message with a particular correlation ID satisfies the completeness condition then the cached messages, irrespective of the correlation ID will be aggregated and grouped based on correlation ID and sent out. Separate output will be sent for each correlation ID.

Example:

Let Ma(Cn) denote message with Correlation ID(C) value n and Message XPath value as a and Aggregator is configured to the Completeness XPath value as X.

Ma(C1), Mb(C2), Mb(C1), Mc(C2), Mc(C3), MX(C1), MX(C2), MX(C3) are sent to the input port of the Aggregator.

- If **Apply completeness on all messages** is set to No

  After receiving MX(C1) message (which satisfies the completeness condition), the messages Ma(C1), Mb(C1) and MX(C1) which have same correlation ID(1) will be aggregated and sent out. After receiving MX(C2) the messages Mb(C2), Mc(C2) and MX(C2) which have same correlation ID(2) will be aggregated and sent out. After receiving MX(C3), the messages Mc(C3) and MX(C3) will be aggregated and sent out.

- If **Apply completeness on all messages** is set to Yes

After receiving MX(C1) message which has the completeness xpath value X, then the messages Ma(C1), Mb(C1) and MX(C1) which have same correlation ID(1) will be aggregated and sent out. Now the cached messages Mb(C2), Mc(C2), Mc(C3) will also be aggregated based on correlation ID and sent out. Mb(C2) and Mc(C2) will be aggregated and sent out in a separate output. Mc(C3) will be sent in a separate output. After receiving MX(C2) which satisfies the completeness condition will be send out without caching. After receiving MX(C3) which satisfies the completeness condition will be send out without caching.

Completeness criteria source

This property determines criteria on which the completeness condition has to be applied. This property will be visible and considered when the **Completeness Condition** set to Dynamic Number of Messages or Timeout with Dynamic number of messages.

- **Input XML Message**

  Aggregator evaluates the Xpath on the incoming XML message to determine the dynamic number of messages or to check the completeness condition. The Xpath can be provided in the property **Completeness Message Xpath**.

- **Event Process Context**

  Aggregator evaluates the Xpath on the Application Context of an incoming XML message to determine the dynamic number of messages or to check the completeness condition. The Xpath can be provided in the property **Completeness Message Xpath**.

- **Document Header Property**

  Aggregator checks the value of the header of the input message to determine the dynamic number of messages or to check the completeness condition. The Header name can be provided in the property **Property Name containing Completeness Condition**.

Aggregation Condition

This property is visible when the **Completeness Condition** is set to Dynamic Number of Messages or Timeout with Dynamic number of messages. Along with **Completeness criteria source**, this property is used to compute the dynamic number of messages to be aggregated.

- **Aggregate till count of messages**

  Messages are aggregated upto dynamically decided number of messages satisfying grouping condition. The number of messages can be determined by evaluating the XPath on completeness criteria source (Input XML Message/Event Process Context/ Document Header Property) and checks if the cached messages are equal to the value determined, if equal then the cached messages are grouped based on the grouping condition and sent out.

  Example: Let Completeness criteria source is set as Document Header Property and Property Name containing Completeness Condition as No_of_Messages and Apply completeness on all messages to No.

  Let MHN(Cn) - Message with Correlation ID C has a value n and the header **No_of_Messages** has value N.

  MH3(C1), MH1(C1), MH3(C2), MH3(C1), MH2(C2) are sent to the input port of Aggregator.

When MH3(C1) is received the component caches the message and determines the completeness message count from the header as 3 and the cached message count as 1. Since, message count is not equal to 3 the Aggregator waits for another message. After receiving MH3(C1) the number of cached messages which are having correlation ID value 1 will become 3 which is equal to the header value of the message, so the messages MH3(C1), MH1(C1) and MH3(C1) will be aggregated and sent out. After receiving MH2(C2) the number of cached messages which are having correlation id value 2 will become 2 which is equal to the header value of the message so the messages MH3(C2) and MH2(C2) will be aggregated and sent out.

- **Aggregate till matching condition**

  When this option is selected, then the completeness condition will be evaluated as follows:

  The XPath which is provided in **Completeness Message Xpath/Property Name containing completeness condition** will be evaluated on the **Completeness Criteria Source** (Input XML Message/Event Process Context/ Document Header Property) and this value is compared to the value provided in the property **Matching String**. If both are equal then the Aggregator will aggregate the cached messages based on the grouping condition.

Completeness Message Xpath

This property is visible when **Completeness Condition** is set to Dynamic Number of Messages or Timeout with Dynamic number of messages. This launches an Xpath Editor which can be used to configure the Xpath. If the **completeness criteria source** is Input XML Message, then before configuring this property, input port XSD must be specified against property **Input Ports XSD** to show the input XML structure in the XPath editor.

Property Name containing completeness condition

This property specifies the name of Header in the input message based on which Aggregator will determine the dynamic number of messages or the completeness condition. This property is visible when **Completeness Condition** is set to Dynamic Number of Messages or Timeout with Dynamic number of messages and Document Header Property as **Completeness criteria source**.

Matching String

This property specifies the string that is to be matched with the xpath/document header value. All messages satisfying the grouping condition are aggregated till a message contains the **Completeness Message Xpath** or **Property Name containing completeness condition** value matching to the **Matching string** property value. This property is visible only when the property **Aggregation condition** is set to Aggregator till matching condition.

4.6.1.1.1.3 Ignore Duplicate Messages

- **Yes**

  Aggregator caches all the messages and doesn't check if the message is duplicate or not.

- **No**

  Aggregator checks for the duplicate messages and ignores the duplicate messages.

The procedure to determine the duplicate message is as follows

- Checks whether the message has a duplicate Correlation ID. If yes, then computes value for the property specified by **Duplication Identifier**.
- Now checks if a message already exists in the cache which have same value for Duplicate Identifier. If so, the Aggregator treats this message as duplicate.

Duplication Identifier

This property determines a condition to identify duplicate documents that the Aggregator should use while ignoring duplicate documents.

Document Identifier can be one of the following:

- **Document Header Property**

  Messages having same value for the property (javax.jms.Message.getStringProperty()) are treated as duplicates. The header name can be provided in the property **Header Property Name**.

- **Application Context**

  Messages containing same Application Context satisfying XPath defined against **XPath** property are treated as duplicate messages. If the XPath is empty or null then messages having same Application Context are treated as duplicates.

- **Text Body**

  Messages containing same text satisfying XPath defined against **XPath** property are treated as duplicates. If the XPath is empty or null then messages having same text are treated as duplicates.

- **Carry Forward Context**

  Messages having same value for REQUEST_ID Property of the Carry Forward Context property are treated as duplicates.

- **Senders Identification**

  Messages having same value for property ESBX__SYSTEM__ SOURCE_SERVICE_INSTANCE of the Carry Forward Context property are treated as duplicates.

XPath

In case the **Duplication Identifier** is either Text Body/Application Context, the Text Body or the Application Context is evaluated by using an Xpath. This property is visible only when the **Duplication Identifier** is set to either Text Body or Application Context.

Header Property Name

In case the **Duplication Identifier** is set to Document Header Property, this value has to be set. It could be one of the names of the message properties. The Duplication Identification is done based on the value of this property.

4.6.1.1.1.4 Name Space

This property specifies the namespace to be used for the root element that encapsulates all aggregated messages. If **Output Ports XSD** is specified, this should be same as the target namespace of the output port XSD. This value will be set automatically when **output port XSD** is specified.

4.6.1.1.1.5 Group messages based on

This property determines a condition to identify similar documents that the Aggregator should use while aggregating. Documents which have same correlation ID are aggregated when the **Completeness condition** is satisfied.

Correlation ID can be one of the following:

- **Document ID**

  Messages having same javax.jms.Message.getJMSMessageID() are aggregated together.

- **Document Correlation ID**

  Messages having same javax.jms.Message.getJMSCorrelationID() are aggregated together.

- **Document Header Property**

  Messages having same value for the property (javax.jms.Message.getStringProperty()) specified are aggregated. The header name can be provided in the property **Header Property Name**.

- **Application Context**

  Messages containing Application Context satisfying XPath defined against **XPath** property are aggregated together. If the XPath is empty or null then messages having same Application Context are aggregated together.

- **Text Body**

  Messages containing text satisfying XPath defined against **XPath** property are aggregated together. If the XPath is empty or null then messages having same text are aggregated together.

- **Carry Forward Context**

  Messages having same value for REQUEST_ID Property of the Carry Forward Context property are aggregated together.

- **Workflow Instance ID**

  Messages having same value for Property ESBX__SYSTEM__ WORK_FLOW_INST_ID in message are aggregated together. This can be used only when Document tracking is enabled.

- **None**

  Messages are grouped as they are received.

Note: Null Correlation ID's are not supported.

Xpath

This property specifies the XPath which will be evaluated on Application Context or Text Body depends on the property **Group messages based on** for grouping. This will launch XPath editor to configure the XPath. If the **Group messages based on** is Text Body, then before configuring this property input port XSD has to be specified against the property **Input Ports XSD** to show the input XML structure in the XPath editor.

Header Property Name

The name of the property based on which grouping of messages is done. This property is visible when the property **Group messages based on** is set to Document Header Property.

4.6.1.1.1.6 Ignore messages after completion

- **Yes**

  Ignores messages containing correlation IDs which are already aggregated.

- **No**

  Restarts aggregating messages containing correlation IDs which are already aggregated.

Example:

Aggregator is configured with completeness condition as Wait for 'N' messages and Completeness Message Count to 3.

Let Mt1(C1) denote message with correlation ID C has a value 1 and text body t1.

Mt1(C1), Mt2(C1), Mt3(C1), Mt4(C1) are sent to the input port of Aggregator. Messages Mt1(C1), Mt2(C1) and Mt3(C1) will be aggregated after receiving message Mt3(C1).

If the property **Ignore messages after completion** is set to Yes, then the message Mt4(C1) will be ignored, since aggregation has been done on the correlation ID 1, and this message is also having same correlation ID so the message will be ignored.

If the property I**gnore messages after completion** is set to No, then the message Mt4(C1) will be cached, and the Aggregator waits for other messages.

4.6.1.1.1.7 Message persistence

- **Yes**

  Persists messages which are not aggregated into a database. If the completeness condition involves only the count of the messages, the old messages are transmitted along with the new messages when the completeness condition is satisfied. In case of timeout, if the timeout occurs when the aggregator is down, the old messages are transmitted when the aggregator is restarted; else, they are transmitted after timeout. The timeout is inclusive of the time when the aggregator was down. In case of persisting messages, message properties are not stored.

  Aggregator itself takes the responsibility of starting database and creating tables. It internally uses **Mckoi** database.

- **No**

  Maintains all messages that are not aggregated in an inMemory data structure.

Table Name

The name of the table that stores messages received by the Aggregator. The value of this field can be left as null. In such an instance, a table with tif_ as prefix is used. This property is visible and the table is used when **Message persistence** property is set to Yes.

4.6.1.1.1.8 Input Ports XSD

The XSD of the expected input messages.

4.6.1.1.1.9 Output Ports XSD

The XSD for the aggregated output message.

4.6.1.1.1.10 Root Element Name

Root element name that encapsulate all aggregated messages. If **Output Ports XSD** is specified, this should be same as the root element set in output port XSD. This value is set automatically when output port XSD is specified.

4.6.1.1.1.11 Override Message Properties

- **Yes:** While aggregating messages, if the messages have a same property with different values, the property value of the last message (having this property) in the aggregation is set as the property value for the aggregated message.
- **No:** While aggregating messages, if the messages have a same property with different values, the property value of the first message (having this property) in the aggregation is set as the property value for the aggregated message.

4.6.1.2 Functional Demonstration

4.6.1.2.1 Scenario 1

Aggregating messages based on the timeout specified.

Configure the Aggregator as described in section Configuration and Testing component to send sample input and to check the response respectively.

In the example below, only **Input ports XSD** is set to chat schema (can use Fetch from Schema for this) in Aggregator Custom Property Sheet and all the remaining properties are left as default.

Figure 4.276: Demonstrating Scenario 1

On timeout, the aggregated messages are sent from the output port to the Display component.

4.6.1.2.2 Scenario 2

This scenario explains the usage of **Dynamic number of messages** completeness condition.

Configure the Aggregator as described in section Configuration and Testing and use chat and display component to send sample input and to check the response respectively.

In the example below, Aggregator is configured in such a way that if it receives a message Fiorano at the input port, it aggregates and sends the messages to the output port. The Figure 3 shows the configuration.

Attributes	
Completeness Condition	**Dynamic Number of Messages**
Apply completeness on all messages?	no
Completeness criteria source	Input XML Message
Aggregation Condition	**Aggregare till matching condition**
Completeness Message XPath	**/ChatMessage/Message[/]**
Match String	**Fiorano**
Ignore Duplicate Messages	**yes**
Name Space	**http://www.fiorano.com/aggregator**
Group Messages based on	None
Ignore Messages after completion	no
Message Persistance	no
Input Ports XSD	Click ... to edit
Output Ports XSD	Click ... to edit
Root Element Name	Root
Override Message Properties	no

Figure 4.277: Sample Configuration used in Scenario 2

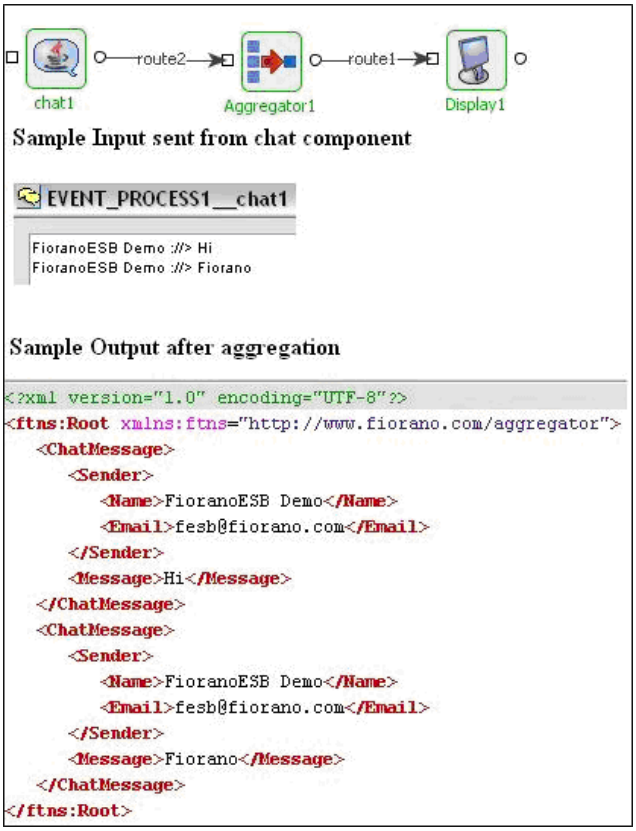


Figure 4.278: Scenario demonstration with sample input and output

4.6.1.3 Useful Tips

- When using the persist option, the database is local to the machine on which the Peer Server is running. In case of a Peer Server failover, the component creates a new table and therefore messages cannot be recovered.
- When using header property for grouping messages, properties with names like/starting with ESBX__SYSTEM__* cannot be used.

4.6.2 CBR

CBR (Content Based Routing) is used to route the incoming messages on to different destinations based on the content of the messages.

The component creates a port for each of the XPath expressions specified and the messages satisfying the particular XPath is sent onto the respective port. In addition to these ports an output port **OUT_FALSE** is created and messages whose content does not satisfy any of the XPath expressions will be sent out of this port. If more than one XPath condition is true, the message is sent on all the ports for which the XPath condition evaluates to true.

4.6.2.1 Configuration and Testing

The CBR component can be configured using its Custom Property Sheet wizard.

4.6.2.1.1 Schema

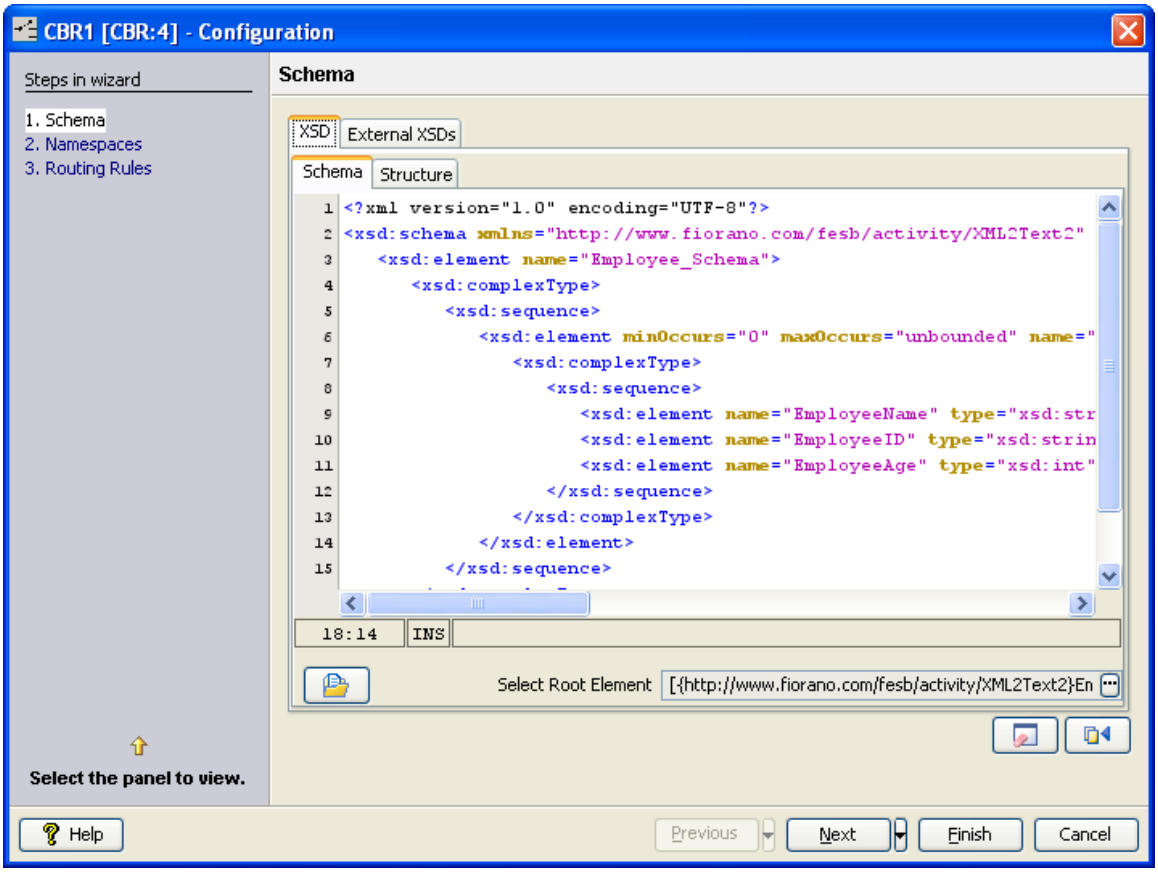


Figure 4.279: Provide schema

The schema of the XML content that is used for routing should be provided using the Schema Editor in the Schema **panel**. The XML content that is used for routing is determined by property Apply Xpath on Context in Routing Rules panel.

Refer to Schema Editor section in Common Component Configurations chapter for information about **Schema Editor** shown in Figure **4.279**.

4.6.2.1.2 Namespaces

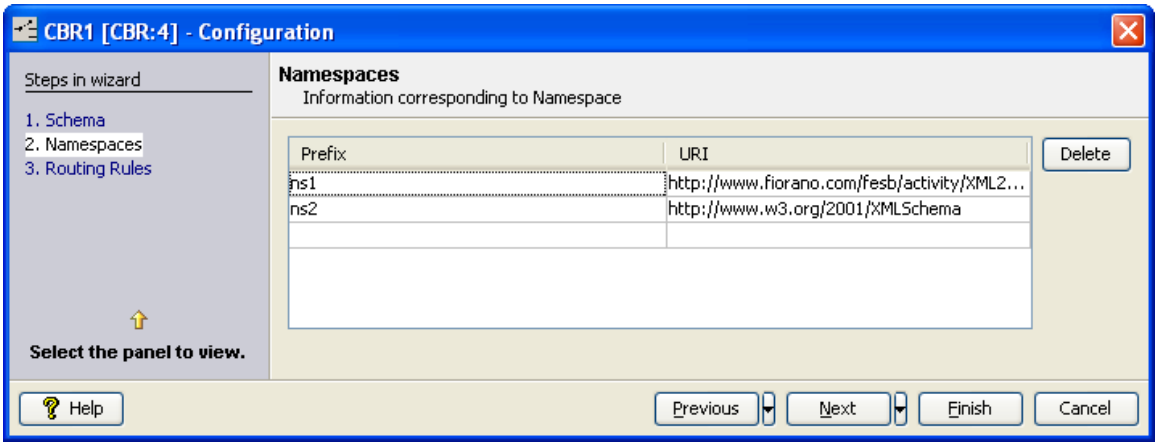


Figure 4.280: List of namespaces

List of namespace prefixes that are used in the XPath and the namespaces they represent are provided in the Namespaces panel. The table in this panel is automatically populated with namespaces defined in XML schema provided in **Schema** panel.

To remove unnecessary namespaces, select the row containing namespace and click the **Delete** button.

To add a new namespace prefix and namespace, enter the details in the empty row and press **Enter** key.

4.6.2.1.3 Routing Rules

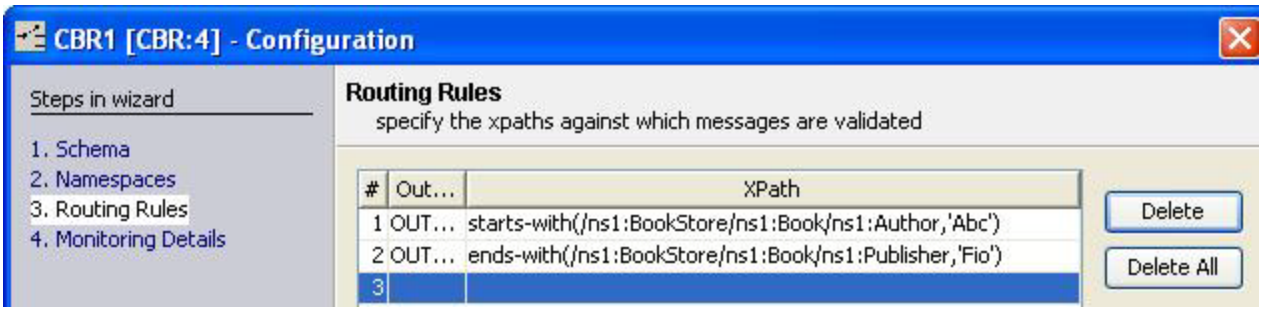


Figure 4.281: XPath(s) to be evaluated on the input messages

XPaths based on which the routing is done can be configured in Routing Rules panel. XPath editor can be used to configure XPaths. Click the ellipsis button in the **XPath** column to open XPath Editor. Alternatively, a valid XPath expression can be typed directly in the table

An empty row is automatically added after closing the XPath Editor or after pressing Enter key after the XPath is manually provided.

Refer to XPath Editor section in Common Configuration Configurations chapter for information about **XPath Editor** shown in Figure **4.279**.

For each configured XPath expression, an output port is created with the name provided in OutPort_Name column, messages satisfying the given XPath condition are routed onto this port. Port names ON_EXCEPTION, OUT_FALSE and ON_TRANSACTION_FAIL are reserved and cannot be set in OutPort_Name column.

Note:

- XPath in CBR should be configured to return a boolean value for routing messages.
- XPath can also contain simply the path of any element, then CBR checks for the existence of the particular element in the input xml. If the input xml contains the element then it will be routed to the corresponding output port of that XPath, if the input xml does not contain the element then it will be routed to the output port **OUT_FALSE**.
- XPath can be valid JMS Message selectors. If the JMS Message selector is used the runtime argument **useFioranoCBR** should be set to **yes** as shown in Figure **4.282**.

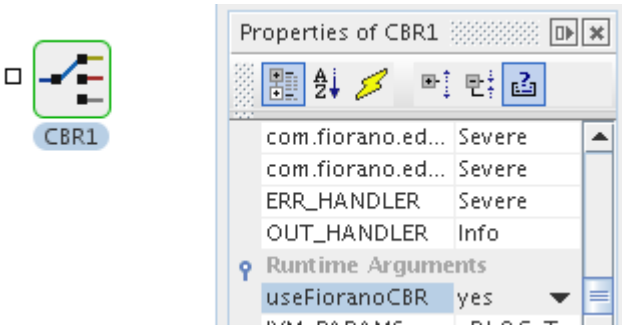


Figure 4.282: Setting runtime argument 'useFioranoCBR'.

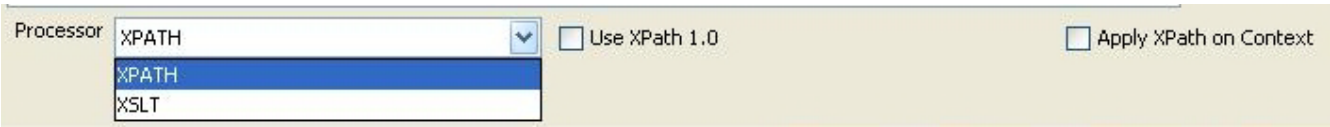


Figure 5: XPath processor properties in Routing Rules page

4.6.2.1.4 Processor

Specifies the processor used to evaluate XPath conditions. If the number of XPath conditions are more (greater than 4), then XSLT processor is preferable for better performance. If the XPath conditions are less then both processors will give equal performance. XPath processor is preferable if the XPaths are complex. XSLT processor might not work for all XPaths.

- **XPath**

  Uses Saxon based XPath evaluator to evaluate XPaths. Values in column XPath should be valid XPath expressions. XPaths are validated if XPath processor is selected. The validation will be done for some simple XPath conditions only. So, if the XPath is validated, there is no guarantee that the XPath results a boolean value.

  - **Use XPath 1.0** – When this property is selected, XPath 1.0 is used for evaluation. Otherwise, XPath 2.0 is used. This checkbox is visible only when the Processor is selected as XPath.

- **Apply XPath on Context –** When this property is selected, the XPath is evaluated on the Application Context of the input message. Otherwise, XPath is evaluated on Body of the message.

  When XPath is evaluated on Context of the message, no schema is set on the ports of CBR. In case of evaluating text body of the message, schema provided earlier in the configuration (if present) will be set on the ports.

  Note: When this option is chosen, provide application context schema in the schema panel.

- **XSLT**

  Uses XSLT to evaluate condition. Values in column XPath can be any valid Boolean condition in XSLT. There is no validation check on XPath conditions if we select XSLT processor.

  Note: While using XSLT processor, if XPath contains any functions having namespace prefixes like saxon:parse(), then the appropriate prefix and namespace URL pair should add in the Namespace panel. For example to use saxon functions namespace, URL http://saxon.sf.net/ should add to prefix saxon in the Namespace panel.

  Example: For a value /ns1:Transaction/ns1:request/ns1:source = 'tserv' provided in column XPath the following XSL is used.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:ns1="http://www.incomm.com/transaction/2008-02"
xmlns:ns2="http://www.w3.org/2001/XMLSchema">

<xsl:output method="text" encoding="UTF-8"/>
<xsl:template match='/'>
<xsl:if test="/ns1:Transaction/ns1:request/ns1:source ='tserv'">TSERV,</xsl:if>

</xsl:template>
</xsl:stylesheet>
```

4.6.2.2 Functional Demonstration

4.6.2.2.1 Scenario 1

CBR is configured to filter the messages on the basis of employee age in the input message. It is sent to one of the output ports depending on the XPath specified in the CBR.

Configure the CBR as described in Configuration and Testing section and use feeder and display component to send sample input and check the response respectively.

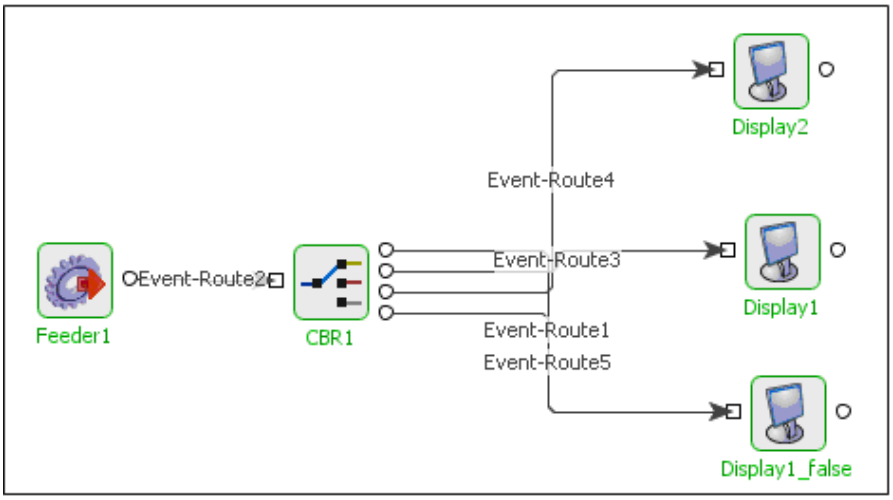


Figure 4.283: Demonstrating Scenario 1 with sample input and output

Sample Input:

```
<ns1:Employee_Schema xmlns:ns1="http://www.fiorano.com/fesb/activity/XML2Text2">
 <ns1:Employee>
<ns1:EmployeeName>ashish</ns1:EmployeeName>
<ns1:EmployeeID>294</ns1:EmployeeID>
<ns1:EmployeeAge>22</ns1:EmployeeAge>
 </ns1:Employee>
</ns1:Employee_Schema>
```

Sample Output:

```
<ns1:Employee_Schema xmlns:ns1="http://www.fiorano.com/fesb/activity/XML2Text2">
 <ns1:Employee>
<ns1:EmployeeName>ashish</ns1:EmployeeName>
<ns1:EmployeeID>294</ns1:EmployeeID>
<ns1:EmployeeAge>22</ns1:EmployeeAge>
 </ns1:Employee>
</ns1:Employee_Schema>
```

4.6.2.3 Use Case Scenario

In Purchasing System sample, a purchase request is sent by the user to a company through a web based interface. The purchase order consists of three inputs namely REQUEST, CREDENTIALS, and SYNC_PO_002. REQUEST is an identifier string for the request. The request identifier is verified by a CBR (Content Based Router) component to be the correct one for which this system is expected to service the request.

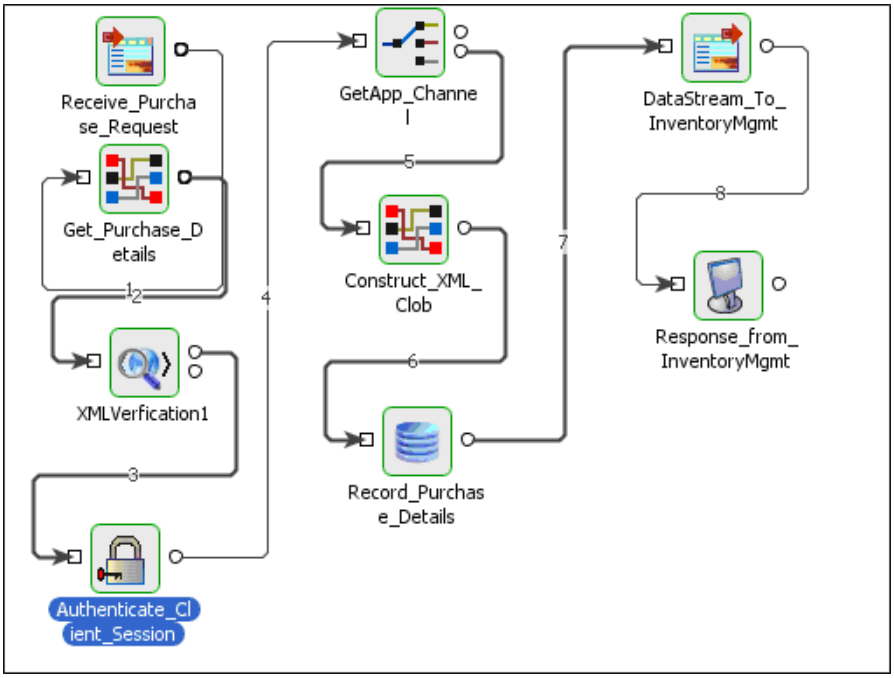


Figure 4.284: Purchasing System sample

4.6.2.4 Useful Tips

The order of the XPath conditions specified in the CPS does not matter.

The CBR component supports XPath version 1.0 and 2.0. The default is version 2.0. To use version 1.0, please select the checkbox Use XPath 1.0 in the CPS.

To monitor time taken for executing each request, set log level for logger com.fiorano.edbc.cbr.monitor to INFO.

When runtime argument for this component, **useFioranoCBR** is set to **yes**, please make sure that the XPaths provided in the configuration are valid JMS Message. More information on JMS Message Selectors can be found in the documentation for **Message** interface in JMS APIs. http://java.sun.com/products/jms/javadoc-102a/index.html.

To apply XPath on context check **Apply XPath on Context** in **Routing rules** panel and provide Application context schema in the **schema** panel. In this case schema will not be set on output ports.

4.6.3 Distribution Service

This component is used for distributing workload of N Jobs amongst M Workflow processors. Typically this component is used before multiple instances of the same component and the load balancing mechanism in the component is used to distribute the messages received by this component. The component uses weighted round robin mechanism

Note: The number of ports configured for the component should be the number of instances being used to share the load.

4.6.3.1 Configuration and Testing

The DistributionService component can be configured using its Custom Proper Sheet wizard.

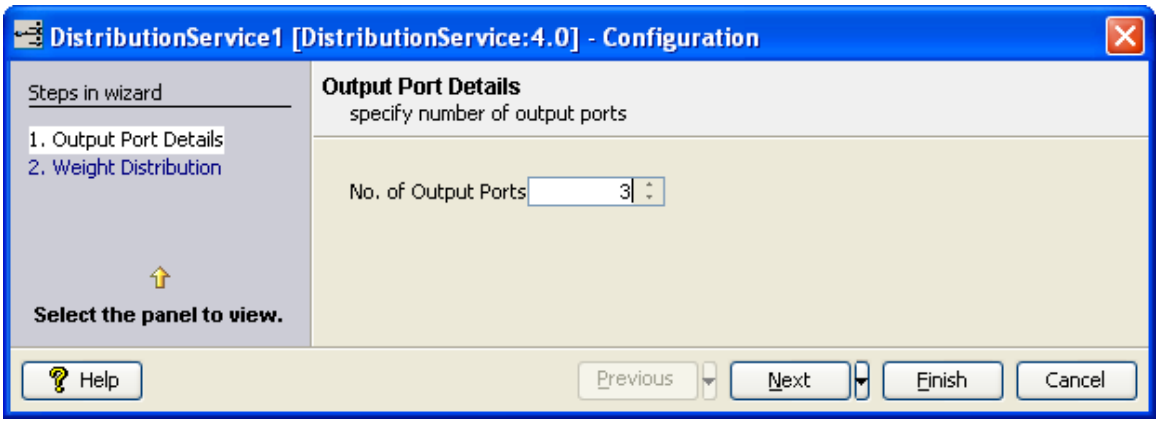


Figure 4.283: Sample DistributionService configuration

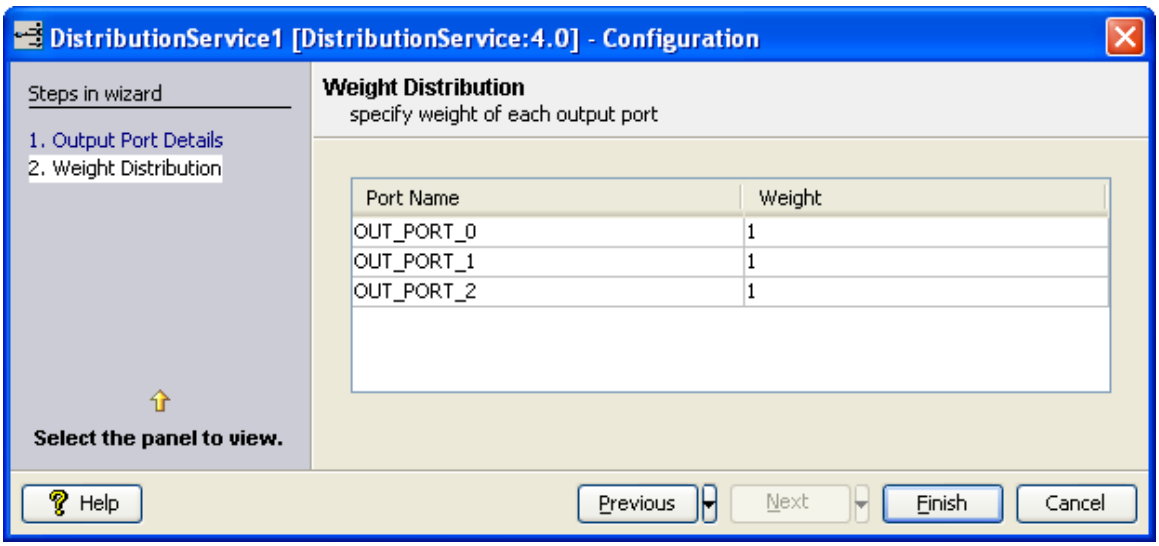


Figure 4.284: Sample DistributionService configuration

4.6.3.2 Functional Demonstration

4.6.3.2.1 Scenario 1

Send multiple requests to DistributionService, each is displayed in separate display components.

Configure the DistributionService as described in *Configuration and Testing* section and use feeder and display component to send sample input and check the response respectively.

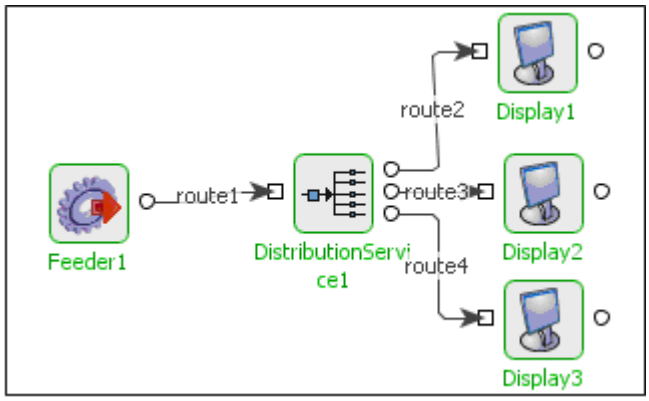


Figure 4.285: Demonstrating Scenario 1 with sample input and output

Sample Input:

```
Input Message
```

Sample Output:

```
Output Message
```

4.6.4 Join

The Join component can be used to join two input XML structures using the Fiorano Mapper into one output XML. This component has two input ports and three output ports. The two input ports, **IN_PORT1** and **IN_PORT2** are used to input the two XML structures that have to be joined. After the join operation is performed messages are sent on each of the output ports.

- Message received on input **IN_PORT1** is sent on output port **OUT_PORT_IN1.**
- Message received on input **IN_PORT2** is sent on output port **OUT_PORT_IN2.**
- The result of the join operation is sent on **OUT_PORT_RESULT**.

When the component receives a message on one of the input ports, it checks if there are messages that are received on the other port which are not already used in join operation.

- If there are no messages that are received on the other port and are not already used in join operation, the message received is added to an internal queue. There is a separate queue for each input port that holds messages until the join operation is performed.

- If there are messages received on the other port which are not already used in the join operation, then the first message is picked from the queue and the join operation is performed.

4.6.4.1 Configuration and Testing

The Join component can be configured using its Custom Property Sheet as shown in figure 1.

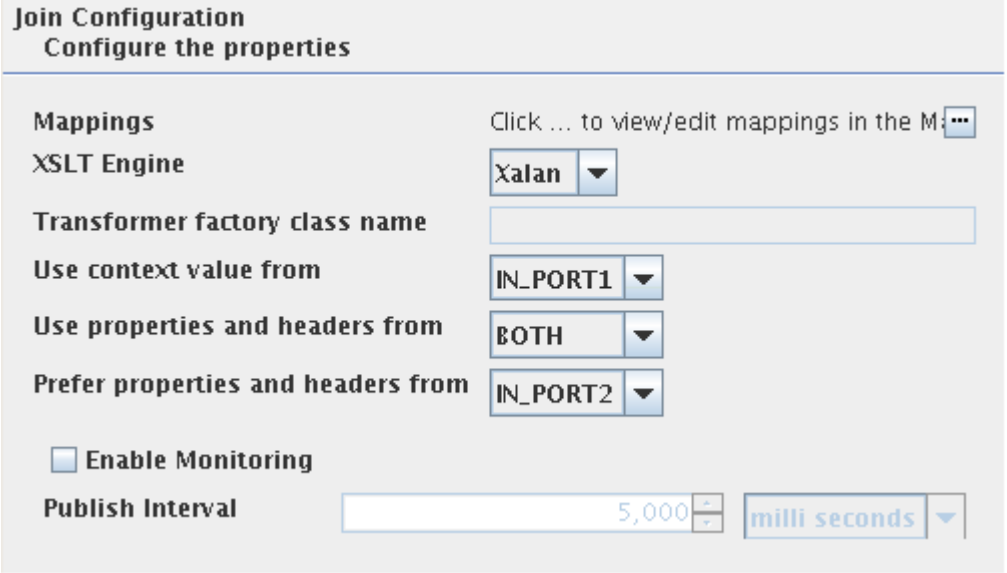


Figure 1: Join configuration property sheet

4.6.4.1.1 Mappings

The mappings between input and output structures can be defined by clicking on the ellipsis button against this property. Fiorano Mapper gets launched upon clicking the ellipsis as shown in figure 2.

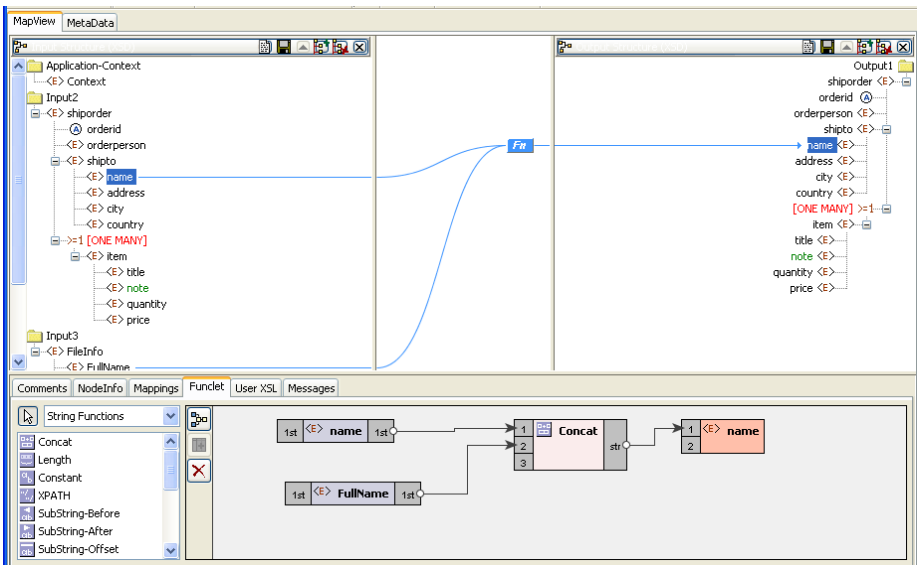


Figure 2: Configuring mappings using Mappe

XSLT engine for the join operation can be specified by this parameter. Join operation is performed using a XSLT. The component can be configured to use a specific XSLT engine to perform XSLT. Xalan (2.7.0) and Saxon (8.4) transformer implementations are bundled with Fiorano environment for performing transformations. By default, the component uses **Xalan**.

- **Xalan**

  Xalan implementation (org.apache.xalan.processor.TransformerFactoryImpl) is used to perform transformation.

  Note: Xalan (2.7.0) does not support XSLT 2.0

- **Saxon**

  Saxon implementation (net.sf.saxon.TransformerFactoryImpl) is used to perform transformation.

  Note: Saxon implementation does not support custom functions.

- **Other**

  This option should be used when a custom transformer implementation has to be used.

  Note: Selecting this option enables the property **Transformer factory class Name** which can be used to provide the transformation factory implementation that should be used.

4.6.4.1.2 Transformer factory class Name

This property determines the fully qualified name of the class which should be used to perform transformation when the property **XSLT Engine** is specified as **Other**. The class provided should be an implementation of javax.xml.transform.TransformerFactory.

Resources (jar files) containing the java class specified against this property should be added as resources to Join component.

4.6.4.1.3 Use context value from

This property determines the input port from which the value of the application context has to be picked. This value is set on the joined message coming out of **OUT_PORT_RESULT** port. One of **IN_PORT1** or **IN_PORT2** can be chosen.

4.6.4.1.4 Use properties and headers from

This property determines the input port from which the headers / properties have to be picked up and set on the joined message coming out of **OUT_PORT_RESULT** port. If either **IN_PORT1** or **IN_PORT2** is chosen, properties are fetched form the chosen port and properties from the other port are discarded. If **BOTH** is chosen, properties from both the ports are set on the joined message.

Note: **When** BOTH **is selected, the property** Prefer Properties and Headers from will be enabled and if both input ports have any properties with same name, the values of such properties are picked based on that property.

4.6.4.1.5 Prefer Properties and Headers from

When the property **Use properties and headers from** is set as **BOTH** and if there are headers / properties with same name on both **IN_PORT1** and **IN_PORT2** with different values, the port from which the values of such headers / properties have to be picked up and set on the joined message is determined by this property.

Note: This property is enabled when the property **Use properties and headers from** is set as **BOTH**.

4.6.4.1.6 Message matching configuration

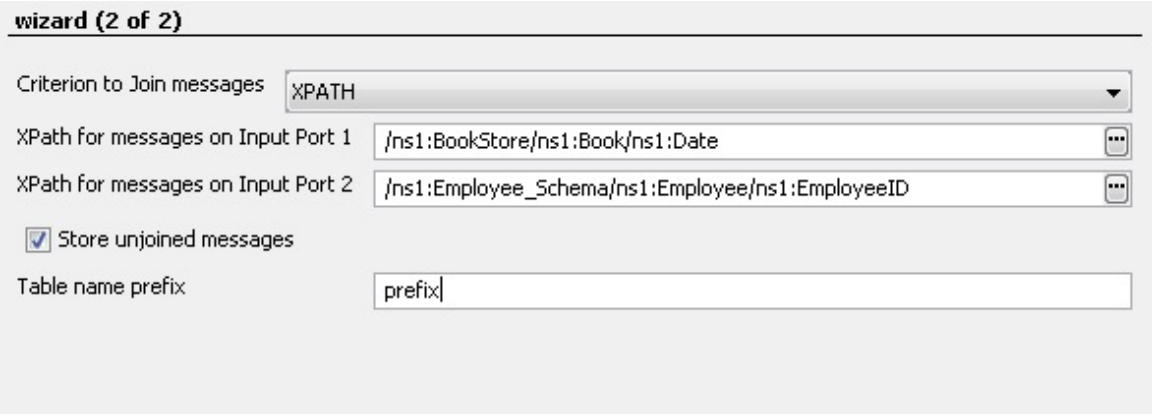


Figure: Message Matching Configuration

The second panel defines the way messages coming on input ports have to be matched.

Criterion to join messages

The messages can be matched based on different criteria

- **NONE**: Messages will be joined in the order in which they come onto the input port without checking for any condition.

  Note: In this case, the unjoined messages will be present on the input ports of the component and on failover/restart of the component, the messages will not be lost.
- **Property**: The messages are joined based on the value of the JMS properties on either ports. The names of the properties can be configured using attributes Property name for messages on Input Port 1 and Property name for messages on Input Port 2. The messages with same value for the respective properties are joined.
- **XPath**: The messages are joined based on the value of the element in the input XML at the XPath specified. The XPaths can be configured using attributes XPath for messages on Input Port 1 and XPath for messages on Input Port 2. The messages with same value at respective XPaths are joined.

Note: If the value(of XPath/Property) does not match, the message will be stored inmemory/database. When a matching message comes on the other port later, this message will be joined with that. If multiple messages are stored for a matching message on other port, then they are retrieved in order in which they are received by the component.

Store unjoined messages

When PROPERTY/XPATH is chosen as joining criterion, the messages will be received by the component as they come and maintained in an internal store till a matching message comes. So the messages will get lost if the component is restarted. If this option is chosen the messages are stored in an internal Mckoi DB.

Table name Prefix

The prefix for the table name used for storing the messages. Two tables with <Prefix>__<Input Port Name> will be created for persitence. If left empty, <EventProcessGUID>_<Service name> will be used.

4.6.4.2 Testing

Join component is configured as shown in Figure 2 and the transformation can be tested from Mapper using the Test XSL button.

Sample schema for input XML message 1

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
                        <xs:element name="shiporder">
                            <xs:complexType>
                            <xs:sequence>
                            <xs:element name="orderperson" type="xs:string"/>
                            <xs:element name="shipto">
                                <xs:complexType>
                                <xs:sequence>
                                    <xs:element name="name" type="xs:string"/>
                                    <xs:element name="address" type="xs:string"/>
                                    <xs:element name="city" type="xs:string"/>
                                    <xs:element name="country" type="xs:string"/>
                                </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="item" maxOccurs="unbounded">
                                <xs:complexType>
                                <xs:sequence>
                                    <xs:element name="title" type="xs:string"/>
                                    <xs:element name="note" type="xs:string"
minOccurs="0"/>
                                    <xs:element name="quantity"
type="xs:positiveInteger"/>
                                    <xs:element name="price" type="xs:decimal"/>
                                </xs:sequence>
                                </xs:complexType>
```

```
                                </xs:element>
                                </xs:sequence>
                                <xs:attribute name="orderid" type="xs:string"
use="required"/>
                                </xs:complexType>
                            </xs:element>
</xs:schema>
```

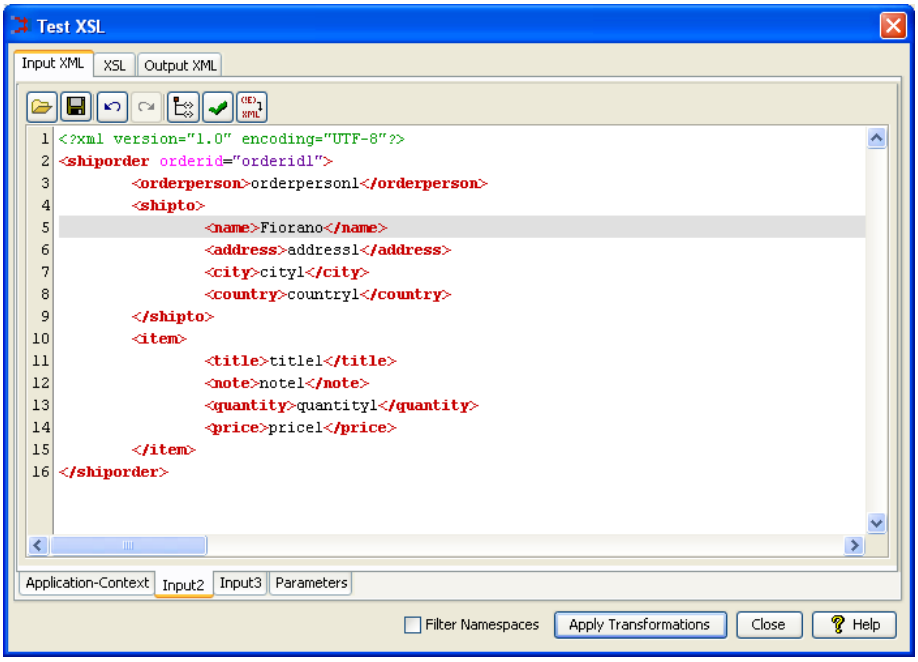


Figure 3: Sample Join input 1 message

Sample schema for input XML message 2

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.fiorano.com/fesb/activity/FileWriter1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.fiorano.com/fesb/activity/FileWriter1">
                            <xsd:complexType name="Result">
                            <xsd:sequence>
                                <xsd:element ref="FileInfo" minOccurs="0"/>
                            </xsd:sequence>
                            </xsd:complexType>
                            <xsd:element name="Result" type="Result"/>
                            <xsd:element name="FileInfo">
                                <xsd:complexType>
                                <xsd:sequence>
                                    <xsd:element name="FullName" type="xsd:string"/>
                                    <xsd:element name="FileName" type="xsd:string"/>
                                    <xsd:element name="FilePath" type="xsd:string"/>
                                    <xsd:element name="Type" type="xsd:string"/>
                                    <xsd:element name="ReadAccess" type="xsd:string"/>
                                    <xsd:element name="WriteAccess" type="xsd:string"/>
                                    <xsd:element name="Size" type="xsd:integer"/>
                                </xsd:sequence>
```

```
                    </xsd:complexType>
                </xsd:element>
</xsd:schema>
```

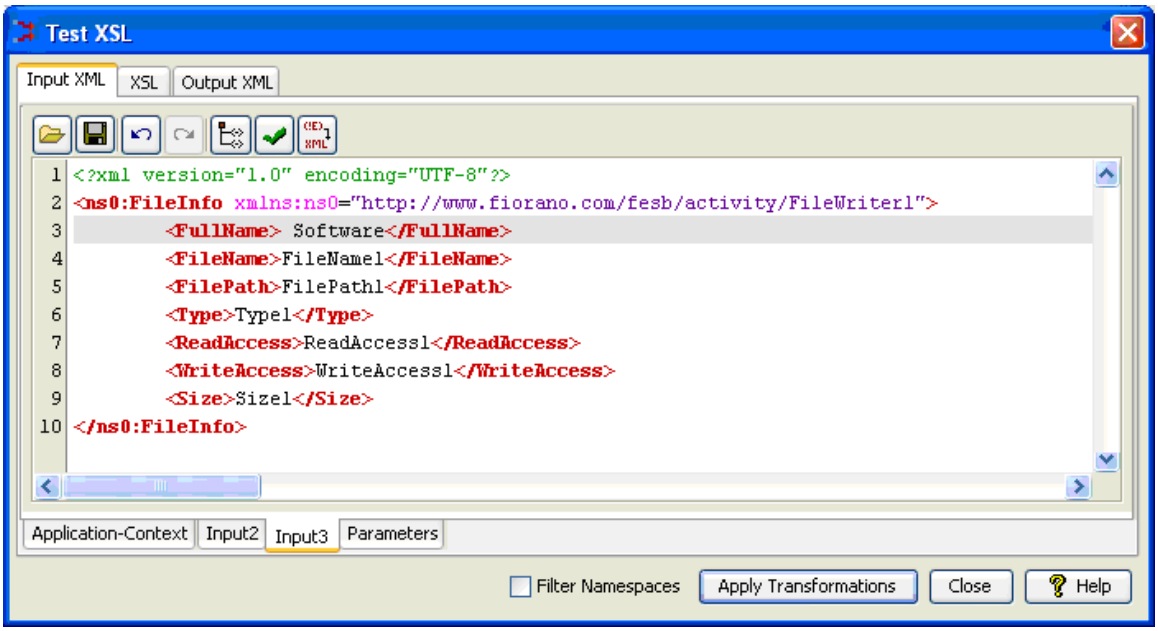


Figure 4: Sample Join input 2 message

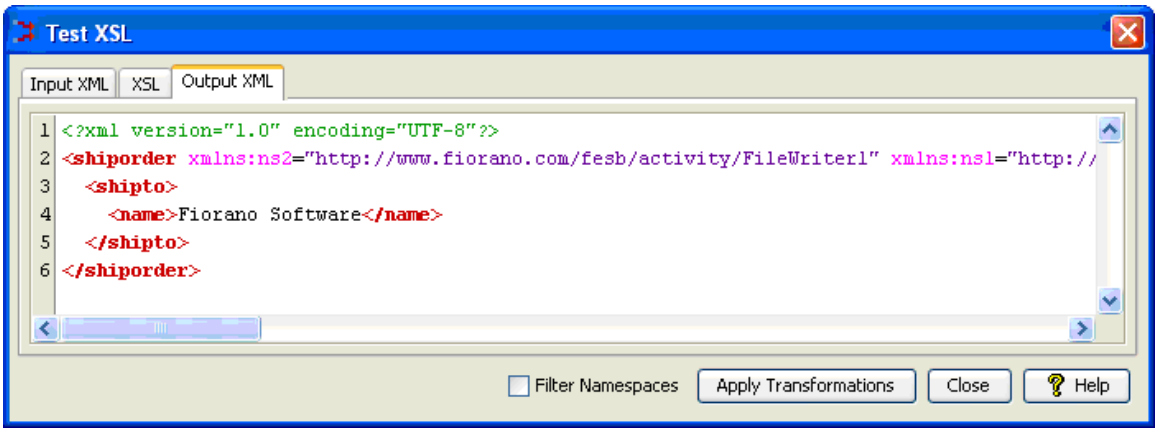


Figure 5: Sample Join output message

4.6.4.3 Functional Demonstration

4.6.4.3.1 Scenario 1

Send two different messages for which mapping is configured in the Join component and displaying the response message.

Configure the Join as described in Configuration and Testing section and use Feeders and Display component to send sample input and check the response.

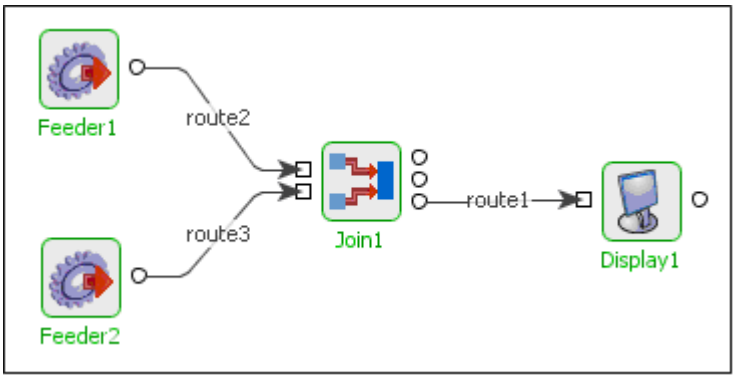


Figure 6: Demonstrating Scenario 1 with sample input and output

4.6.4.4 Sample Input

Input 1

```
<shiporder orderid="orderid">
   <orderperson>orderperson</orderperson>
   <shipto>
      <name>Fiorano</name>
      <address>address</address>
      <city>city</city>
      <country>country</country>
   </shipto>
   <item>
      <title>title</title>
      <note>note</note>
      <quantity>60</quantity>
      <price>-162</price>
   </item>
</shiporder>
```

Input 2

```
<ns1:FileInfo xmlns:ns1="http://www.fiorano.com/fesb/activity/FileWriter1">
   <FullName>Software</FullName>
   <FileName>FileName</FileName>
   <FilePath>FilePath</FilePath>
   <Type>Type</Type>
   <ReadAccess>ReadAccess</ReadAccess>
```

```
   <WriteAccess>WriteAccess</WriteAccess>
   <Size>-84</Size>
</ns1:FileInfo>
```

Sample Output

```
<?xml version="1.0" encoding="UTF-8"?>
<shiporder xmlns:ns2="http://www.fiorano.com/fesb/activity/FileWriter1"
xmlns:ns1="http://www.w3.org/2001/XMLSchema">
  <shipto>
    <name>Fiorano Software</name>
  </shipto>
</shiporder>
```

4.6.4.5 Useful Tips

- It is advised to configure the components connected to IN_PORT1, IN_PORT2 and OUT_PORT_RESULT of the Join component before configuring Join component. This allows join component to pick input and output structures appropriately. Input and output structures can also be provided in the CPS of Join. But, to make sure there are no schema mismatches after configuring all components, it is suggested to configure the components connected to this component before Join is configured
- More than three input structures (including Application Context) and one output structure cannot be added.
- Only after Join component receives at least one input message on each input port, it will perform the join operation and sends message onto output ports. If ten messages are received on the first input port and five on the second input port or vice versa, then the join is performed five times.

4.6.5 Sleep

The Sleep component is used to induce a specified delay in the flow of execution of an event process. The component holds the message in memory for the said amount of time and sends it to the output port.

Note: If the property Sleep after Every Message is set to yes, then every message received sleeps for the specified duration. If the property is set to no, the message will only sleep for (specified interval – (current time – message received time))

4.6.5.1 Configuration

Sleep component has the following properties in its Custom Property Sheet.

Sleep Interval Time Unit - Specifies the unit in which the Sleep Interval time duration is specified.

Sleep Interval - Specifies the time duration after which the flow of execution is allowed to leave this component.

Sleep after Every Message - Specifies whether a sleep delay is to be induced after every message is received.

Input/Output XSD - Specifies the XSD schema for the IN_PORT / OUT_PORT ports. If you provide the XSD, it is only set on the ports so that it can be used during transformations. This is not used by the component during execution.

4.6.5.2 Functional Demonstration

4.6.5.2.1 Scenario 1

Simple flow demonstrating the basic functionality of Sleep.

Configure the Sleep as described in *Configuration* section and use feeder and display component to send sample input and to check the response respectively.

The Sleep adapter is configured to sleep after every message for an interval of 60,000 milliseconds. Check the time of the messages in Feeder and Display components in the attached screenshots.

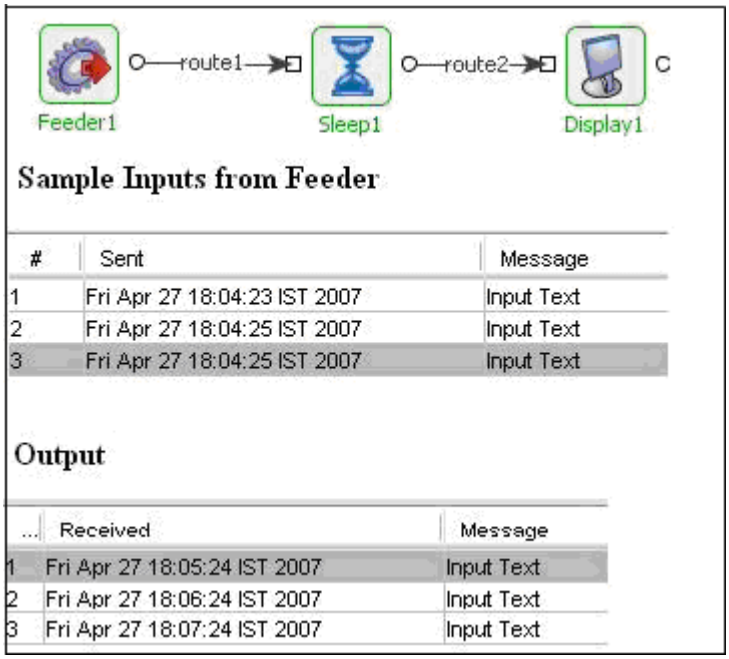


Figure 4.291: Scenario demonstration with sample input and output

4.6.5.3 Useful Tips

If the property Sleep after Every Message is set to **yes**, then every message received sleeps for the specified duration. If the property is set to **no**, the message only sleeps for specified interval - (current time - message received time).

4.6.6 Timer

The Timer component is used to trigger sending of messages to a component connected to its output ports. The date and the time at which this component needs to start sending messages can be configured. The number of messages and the format of the message can also be specified.

This component has no input ports, but 2 output ports:

- Message Port
- Timer Port

Timer component is capable of sending messages with the format specified in the Custom Property Sheet to its Message port. Sending messages which contains the date and time to its Timer Port. The component uses java.util.Timer class for scheduling.

Points to note

If the Timer component is configured for a start date and time which is in past when the component starts, all messages which could have been sent had the component started at the configured date and time, is sent immediately on startup. To avoid this, the component can be configured to send the first message at the first interval which comes in the future of the time when the component launched. The configuration parameter to check is Start execution from next interval.

4.6.6.1 Configuration and Testing

4.6.6.1.1 Scheduler Configuration

The Scheduling information can be configured in Scheduler Configuration panel.

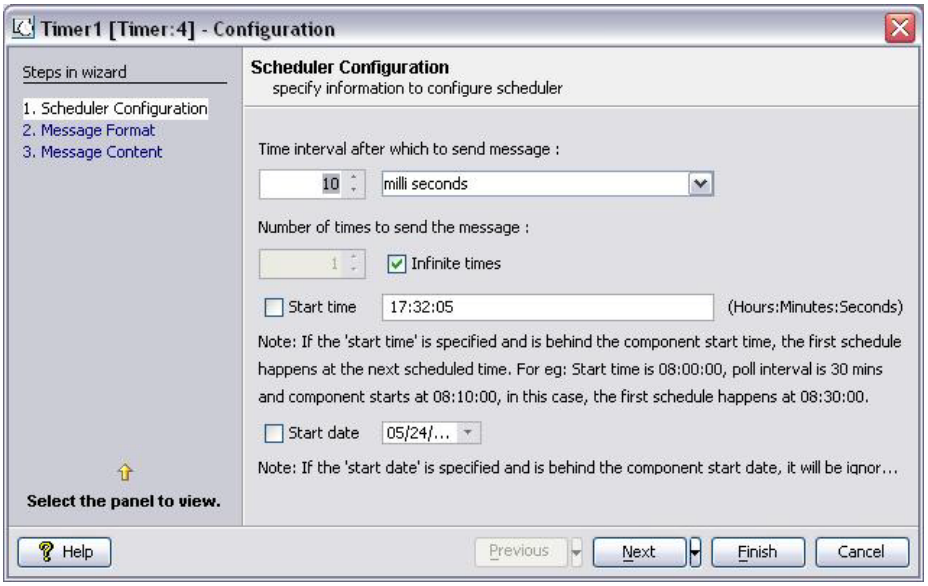


Figure 4.292: Scheduler Configuration Panel

In this panel, the following parameters are configured:

Start Time – The Time at which the Timer has to start sending the messages. This can also be configured for a future time or for the past time.

- Interval between consecutive messages.
- Number of messages to be sent.
- Start Date.

4.6.6.1.2 Message Format

The output message format for the Message Port can be configured in this panel. Timer supports plain text format and XML format messages.

If you select the message format as XML, then you need to provide sample XSD in this panel.

4.6.6.1.3 Message Content

This panel is used to generate the sample message which has to be sent by the Timer service.

If you configure to send the XML message in Message Format Panel, then you can generate the message in XML by using the **Generate Sample XML** button.

If the timer is configured to send Plain Text message, then the content can be specified in the text area.

4.6.6.2 Output Schema

The messages that are sent from the Timer Port have the following schema:

Schema	Description
<Timer>	Root element of the schema
<time>	The number of milliseconds since January 1, 1970
<second>	Seconds
<minute>	Minutes
<hour>	Hour
<dayOfWeek>	Day of the week(Eg: 1 for Sunday)
<dayOfMonth>	Day of month
<weekOfMonth>	Week of month
<weekOfYear>	Week of year
<month>	Month
<year>	Year
<date>	Date

4.6.6.3 Functional Demonstration

4.6.6.3.1 Scenario 1

Sending messages on its output ports when the timer clicks.

Configure the Timer as described in, **Configuration** section to send sample XML message to its Message Port and Timer Port after every 10 seconds after the component launch.To display components are connected to Timer output ports to receive the sample output.

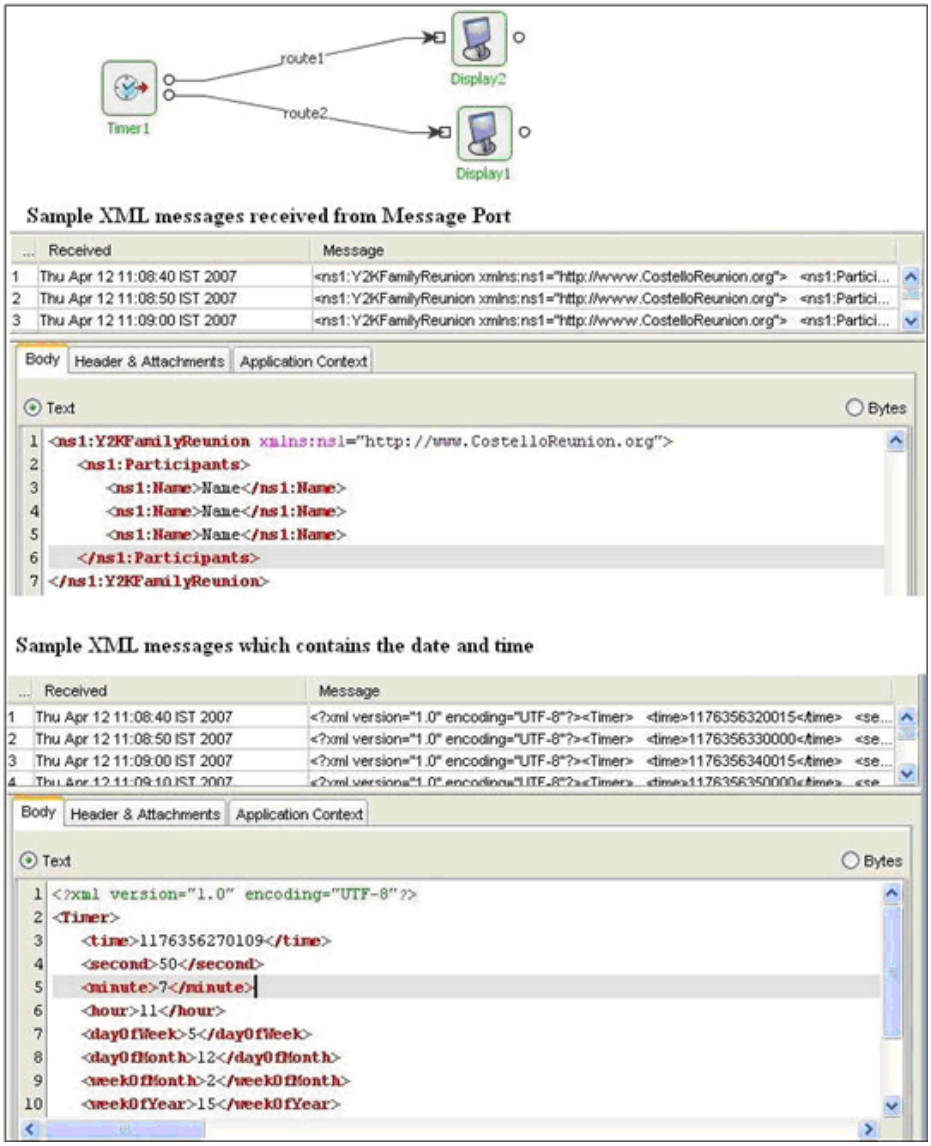


Figure 4.293: Scenario demonstration

4.6.6.4 Useful Tips

If the Timer component is configured for a start date and time which is in past when the component starts, all messages which could have been sent had the component started at the configured date and time, are be sent immediately on startup. To avoid this, the component can be configured to send the first message at the first interval which comes in the future of the time when the component launched. The configuration parameter to check is Start execution from next interval.

4.6.7 WorkList

The Work List component enables you to define manual intervention based business processes to define rules to perform time-based or role-based escalation of designated tasks.

Every user can use the web interface to access the designated tasks and perform the action as required and available based on the role.

Points to note

- To use the web interface, please start the ESB Web Container and access http://localhost:1980/worklistexplorer.
- All WorkList components should be running on the same peer as the WorkListManager component.
- Changing the Node Name at runtime for Worklist and Aggregator Components is not supported. Unlike other components, Worklist and Aggregator components have state information written to the local disk.Moving the Worklist (or Aggregator) from one peer server to another one results in state data loss. In case of Worklist, not only the data loss, the external application, that is, Worklist web application will not show work items saved in the Worklist after the node change.
- To achieve high availability for Stateful components, configure the back-end data store in clustered/HA relational database, like Oracle, DB2, and so on. Or deploy the components on a Peer Server that is running in Shared HA mode.
- Character encoding of web pages can be changed in %FIORANO_HOME%\esb\server\jetty\fes\webapps\worklistexplorer\WEB-INF\web.xml. By default this value is set to UTF-8 in web.xml as shown in following fragment of web.xml

```
<filter>
   <filter-name>EncodingFilter</filter-name>
   <filter-class>worklistexplorer.EncodingFilter</filter-class>
       <init-param>
     <param-name>encoding</param-name>
     <param-value>UTF-8</param-value>
   </init-param>
 </filter>
```

4.6.8 WorkList Manager

The WorkListManager component is used to manage one or more instances of the WorkList component. This component offers a web-based interface to enable you to track your set of tasks as well as administer various task specific activities and escalate or re-assign them based on individual requirements.

This is a non-configurable component and acts as the Manager of all WorkList components part of an Event Process. An instance of this component is used in the WorkListManager Sample Event Process.

Points to note

- To use the web interface, please start the ESB Web Container and access http://localhost:1980/worklistexplorer.
- Running more than one instance of this component is not advisable.
- This component should be running on the same peer server where all the WorkList components are running.
- Character encoding of web pages can be changed in %FIORANO_HOME%\esb\server\jetty\fes\webapps\worklistexplorer\WEB-INF\web.xml. By default this value is set to UTF-8 in web.xml as shown in following fragment of web.xml

```
<filter>
   <filter-name>EncodingFilter</filter-name>
   <filter-class>worklistexplorer.EncodingFilter</filter-class>
       <init-param>
     <param-name>encoding</param-name>
     <param-value>UTF-8</param-value>
   </init-param>
 </filter>
```

4.6.9 XMLSplitter

XMLSplitter can be used to split XML documents based on the configured XPath. This component is useful when there are repeated elements in the documents that can be processed by independently by subsequent components.

4.6.9.1 Configuration and Testing

The following attributes can be configured in the interaction configuration panel.

Attributes	
Schema	Click ... to specify Schema
Namespaces	{}
XPath	/
Operation	Split
Processor	XPath
Output Schema	Click ... to specify Schema
Read XML from file?	**yes**
XML File Directory	
XML File Name	null
Cleanup resources (excluding c...	yes
Target Namespace	http://www.fiorano.com/fesb/activity...
Monitoring configuration	disabled: 5000 milli seconds
Store imported schemas	no

Figure 1: Interaction Configuration Properties

4.6.9.1.1 Attributes

4.6.9.1.1.1 Schema

The XSD of the input XML message has to be specified here. The XSD can be provided using schema editor which opens up on clicking ellipsis button against this property. If the schema has any namespaces then they will be automatically populated in the Namespaces table shown in figure 2.

4.6.9.1.1.2 Namespaces

Namespace prefixes that are used instead of complete namespace in XPath expression can be specified by clicking the ellipsis button against this property which opens a Namespaces table as shown in the figure 2. The namespaces present in the input schema, if any, are automatically populated in the table. If the user wish to provide XPaths manually and wish to use namespaces which are not present in the schema provided, they can be added using the namespaces table.

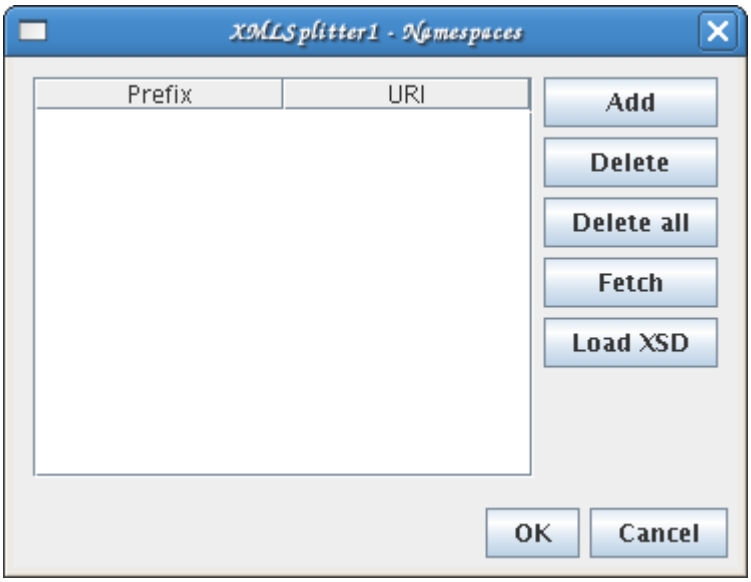


Figure 2: Namespace Table

- **Prefix**

  The prefix with which a given namespace is identified. Prefix fsoa is reserved.

- **URI**

  The URI of the namespace.

Operations that can be performed in the namespace table:

- **Add**

  Namespaces present in the schema provided against property **Schema** are populated by default in the table. To use any namespaces that are not present in that schema, this option can be used. When the add button is clicked, a new namespace will be added with default prefix and URI. The columns are editable and thus an appropriate value can be specified in place of the default values.

- **Delete**

  Existing namespaces can be deleted from the table using this option. When namespaces are fetched from connected components or some XSD, there is a possibility that namespaces with same URI are added with different namespace prefix. In such cases, redundant namespaces can be deleted. This will not affect the schemas in which the namespaces are present.

- **Delete all**

  Deletes all the namespaces in the table.

- **Fetch**

  Fetches namespaces from the schemas present on ports of other components connected to XMLSplitter component. Thus it is advisable to configure the connected components completely before using this option.

- **Load XSD**

  Loads namespaces from the schema which is provided in the text editor opened when this button is clicked. This option can be used when there are XSDs whose namespaces are required for configuration. One schema must be loaded a time. The schemas are not stored in the component.

4.6.9.1.1.3 XPath

The XPath of the element based on which the split or group operation has to be performed is configured here. Click the ellipsis button to open XPath editor. Choose an element/attribute from the list displayed in the left side panel of XPath editor and drag it onto the easel on the right side. The configured XPath expression must always evaluate **an element or an attribute**. XPath expressions that evaluate to any other types are not valid.

Note: There is no validation check for the XPath provided at configuration time, so the user has to configure XPath to return element.

4.6.9.1.1.4 Operation

The operation that has to be performed on the input XML message.

- **Split**

  Splits the input XML at XPath defined and sends out output XMLs. The number of output XMLs is equal to the number of times the element/attribute defined by the property **XPath** is present in the input message.

  '**Processor**' property is visible when this option is selected and '**Root Element Name**' and '**Root Element Namespace**' are hidden.

  Note: when the XPath is specified as an attribute, then the value of the attribute is sent as the output message.

  Example: If the input contains all the Employees details conforming to schema as shown in figure 3 and if it is required to split individual Employee details into separate messages then configure **'XPath'** to ns1:Employee_Schema/ns1:Employee and select '**Operation'** as Split. Refer to scenario 1 under Functional demonstration section.

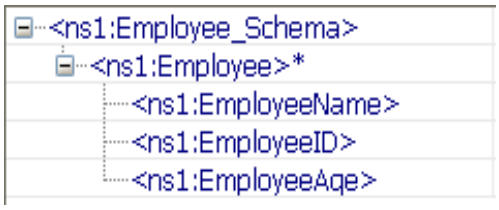


Figure 3: Schema for Employees details

- **Group**

  Splits the input XML at the element whose XPath is specified by the property **XPath** and then regroups the split XMLs which have the same value for that element into a single message. Thus the number of messages sent onto the output port depends on number of unique values present in XML for the element whose XPath is specified by the property **XPath**.

  '**Root Element Name**' and '**Root Element Namespace**' properties are visible when this option is selected and '**Processor**' property is hidden. XSLT Processor is used for this operation.

  Note: When an element is selected using property **XPath** then those elements having same value will be grouped. If an attribute is selected as **XPath** then those **elements** for which this attribute is defined and having same value for this attribute will be grouped.

  Example: If the input contains all the Employees details conforming to schema shown in figure 4 and if it is required to group individual employee details which are in the same group. (group information is stored in the attribute Group of Employee element), then configure **'XPath'** to ns1:Employee_Schema/ns1:Employee/@Group and choose property '**Operation'** as Group. Employee elements whose group attribute is same will be combined into a single message.

Figure 4: Schema for Employees details with a group attribute

  - Root Element Name: Root Element Name for the output XML(s). This property will be visible when the **Operation** is selected as Group.
  - Root Element NameSpace: Namespace for the root element in output XML(s). If the default value is selected, then the root element namespace will be same as the target namespace of the input XSD provided. This property will be visible when the **Operation** is selected as Group.

4.6.9.1.1.5 Processor

The Processor to be used for splitting.

- **Xpath**

  Uses XPath Processor for splitting.

- **XSLT**

  Uses XSLT for splitting. When the '**Operation'** is '**Group**', XSLT processor will be used for processing, so this property will not be visible in that case.

  Note: Prefer XSLT for simple split Paths and XPath for complex paths. All kinds of split paths may not be supported by XSLT.

4.6.9.1.1.6 Output Schema

Schema for the output message can be specified. Schema can be specified exclusively or can be generated with the help of input schema and XPath by clicking **Get schema based on input and XPath** button in the schema editor. This is not guaranteed to give a valid schema always. Please verify when using this feature.

4.6.9.1.1.7 Read XML From File?

The XML can be read directly from the specified file by setting this property as yes. 'XML File Directory' and 'XML File Name' properties are visible when this option is set to yes.

4.6.9.1.1.8 XML File Directory

Directory of the input XML file.

4.6.9.1.1.9 XML File Name

Name of the input XML file which has to be split.

4.6.9.2 Sample Input and Output

The configuration can be tested by clicking the **Test** button in the interaction Configuration panel.

Figure 5: Sample Input Message

```
<?xml version="1.0" encoding="UTF-8"?><nsl:Author xmlns:ns1="http://www.books.org">
Author</nsl:Author>
```

Figure 6: Response Generated for XPath /ns1:BookStore/ns1:Book/ns1:Author

4.6.9.3 Functional Demonstration

4.6.9.3.1 Scenario 1

Splitting the input XML with respect to an element.

Configure the XMLSplitter as shown in Figure 7.

Attributes	
Schema	Click ... to edit Schema
Namespaces	{ns2=http://www.w3.org/2001/XMLSche...
XPath	**/ns1:BookStore/ns1:Book/ns1:Author**
Operation	Split
Processor	XPath
Output Schema	Click ... to edit Schema
Monitoring configuration	disabled: 5000 milli seconds

Figure 7: configuration for scenario 1

Use feeder and display component to send sample input and to check the response respectively. In the example given below, the split element selected is **Author**.

Sample Input

```
<ns1:BookStore xmlns:ns1="http://www.books.org">
   <ns1:Book Category="non-fiction" InStock="false" Reviewer=" ">
      <ns1:Title>Title</ns1:Title>
      <ns1:Author>Author</ns1:Author>
      <ns1:Date>Date</ns1:Date>
      <ns1:ISBN>ISBN</ns1:ISBN>
      <ns1:Publisher>Publisher</ns1:Publisher>
   </ns1:Book>
      <ns1:Book InStock="false" Reviewer=" ">
      <ns1:Title>Title</ns1:Title>
      <ns1:Author>Author1</ns1:Author>
      <ns1:Date>Date</ns1:Date>
      <ns1:ISBN>ISBN</ns1:ISBN>
      <ns1:Publisher>Publisher</ns1:Publisher>
   </ns1:Book>
      <ns1:Book Category="non-fiction" InStock="false" Reviewer=" ">
      <ns1:Title>Title</ns1:Title>
      <ns1:Author>Author1</ns1:Author>
      <ns1:Date>Date</ns1:Date>
      <ns1:ISBN>ISBN</ns1:ISBN>
      <ns1:Publisher>Publisher</ns1:Publisher>
   </ns1:Book>
</ns1:BookStore>
```

Sample Output

Output 1

```
<?xml version="1.0" encoding="UTF-8"?><ns1:Author xmlns:ns1="http://www.books.org">Author</ns1:Author>
```

Output 2

```
<?xml version="1.0" encoding="UTF-8"?><ns1:Author xmlns:ns1="http://www.books.org">Author1</ns1:Author>
```

Output 3

```
<?xml version="1.0" encoding="UTF-8"?><ns1:Author xmlns:ns1="http://www.books.org">Author1</ns1:Author>
```

Figure 8: Scenario demonstration with sample input and output

4.6.9.3.2 Scenario 2

Grouping the input XML based on the XPATH provided.

Configure the XMLSplitter as described in *Configuration and Testing* section. The configuration for this example is shown below. In the example given below, the xpath element selected is Author.

Observe the two outputs shown in below figure

Attributes	
Schema	Click ... to edit Schema
Namespaces	{ns2=http://www.w3.org/2001/XMLSchema, ns1=http://www.books.org}
XPath	**/ns1:BookStore/ns1:Book/ns1:Author**
Operation	Group
Root Element Name	Root
Root Element NameSpace	http://www.fiorano.com/xmlsplitter/result0
Output Schema	Click ... to edit Schema

Figure 9: Configuration Properties panel

```
<ns1:BookStore xmlns:ns1="http://www.books.org">
   <ns1:Book>
      <ns1:Title>Title</ns1:Title>
      <ns1:Author>Author1</ns1:Author>
      <ns1:Date>Date</ns1:Date>
      <ns1:ISBN>ISBN</ns1:ISBN>
      <ns1:Publisher>Publisher</ns1:Publisher>
   </ns1:Book>
   <ns1:Book>
      <ns1:Title>Title</ns1:Title>
      <ns1:Author>Author</ns1:Author>
      <ns1:Date>Date</ns1:Date>
      <ns1:ISBN>ISBN</ns1:ISBN>
      <ns1:Publisher>Publisher</ns1:Publisher>
   </ns1:Book>
   <ns1:Book>
      <ns1:Title>Title</ns1:Title>
      <ns1:Author>Author1</ns1:Author>
      <ns1:Date>Date</ns1:Date>
      <ns1:ISBN>ISBN</ns1:ISBN>
      <ns1:Publisher>Publisher</ns1:Publisher>
   </ns1:Book>
</ns1:BookStore>

Sample Output:

Output1:

<?xml version="1.0" encoding="UTF-8"?>
     <fsoa:Root xmlns:ns1="http://www.books.org"
           xmlns:fsoa="http://www.fiorano.com/xmlsplitter/result0"
           xmlns:ns2="http://www.w3.org/2001/XMLSchema">
           <ns1:Author>Author1</ns1:Author>
           <ns1:Author>Author1</ns1:Author>
      </fsoa:Root>
Output2:

<?xml version="1.0" encoding="UTF-8"?>
     <fsoa:Root xmlns:ns1="http://www.books.org"
           xmlns:fsoa="http://www.fiorano.com/xmlsplitter/result0"
           xmlns:ns2="http://www.w3.org/2001/XMLSchema">
           <ns1:Author>Author</ns1:Author>
      </fsoa:Root>
```

Figure 10: Sample input and output for Scenario 2

4.6.9.4 Use Case Scenario

In the Bond Trading sample Event Process, XML splitter is used to split the Isin data into individual Isin elements.

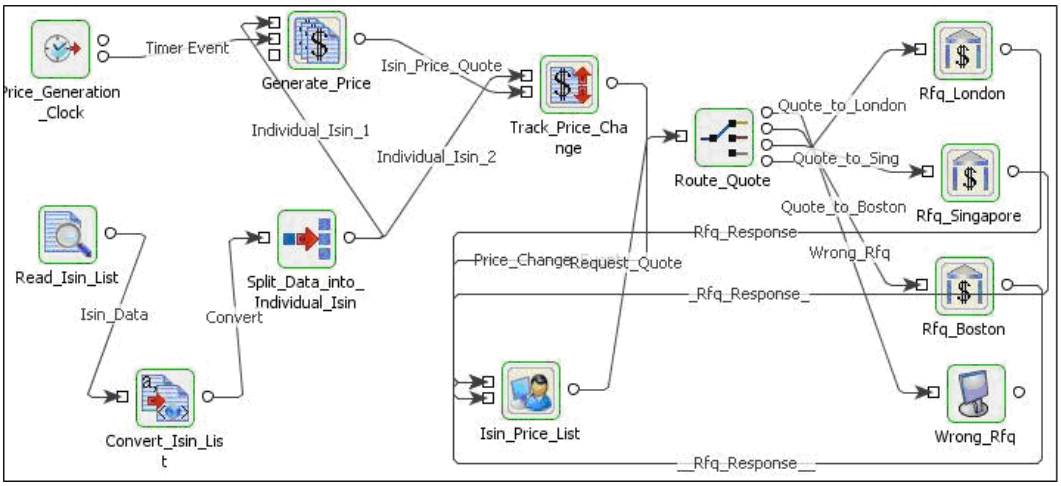


Figure 11: Sample use case scenario

The event process demonstrating this scenario is bundled with the installer.

Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Studio.

4.6.9.5 Useful Tips

- The output schema can be computed from the input schema and the XPath used to split the XML document using the **Get Schema based on Input and XPath** button in the schema editor for property **OutputSchema**. This is not guaranteed to give a valid schema always. Please verify when using this feature.
- Prefer XSLT for simple split paths and XPath for complex paths. All kinds of split paths may not be supported by XSLT.
- When component configuration sends multiple messages, messages contain the following JMS properties to identify first and last messages.
    - First document - START_EVENT=true
    - All documents - RECORD_INDEX=<index of output message>
    - Last document - CLOSE_EVENT=true
- When the input XML does not contain the element specified at Xpath, splitting/grouping is not performed and there will be no output messages in this case.
- The output generated has the same instance id for all the split elements. This causes issues when document tracking, if you try to track both documents on the output of the splitter, you only get a single entry (due to the duplicate instance ID's).
- To have different Instance Ids for different documents, workflow items in the flow have to be set properly.

- The Instance ID is set at the first "Workflow Item" encountered in the event process, so if the 1st workflow item is set before OUT_PORT of XMLSplitter then there will be only one "Instance ID" for all splitted messages, however all splitted messages can have different "Document ID". On the other hand if the first "Workflow Item" is set at OUT_PORT of XMLSplitter each element will have a different "Instance ID".

4.6.10 XMLVerification

The XMLVerification allows user to validate content present in the message body or application context or both, against configured XSD(s) or DTD(s).

The component has two output ports **OUT_PORT** and **FAILED_PORT**. If the validation is successful then the message "without any changes" is sent out on **OUT_PORT**, else the message is sent on **FAILED_PORT** with two additional properties **ERROR_MESSAGE** and **STACKTRACE** containing the error message indicating the problem and the source of problem.

Content present in message body or application context or both is considered valid only if

- The content has to be validated and is an XML instance.
- The structure of XML instance is valid according to corresponding XSD/DTD configured.
- The qualified root element of XML instance matches the configured qualified root element.

4.6.10.1 Configuration and Testing

The configuration for XML Verification is defined as shown in Figure 1

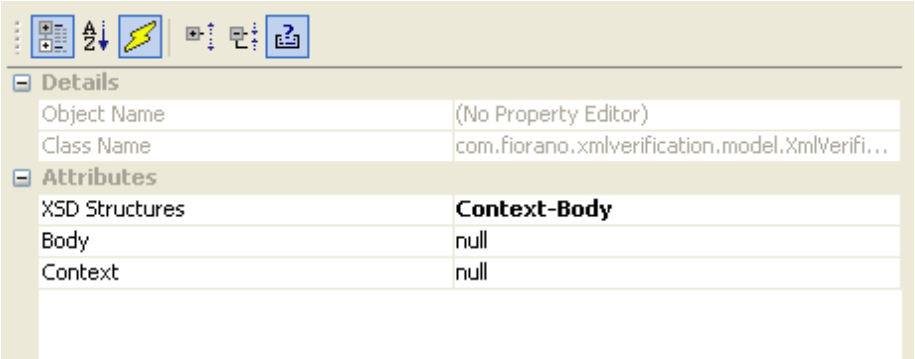


Figure 1: Configuration of XML Verification

4.6.10.1.1 Attributes

4.6.10.1.1.1 XSD Structures

This property determines the source(s) of content that should be validated

- **Body** - Only content of message body has to be validated with XSD/DTD defined against property Body. When this option is selected, property Body is visible and property Context is hidden.

- **Context** - Only content of application context has to be validated with XSD/DTD defined against property Context. When this option is selected, property Context is visible and property Body is hidden.
- **Context-Body** - Contents of both message body and application context have to be validated with XSD/DTD defined against properties Body and Context respectively. When this option is selected, both properties Context and Body are visible.

4.6.10.1.1.2 Body

This property defines the XSD/DTD with which content of message body has to be validated. Click **ellipsis** button to open the editor to provide XSD/DTD as shown in Figure 2

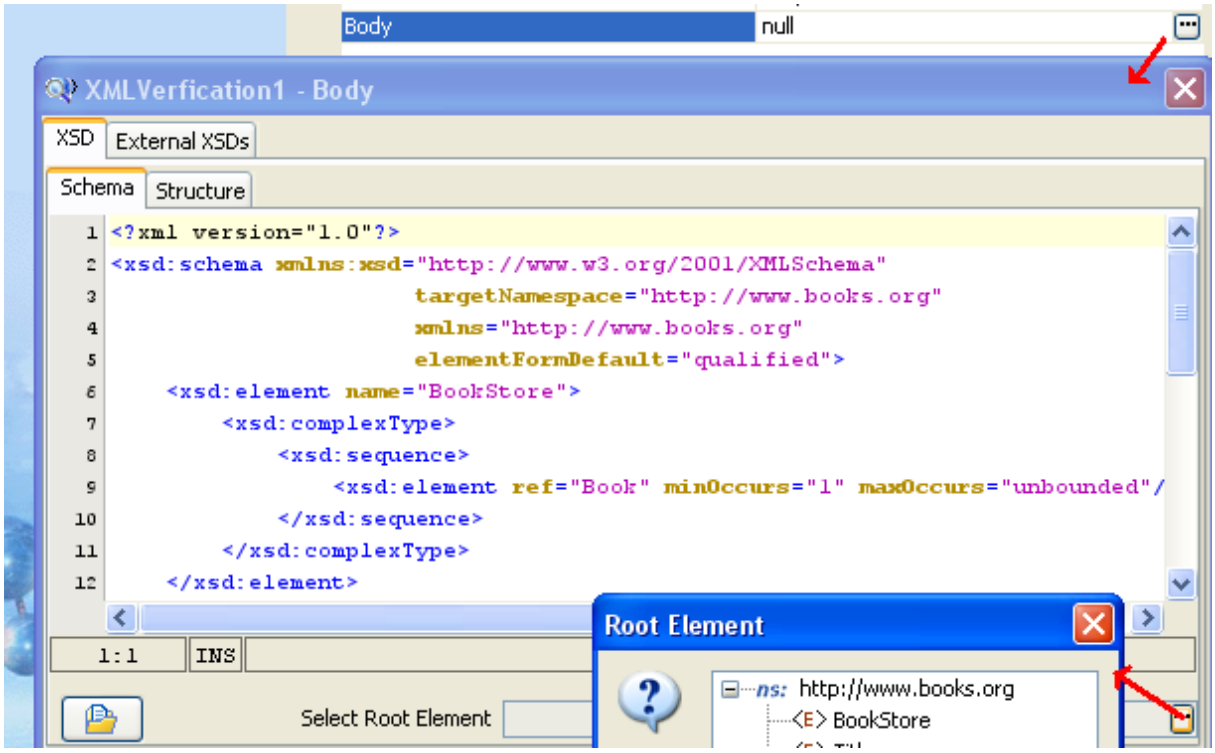


Figure 2: Defining XSD / DTD

4.6.10.1.1.3 Context

This property defines the XSD/DTD with which content of application context has to be validated. Click on ellipsis button to open the editor to provide XSD/DTD as shown in Figure 2.

4.6.10.2 Functional Demonstration

4.6.10.2.1 Scenario 1

Validation of input XML based on the schema provided.

Configure the XMLVerification as described in Configuration section for XSD Structure Body and use feeder and display components to send sample input and to check the response respectively. If the Sample Input verification is successful, then the input message is sent to **Display_Out** and if the verification fails, the input message is sent to **Display_False**.

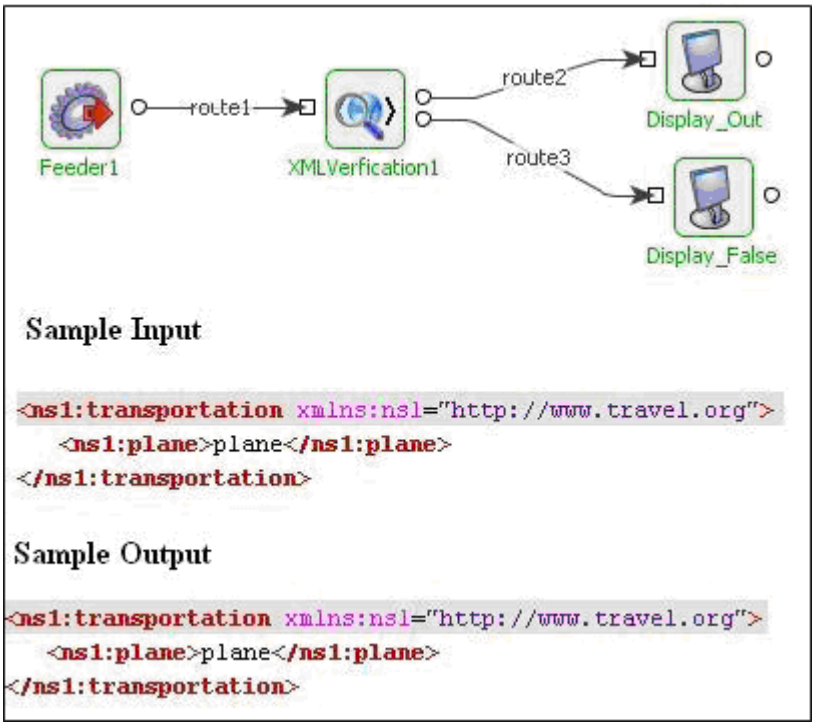


Figure 3: Demonstrating scenario with sample input and sample output

4.6.10.3 Useful Tips

- When the XSD defines multiple imported schemas containing same target namespace, only the first schema is used for validation and any elements defined in other schema are identified by the component.

4.6.11 Cache

Cache component holds data in an in-memory lookup table as a set of **name value pairs**, identified by one or more designated **key name value pairs**. A set of key fields and data fields are defined in the CPS of the component. The component stores values for the defined fields as **Name Value pairs**, where name is the name of the field and value is the data held in the field. A field can be a **key field** or a **data field**.

The component support the following operations

- **lookup** – returns all the data fields that are mapped for a given key field from the in-memory lookup table.
- **add** – adds a set of mappings from key field to data fields to the in-memory lookup table.

- **update** – updates a set of mappings that are defined for a key field in the in-memory lookup table.
- **delete** – deletes a set of mappings that are defined for a key field in the in-memory lookup table.

The component has the following ports:

- **ADD_PORT** – Input port ADD_PORT is used to send input request to perform add, update or lookup operations.
- **DEL_PORT** – Input port DEL_PORT is used to send input request to perform delete operations.
- **OUT_PORT** – Output port OUT_PORT is used to read output request that contains results of add, lookup, update or delete operations.

4.6.11.1 Configuration and Testing

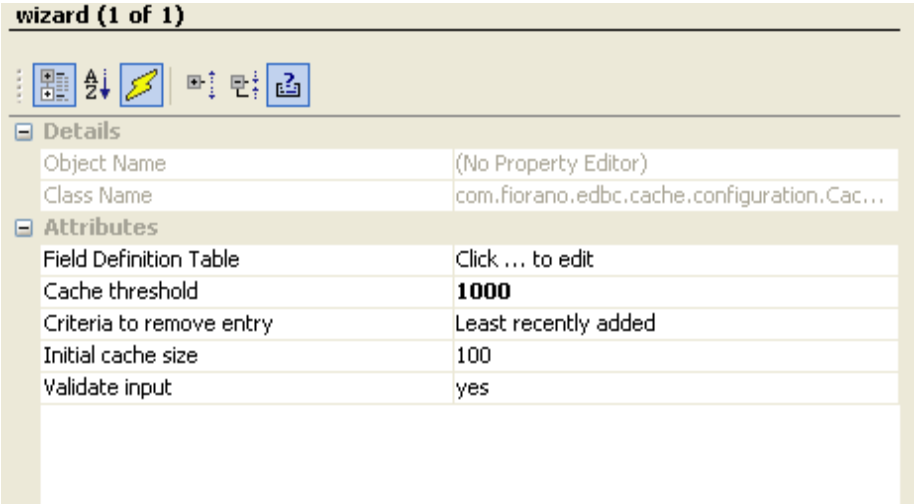


Figure 1: CPS of Cache component

4.6.11.1.1 Attributes

4.6.11.1.1.1 Field Definition Table

Click to launch **Field Definition Table** – an editor to define fields.

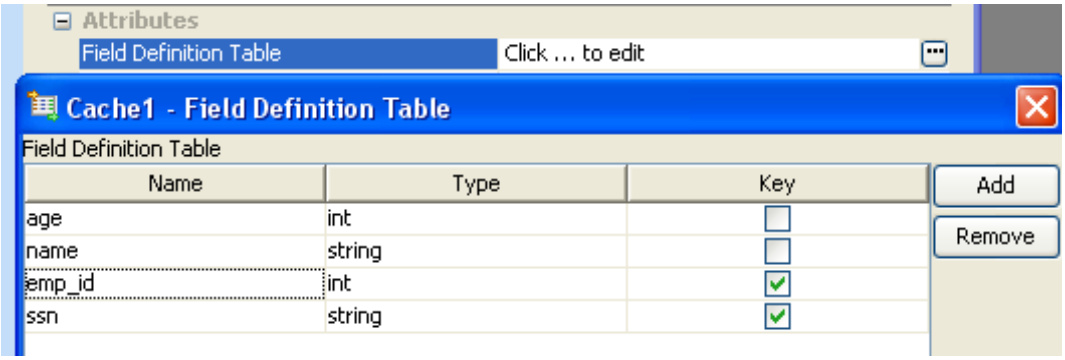


Figure 2: Editor to define fields in lookup table

The **Field Definition Table** contains **Name**, **Type** and **Key** fields.

- **Name** – Name of the field.
- **Type** – Data type of the field. Data type field contains the following field values – int, long, short, float, double, boolean, string and date.
- **Key** – Specifies whether the current field is a Key or a Data field.

To define field definitions –

1. Click **Add** button to add a new field definition.
2. Edit the value in **Name** column of the newly added row and provide the name of the field.
3. Select the data type of field value from the drop-down list in **Type** column.
4. Select the check box in **Key** column, if the field is a key field.
5. Repeat steps 1 to 4 to add all field definitions.
6. Click **Ok**.

Note:

- To delete any field definitions, select the field definitions and click **Remove**.
- There should be at least one key field and one data field defined in the **Field Definition Table**.

4.6.11.1.1.2 Cache threshold

This property determines the maximum number of entries that are held in the lookup table. If the number of entries in the lookup table exceeds the value defined by this property, oldest entries are discarded based on the property **Criteria to remove entry**.

Note:

- Value should be either -1 or greater than 0.
- -1 implies infinite threshold size.
- The property **Criteria to remove entry** is visible only when this value is greater than 0.

- Entry is removed from lookup table only after addition of an entry results in cache size exceeding the value defined by this property. The entry is removed in the same operation which adds the additional entry.

4.6.11.1.1.3 Criteria to remove entry

When a threshold is defined, this property determines which entry should be removed after the threshold is reached. Figure 3 shows all the available criteria.

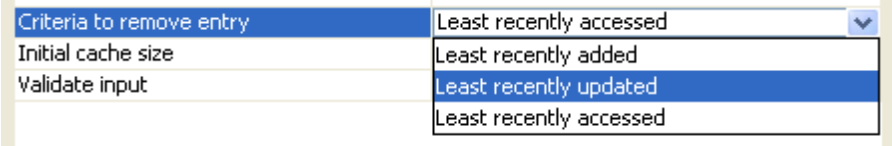


Figure 3: Criteria for removing entries after the threshold is reached

- **Least recently added** - The oldest entry in the lookup table will be removed first irrespective of the frequency at which an entry is used/updated.
- **Least recently updated** - The entry which has been in the lookup table for the longest duration without being updated will be removed.
- **Least recently accessed** – The entry has been in the lookup table for the longest time without being looked up or updated for the longest duration will be removed first.

4.6.11.1.1.4 Initial cache size

Initial size with which lookup table has to be initialized. The table will be resized depending on the underlying implementation. Providing an appropriate initial size will reduce the number of resizes.

4.6.11.1.1.5 Validate input

This property determines whether the input message has to be validated against the schema defined on the input ports.

- **Yes:** Input messages are validated against the schema defined on the input port on which the message is received.
- **No:** Input messages are not validated.

Note:

If the input message is not valid and the input validation is disabled, the behavior of the component is undefined.

4.6.11.1.2 Using Named Configurations

Changes are done in eStudio wrt to Named Configurations. In Component Configuration panel of Cache, Field Definition Table is modified to use Named Configurations.

4.6.11.2 Input and Output

The schema for input and output are auto generated based on the field definitions provided in the CPS.

4.6.11.2.1 Input

The structure of messages on **ADD_PORT** for the field definitions shown in Figure 2 is shown in Figure 4.

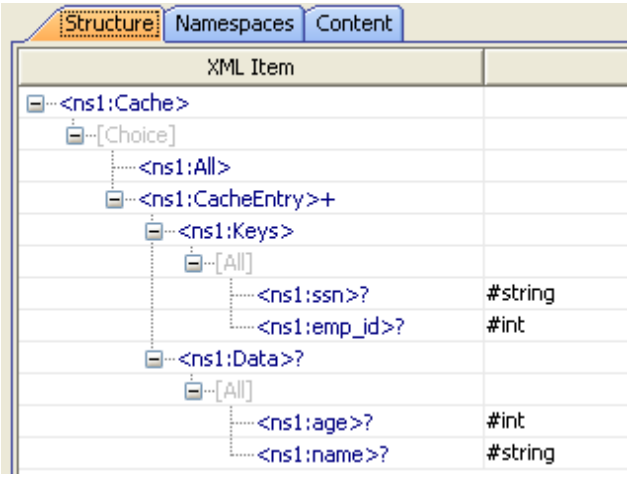


Figure 4: Structure of messages on ADD_PORT

Note: If the value is not defined for any of the keys, the behavior is not defined.

When a message is received on the **ADD_PORT** an entry is added, updated or looked up from lookup table based on the elements defined in the input message.

- An add operation is performed if **Data** element is present under **CacheEntry** element and at least one of the data fields is present and the value defined for a key field is not already present in the lookup table.
- An update operation is performed if **Data** element is present under **CacheEntry** element and at least one of the data fields is present and the value defined for a key field is already present in the lookup table.
- A lookup operation is performed if **Data** element is not present under **CacheEntry** element or none of the data fields are present.
- A lookup of all entries is performed if **All** element is present as defined in the structure.

The structure of messages on **DEL_PORT** for the field definitions shown in Figure 2 is shown in Figure 5.

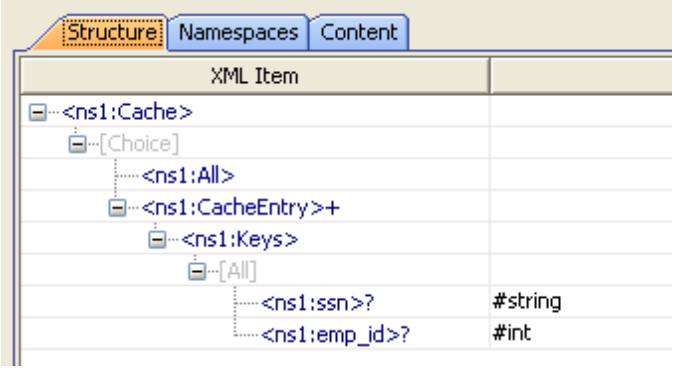


Figure 5: Structure of messages on DEL_PORT

Note: If the value is not defined for any of the keys, the behavior is not defined.

4.6.11.2.2 Output

Figure 6 contains the structure of messages on **OUT_PORT** for the field definitions shown in Figure 2.

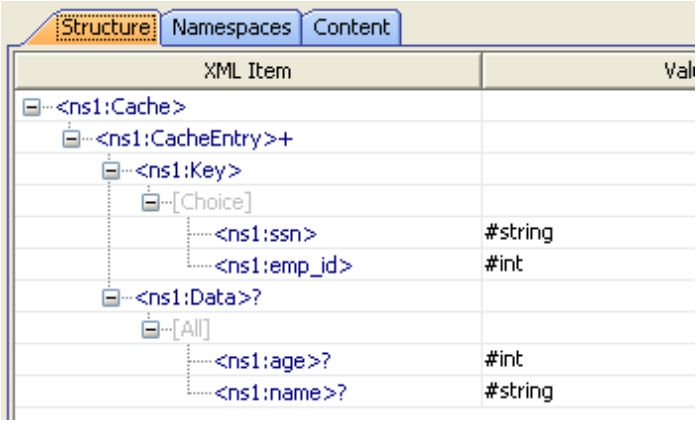


Figure 6: Structure of messages on OUT_PORT

4.6.11.3 Functional Demonstration

4.6.11.3.1 Scenario 1

This scenario demonstrates the functioning of the Cache component. Perform the steps shown in this scenario in strict order.

Configure the Cache component as shown in Figure 2 and create a flow as shown in Figure 7. Output port of **AddUpdateLookupRequest** should be connected to **ADD_PORT** and output port of **DeleteRequest** should be connected to **DEL_PORT**.

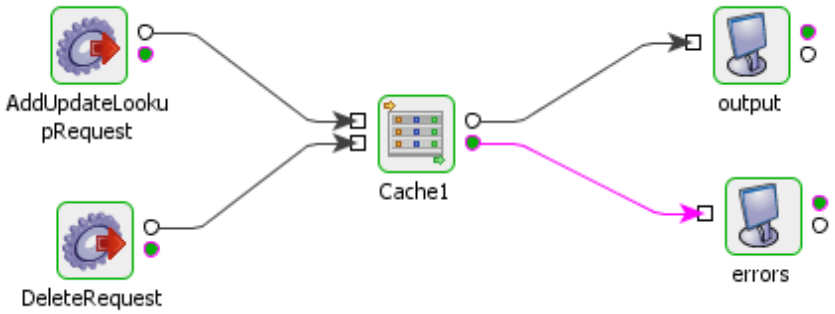


Figure 7: Sample flow to demonstrate scenario 1

4.6.11.3.2 Add operation

Send the input shown in Figure 8 from Feeder **AddUpdateLookupRequest** to add the following entries to lookup table.

- ssn:ssn1334 → age:29, name:name
- emp_id:88544 → age:29, name:name

Where an entry is in form <key field name>:<key field value> → <data field name>:<data field value>, <data field name>:<data field value>....

```
<ns1:Cache xmlns:ns1="http://www.fiorano.com/SOA/bc/cache">
   <ns1:CacheEntry>
      <ns1:Keys>
         <ns1:ssn>ssn1334</ns1:ssn>
         <ns1:emp_id>88544</ns1:emp_id>
      </ns1:Keys>
      <ns1:Data>
         <ns1:age>29</ns1:age>
         <ns1:name>name</ns1:name>
      </ns1:Data>
   </ns1:CacheEntry>
</ns1:Cache>
```

Figure 8: Input for add operation

The output message contains all the entries that are added as shown in Figure 9.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:Cache xmlns:tns="http://www.fiorano.com/SOA/bc/cache">
   <tns:CacheEntry>
      <tns:Key>
         <tns:ssn>ssn1334</tns:ssn>
      </tns:Key>
      <tns:Data>
         <tns:age>29</tns:age>
         <tns:name>name</tns:name>
      </tns:Data>
   </tns:CacheEntry>
   <tns:CacheEntry>
      <tns:Key>
         <tns:emp_id>88544</tns:emp_id>
      </tns:Key>
      <tns:Data>
         <tns:age>29</tns:age>
         <tns:name>name</tns:name>
      </tns:Data>
   </tns:CacheEntry>
</tns:Cache>
```

Figure 9: Result of add operation

4.6.11.3.2 Update operation

Send the input shown in Figure 10 from Feeder **AddUpdateLookupRequest** to update the name field for the entry with key field **ssn** whose value is ssn1334.

```
<ns1:Cache xmlns:ns1="http://www.fiorano.com/SOA/bc/cache">
   <ns1:CacheEntry>
      <ns1:Keys>
         <ns1:ssn>ssn1334</ns1:ssn>
      </ns1:Keys>
      <ns1:Data>
         <ns1:name>name1</ns1:name>
      </ns1:Data>
   </ns1:CacheEntry>
</ns1:Cache>
```

Figure 10: Input for update operation

The output message contains the updated entry as shown in Figure 11.

```
<tns:Cache xmlns:tns="http://www.fiorano.com/SOA/bc/cache">
   <tns:CacheEntry>
      <tns:Key>
         <tns:ssn>ssn1334</tns:ssn>
      </tns:Key>
      <tns:Data>
         <tns:name>name1</tns:name>
      </tns:Data>
   </tns:CacheEntry>
</tns:Cache>
```

Figure 11: Result of update operation

Entries in the database after the update operation are:

- ssn:ssn1334 → age:29, name:name1
- emp_id:88544 → age:29, name:name1

Note:

Since both key fields **ssn** and **emp_id** are defined under single **CacheEntry** element in the input of add operation shown in Figure 8, updating a data field value for entry corresponding to **ssn** key field implicitly updates the entry for **emp_id** key field as well.

In general, all the key fields defined under single **CacheEntry** maintain same reference to data fields and updating one effects the other. If this is not the desired behavior modify the input to contain to **CacheEntry** elements one for each key field as shown in Figure 12

```
<tns:Cache xmlns:tns="http://www.fiorano.com/SOA/bc/cache">
   <tns:CacheEntry>
      <tns:Key>
         <tns:ssn>ssn1334</tns:ssn>
      </tns:Key>
      <tns:Data>
         <tns:age>29</tns:age>
         <tns:name>name1</tns:name>
      </tns:Data>
   </tns:CacheEntry>
   <tns:CacheEntry>
      <tns:Key>
         <tns:emp_id>88544</tns:emp_id>
      </tns:Key>
      <tns:Data>
         <tns:age>29</tns:age>
         <tns:name>name1</tns:name>
      </tns:Data>
   </tns:CacheEntry>
</tns:Cache>
```

Figure 12: Input in Figure 8 modified to not maintain same reference for key fields

4.6.11.3.3 Lookup operation

Send the input shown in Figure 13 from Feeder **AddUpdateLookupRequest** to lookup the name and age fields for the entry with key field **ssn** whose value is ssn1334.

```
<ns1:Cache xmlns:ns1="http://www.fiorano.com/SOA/bc/cache">
   <ns1:CacheEntry>
      <ns1:Keys>
         <ns1:ssn>ssn1334</ns1:ssn>
      </ns1:Keys>
   </ns1:CacheEntry>
</ns1:Cache>
```

Figure 13: Input for lookup operation

The output message contains the entries looked up from lookup table as shown in Figure 14.

```
<tns:Cache xmlns:tns="http://www.fiorano.com/SOA/bc/cache">
   <tns:CacheEntry>
      <tns:Key>
         <tns:ssn>ssn1334</tns:ssn>
      </tns:Key>
      <tns:Data>
         <tns:age>29</tns:age>
         <tns:name>name1</tns:name>
      </tns:Data>
   </tns:CacheEntry>
</tns:Cache>
```

Figure 14: Result of lookup operation

To lookup all the entries in the lookup send, the input shown in Figure 15 from **AddUpdateLookupRequest**.

```
<ns1:Cache xmlns:ns1="http://www.fiorano.com/SOA/bc/cache">
   <ns1:All/>
</ns1:Cache>
```

Figure 15: Input for lookup of all entries

The output containing all the entries in the lookup table is shown in Figure 16

```
<tns:Cache xmlns:tns="http://www.fiorano.com/SOA/bc/cache">
   <tns:CacheEntry>
      <tns:Key>
         <tns:ssn>ssn1334</tns:ssn>
      </tns:Key>
      <tns:Data>
         <tns:age>29</tns:age>
         <tns:name>name1</tns:name>
      </tns:Data>
   </tns:CacheEntry>
   <tns:CacheEntry>
      <tns:Key>
         <tns:emp_id>88544</tns:emp_id>
      </tns:Key>
      <tns:Data>
         <tns:age>29</tns:age>
         <tns:name>name1</tns:name>
      </tns:Data>
   </tns:CacheEntry>
</tns:Cache>
```

Figure 16: Output of lookup of all entries

4.6.11.3.4 Delete operation

Send the input shown in Figure 17 from Feeder **DeleteRequest** to delete the entry with key field **ssn** whose value is ssn1334.

```
<ns1:Cache xmlns:ns1="http://www.fiorano.com/SOA/bc/cache">
   <ns1:CacheEntry>
      <ns1:Keys>
         <ns1:ssn>ssn1334</ns1:ssn>
      </ns1:Keys>
   </ns1:CacheEntry>
</ns1:Cache>
```

Figure 17: Input for delete operation

The output message contains the entries that are deleted from lookup table as shown in Figure 18.

```
<tns:Cache xmlns:tns="http://www.fiorano.com/SOA/bc/cache">
   <tns:CacheEntry>
      <tns:Key>
         <tns:ssn>ssn1334</tns:ssn>
      </tns:Key>
      <tns:Data>
         <tns:age>29</tns:age>
         <tns:name>name1</tns:name>
      </tns:Data>
   </tns:CacheEntry>
</tns:Cache>
```

Figure 18: Output for delete operation

Send the input shown in Figure 18 from Feeder **DeleteRequest** to delete all entries in lookup table.

```
<ns1:Cache xmlns:ns1="http://www.fiorano.com/SOA/bc/cache">
   <ns1:All/>
</ns1:Cache>
```

Figure 19: Input for deleting all entries

The output message contains the entries that are deleted from lookup table as shown in Figure 20. Since there is only undeleted entry in the database the output message contains only one entry.

```
<tns:Cache xmlns:tns="http://www.fiorano.com/SOA/bc/cache">
   <tns:CacheEntry>
      <tns:Key>
         <tns:ssn>ssn1334</tns:ssn>
      </tns:Key>
      <tns:Data>
         <tns:age>29</tns:age>
         <tns:name>name1</tns:name>
      </tns:Data>
   </tns:CacheEntry>
</tns:Cache>
```

Figure 20: Result of delete all entries operation

4.6.11.4 Useful Tips

- Size of the lookup table should be within in the memory limit available for the component.
- The lookup table should always be populated after the component starts as the lookup table is not persisted implicitly.
- Use this component in conjunction with a DB component to improve the performance of lookups and updates.
- Lookup can be performed using any of key fields which are part of add request.
- Update to any key field also effects the lookup data returned by other key fields which are part of add request.
- Disable input message validation to improve the performance of the component.
- Send **All** element in the input request on **ADD_PORT** to retrieve all the entries from the lookup table
- Send **All** element in the input request on **DEL_PORT** to delete all the entries from the lookup table
- When performing multiple add, update or lookup operation in a single input request on ADD_PORT the output contains corresponding result for each of the operation based on the state of lookup table when the operation is performed. If operations are performed in the order - add, lookup, add, lookup – the first lookup will contain only entry but the second lookup contain both the entries.

4.7 MOMs

The MOMs category consists of components like JMSIn 4.0 and 5.0, JMSOut 4.0 and 5.0, JMSReplier, JMsRequestor 4.0 and 5.0, MQSeriesIn, MQSeriesOut, MSMQReceiver, MSMQSender, TibcoRVIn, and TibcoRVOut. The following section describes each component.

4.7.1 JMSIn 4.0

The JMSIn component is used to transfer messages to a JMS topic or queue. Using the Configuration Property Sheet wizard, you can specify the topic or queue on which the message is to be sent. This component retrieves messages from a component and sends them to a JMS topic or queue. Additionally, you can create a Publisher or a Sender on Topic or Queue respectively, and configure the component to publish or send a message on a JMS Server at runtime.

This component can be used to send Text, Bytes or Map messages. The only restriction on Map Messages is that this component does not support Objects in Map Messages.JMSIn is capable of handling following types of messages:

Text messages

A TextMessage object's message body contains a java.lang.String object.

Map messages

A MapMessage object's message body contains a set of name-value pairs, where names are String objects, and values are Java primitives. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined.

Bytes Message

A BytesMessage object's message body contains a stream of uninterrupted bytes. This message type is for literally encoding a body to match an existing message format.

JMSIn uses JMS APIs to process the messages.

Points to note

- The only restriction on Map Messages is that this component does not support Objects in Map Messages.
- When adding the Initial Context Factory class for non Fiorano MQ server, the jar file(s) should be added as resource(s) to the JMSAdapters system library.

4.7.1.1 Configuration and Testing

The JMSIn component connection related properties can be configured in the Managed Connection Factory panel of CPS.

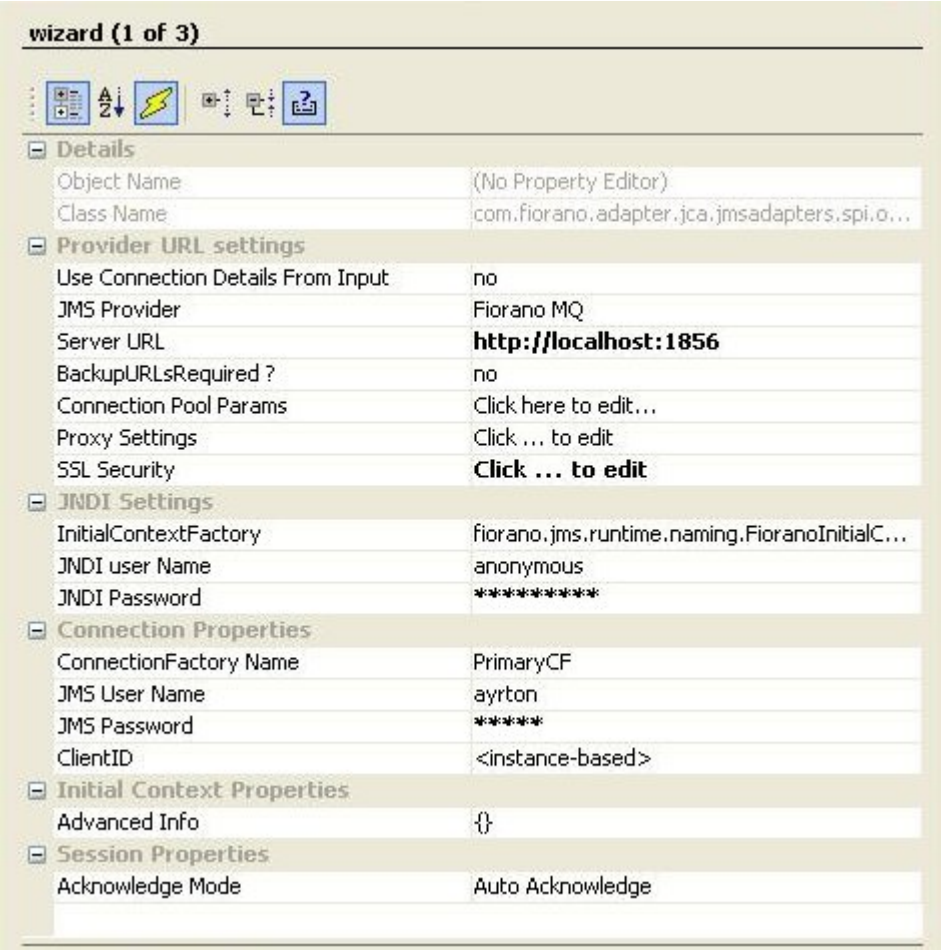


Figure 4.315: Sample JMSIn MCF configuration

Use Connection Details From Input: Parameters to create the connection can be specified in the input message when this property is set to true. If this property is selected the validation errors in the managed connection factory panel of the CPS are treated as warnings. So user can bypass this step without giving valid configuration and complete the configuration of the component. If valid properties are not provided even in the input message exception will be thrown at runtime.

The JMS providers supported are Fiorano MQ, BEA Weblogic, Oracle AQ and Oracle Streams AQ. For working with these JMS providers the jars that should be explicitly added as resources to the component are given below:

BEA WebLogic:

- %BEA_HOME%\server\lib\wlclient.jar
- %BEA_HOME%\server\lib\wljmsclient.jar (required only when the Weblogic server is running on a remote machine)

Note: For BEA Weblogic 10.0, %BEA_HOME% refers to <BEA WebLogic Installation directory>\wlserver_10.0

Oracle AQ:

- %ORACLE_HOME%\rdbms\jlib\aqapi13.jar (If JDK1.2 / JDK1.1 is used, aqapi12.jar/aqapi11.jar has to be used respectively)
- %ORACLE_HOME%\jdbc\lib\classes12.zip
- %ORACLE_HOME%\jdbc\lib\nls_charset12.jar

Note: For Oracle Database 9.2.0.1.0, %ORACLE_HOME% refers to <Oracle Installation directory>\ora92

Oracle Streams AQ:

- %ORACLE_HOME%\rdbms\jlib\aqapi.jar
- %ORACLE_HOME%\jdbc\lib\ojdbc14.jar

Note: For Oracle Database 10.2.0.1.0, %ORACLE_HOME% refers to <Oracle Installation directory>\product\10.2.0\db_1

Note the following:

- If these jars are added to resources of the System library JMSAdapters, the jars are available for JMSIn, JMSOut and JMSRequestor components.
- In case of BEA Weblogic, InitialContext can be created by specifying empty values for JNDI Username and JNDI password as well.

4.7.1.1.1 Using Named Configurations

Changes are done in eStudio w.r.t to Named Configurations. With the implementation of Named Configurations MCF panel is changed as shown in Figure below.

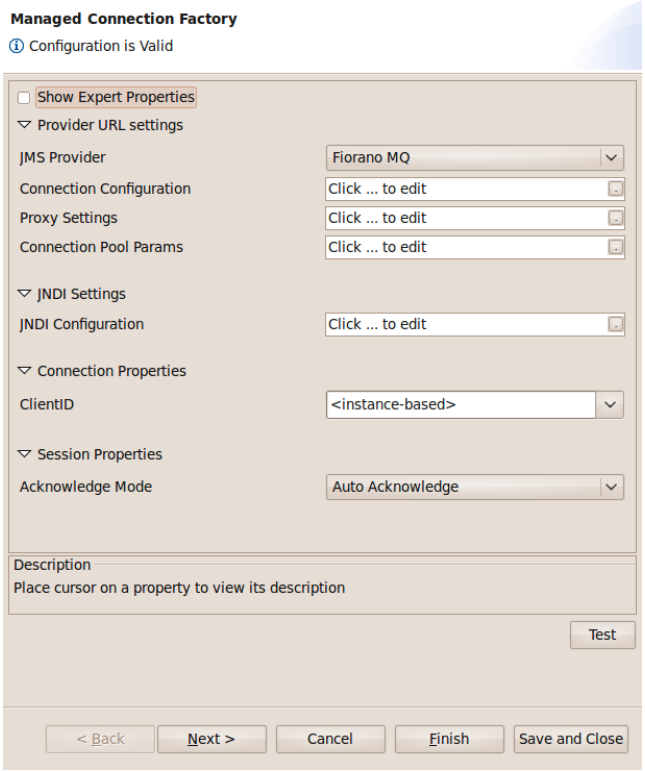


Figure: Modified Managed Connection Factory

Some properties are grouped as one property in the modified configuration panel. ServerURL, BackupURL, CF lookup name, JMS username, JMS password which were present previously in the MCF panel are grouped under the name ConnectionConfiguration.

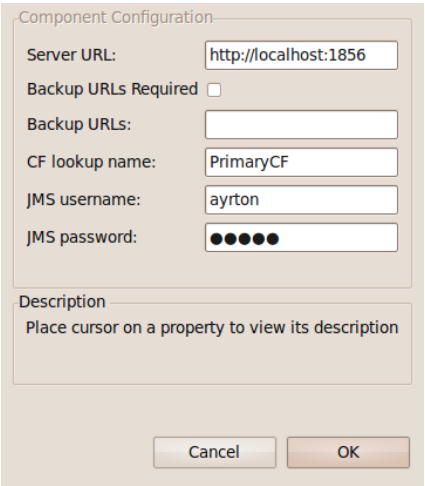


Figure: Connection Configuration

Initial Context Factory, JNDI username, CF lookup name, JNDI password which were present previously in the MCF panel are grouped under the name JNDI Settings.

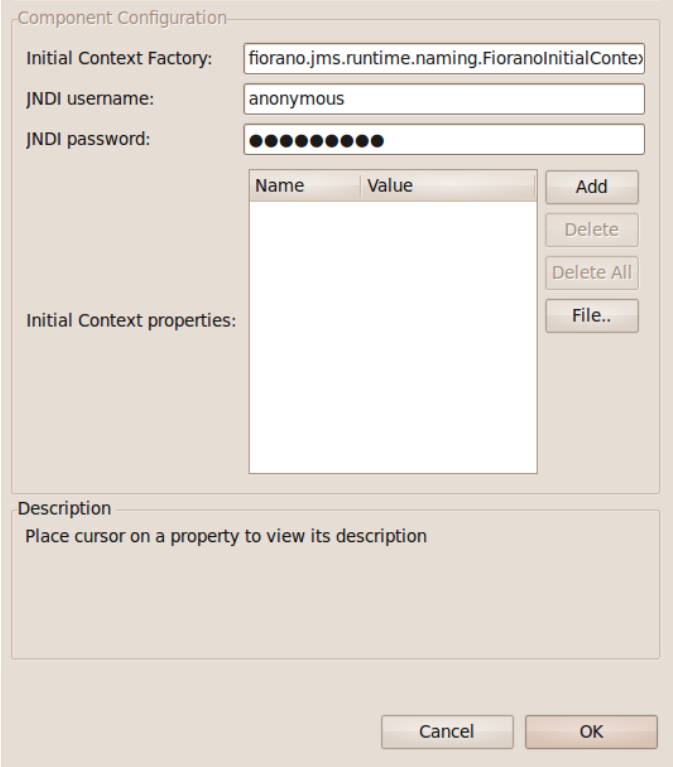


Figure: JNDI Settings

Destination Name, Destination Type, Lookup Destination which were present previously in the Interaction Configurations panel are grouped under the name Destination Configuration.

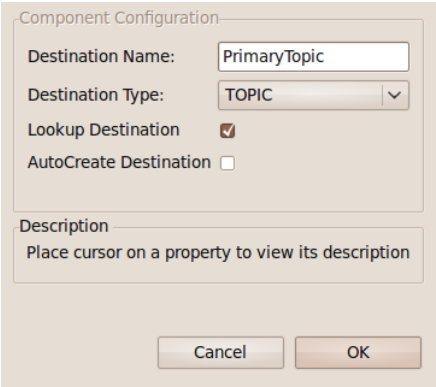


Figure: Destination Configuration

Delivery Mode, Time to Live, Priority which were present previously in the Interaction Configurations panel are grouped under the Producer Configuration.

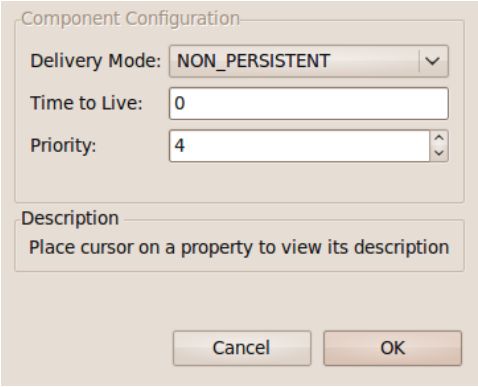


Figure: Producer Configuration

Similarly SSL Security details and Proxy Settings Panels are modified to use Named Configurations.

4.7.1.1.2 Working with Fiorano MQ HA profiles

When Configuring JMSIn with Fiorano MQ HA profiles we should provide Intial Context Properties in Adavnced Info in the MCF Panel of the cps.

These Properties are "java.naming.provider.url" set to Server Url and "BackupConnectURLs" set to backupUrl's.

4.7.1.1.3 Secured Connections to Fiorano MQ (SSL)

If Fiorano MQ server uses SSL then, provide the following properties in Advanced Info Under Initial Context Properties.

- **java.naming.security.protocol**

  Name of the security protocol used to create secure connections with the MQ server.

  Valid value: The possible values that this variable can take are PHAOS_SSL and SUN_SSL.

- **SecurityManager(optional)**

  Note: If this property is not provided then, default security provider will be used.

  The Security Manager implementation used to create secure connections [HTTPS or SSL] with the MQ server. The manager class should be an implementation of the fiorano.jms.runtime.IFMQSecurityManager interface provided by FioranoMQ.

  Valid value: Qualified class name of security Manager class.

Server connection can be tested from within the CPS by clicking on **Test** in the connection properties panel.

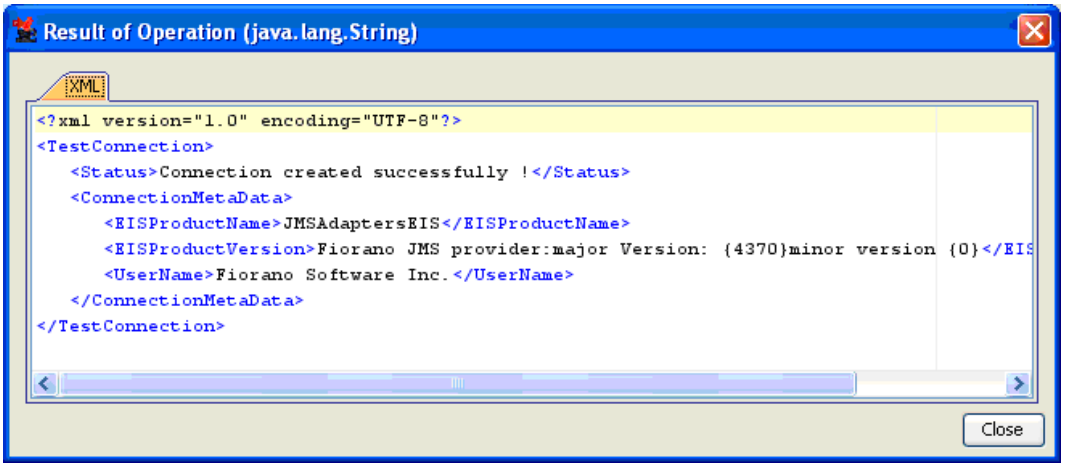
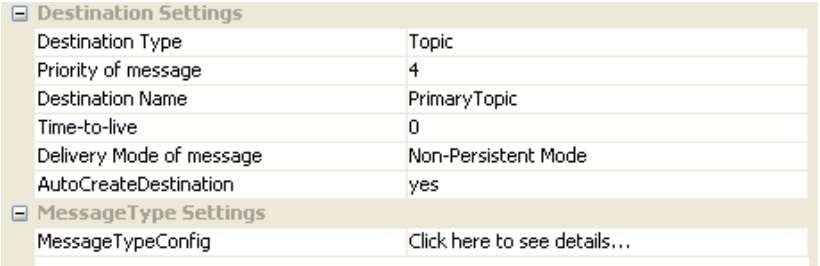


Figure 4.316: Sample connection test result indicating success

The JMSIn component can be configured using its Custom Proper Sheet wizard.



Destination Settings	
Destination Type	Topic
Priority of message	4
Destination Name	PrimaryTopic
Time-to-live	0
Delivery Mode of message	Non-Persistent Mode
AutoCreateDestination	yes
MessageType Settings	
MessageTypeConfig	Click here to see details...

Figure 4.317: Sample JMSIn configuration

Destination specified for the property **Destination Name** should already exist if BEA Weblogic/Oracle AQ/Oracle Streams AQ is being used. Dynamic creation of destinations is not supported for these providers.

For BEA Weblogic, value for **Destination Name** should be the JNDI name of the destination. **Example:** weblogic.examples.jms.exampleTopic.

Above configuration shown in the Figure 4.317 can be tested from within the CPS by clicking on test button in the CPS panel.

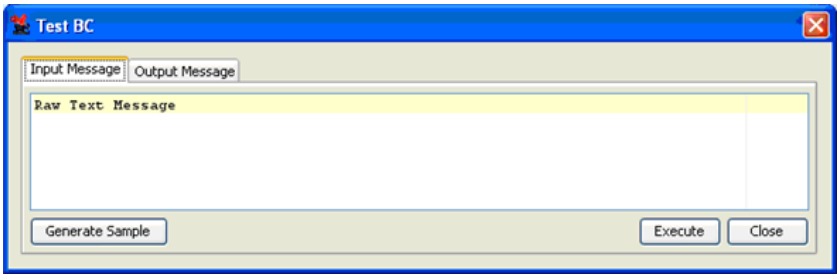


Figure 4.318: Sample JMSIn input message

Figure 4.319: Sample JMSIn output message

4.7.1.2 Schema

4.7.1.2.1 Input Schema

When the property Use Connection details from input is chosen, the following input schema with the element ConnectionFactorySettings, is generated. Properties that are used to create the connection are present under this element.

XML Item	Value Type
<ns1:Message>	
<ns1:ConnectionFactorySettings>?	
<ns1:ProviderUrlSettings>?	
<ns1:JMSProvider>?	Fiorano MQ | BEA WebLogic | Or...
<ns1:ServerUrl>?	#string
<ns1:BackupUrlsRequired>?	true | false
<ns1:BackupUrls>?	#string
<ns1:JNDISettings>?	
[Choice]	
<ns1:BEAWEBLogic>	
<ns1:IntialContextFactory>?	#string
<ns1:JNDIUserName>?	#string
<ns1:JNDIPassword>?	#string
<ns1:FioranoMQ>	
<ns1:IntialContextFactory>?	#string
<ns1:JNDIUserName>?	#string
<ns1:JNDIPassword>?	#string
<ns1:OracleAQ>	
<ns1:IntialContextFactory>?	#string
<ns1:DestinationType>?	Queue | Topic
<ns1:OracleStreamsAQ>	
<ns1:IntialContextFactory>?	#string
<ns1:Other>	
<ns1:IntialContextFactory>?	#string
<ns1:JNDIUserName>?	#string
<ns1:JNDIPassword>?	#string
<ns1:ConnectionProperties>?	
<ns1:ConnectionFactoryName>?	#string
<ns1:JMSUserName>?	#string
<ns1:JMSPassword>?	#string
<ns1:ClientID>?	#string
<ns1:IntialContextProperties>?	
<ns1:AdvancedInfo>?	#string
<ns1:SessionProperties>?	
<ns1:AcknowledgeMode>?	Auto Acknowledge | Client Ackn...
<ns1:MessageAckCount>?	#int
<ns1:SSLSettings>?	

Figure 4.320: Input schema with ConnectionFactorySettings

4.7.1.2.2 Output Schema

Schema Element	Description
<Body>	Body of the response message
<SentMessage>	Message sent to the destination

4.7.1.3 Functional Demonstration

4.7.1.3.1 Scenario 1

Putting a simple Text message on a destination and displaying the response message.

Configure the JMSIn as described in *Configuration and Testing* section and use feeder and display component to send sample input and check the response respectively.

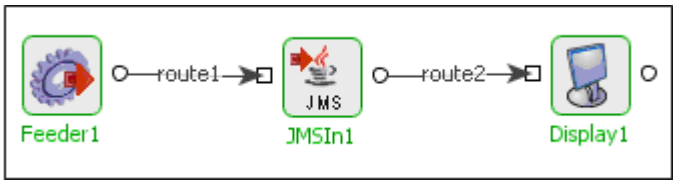


Figure 4.321: Demonstrating Scenario 1 with sample input and output

Sample Input:

```
Input Text
```

Sample Output:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:Message xmlns:ns1="http://www.fiorano.com/fesb/activity/JMSIn1/jmsIn/Out">
 <ns1:Body>Message sent to Destination PrimaryTopic</ns1:Body>
 <ns1:SentMessage>Input Text</ns1:SentMessage>
</ns1:Message>
```

4.7.1.4 Useful Tips

- Set optimal message age (Time-to-live property) so as to reduce memory overhead, thus improving performance.
- Less message size gives better performance and vice versa. For example, ByteMessage takes less memory than TextMessage, hence choose message type carefully to avoid unnecessary memory overhead.
- Delivery mode defines whether the message can be persistent or non-persistent; this factor has an impact on the performance. Choose non-persistent messages where appropriate.

4.7.2 JMSIn 5.0

The JMSIn component is used to send messages to a JMS destination (topic or queue). This component sends the messages received on the input port to the configured JMS destination. The destination on which the message is to be sent can be specified in the Configuration Property Sheet wizard.

This component can be used to send Text, Bytes or Map messages. The only restriction on Map Messages is that this component does not support Objects in Map Messages.

Points to note:

- The JMS providers supported are Fiorano MQ and BEA Weblogic. For working with these JMS providers the jars that should be explicitly added as resources to the component are given below

  BEA WebLogic:

  - %BEA_HOME%\server\lib\wlclient.jar
  - %BEA_HOME%\server\lib\wljmsclient.jar (required only when the Weblogic server is running on a remote machine)

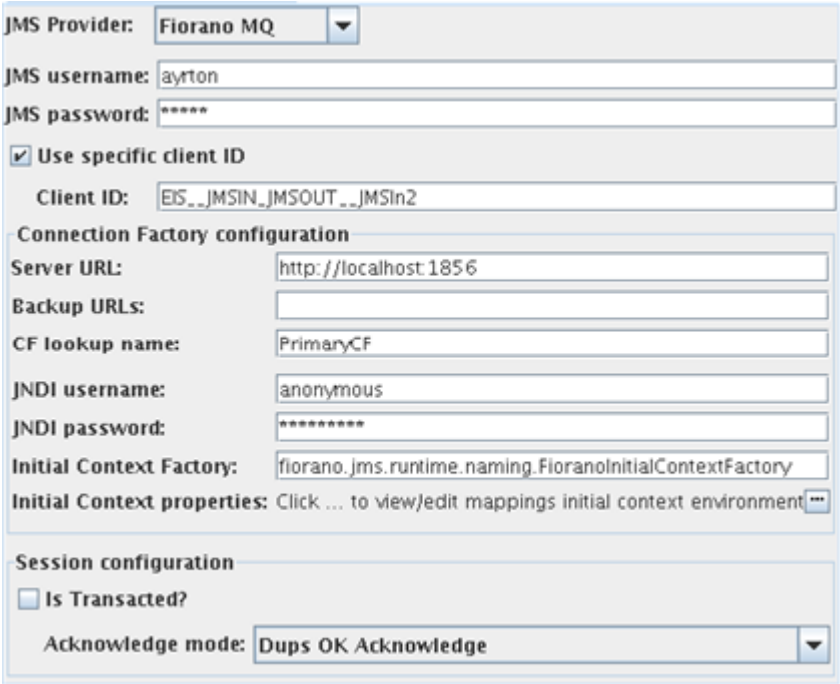
  Note: For BEA Weblogic 10.0, %BEA_HOME% refers to <BEA WebLogic Installation directory>\wlserver_10.0. If these jars are added to resources of the System library JMSAdapters, the jars will be available for JMSIn, JMSOut & JMSRequestor components.

- In case of BEA Weblogic, InitialContext can be created by specifying empty values for JNDI Username and JNDI password as well.
- Set optimal message age (Time-to-live property) so as to reduce memory overhead, thus improving performance.
- Less message size gives better performance and vice versa. For example, ByteMessage takes less memory than TextMessage, hence choose message type carefully to avoid unnecessary memory overhead.
- Delivery mode defines whether the message can be persistent or non-persistent. This factor has an impact on the performance. Choose non-persistent messages where appropriate.

4.7.2.1 Connection

4.7.2.1.1 Connection Configuration

Connection configuration details can be specified in this panel.



JMS Implementation:

Choose the MQ server from the dropdown list provided.

JMS Username: The user name with which the client connects to the MQ server.

> **Note:** When the username is not provided, the password is ignored too and JMS connection creation is attempted without username and password.

JMS Password: The password for the username that is provided.

Use specific client ID: Select this option to specify a user defined Client ID. If it is not selected client ID is generated automatically at runtime.

Client ID: The client ID that will be set on the connection with the server.

> **Note**: This will be used only when the option "Use specific client ID" is selected.

Server URL: The URL of the server to which the component connects.

Backup URLs: The backup URLs to which the component tries to connect if the server specified by the property "server URL" is down. Multiple backup URLs can be specified by separating them with ";".

CF lookup name: The lookup name of the Connection Factory.

JNDI Username: The name with which the user connects to the JNDI server to perform lookup operations.

JNDI Password: The password for the JNDI username.

Initial context factory: The name of the initial context factory.

Initial context properties: Additional properties which can be used for creating an Initial Context can be specified here. For instance, if JMSAdapter has to connect to the FMQ server using HTTP protocol instead of the default TCP protocol, then a property 'TransportProtocol' can be specified with the value 'HTTP'.

- **Is transacted**: Select this option if session to be created should be transacted. If this is not selected appropriate Acknowledge mode has to be chosen.
- **Acknowledgement mode**: The type of Acknowledgement mode can be specified here. A session retains messages it consumes until they have been acknowledged.
    - **Dups OK Acknowledge**: This acknowledgment mode instructs the session to lazily acknowledge the delivery of messages.
    - **Auto Acknowledge**: The session automatically acknowledges a client's receipt of a message either when the session has successfully returned from a call to **receive** or when the message listener the session has called to process the message successfully returns.
    - **Client Acknowledge**: With this acknowledgment mode, the client acknowledges a consumed message.

  Note: This option is relevant only when **isTransacted** is not selected.

Secured Connections to Fiorano MQ (SSL)

If Fiorano MQ server uses SSL then, provide the following properties in Initial Context Properties.

- **java.naming.security.protocol**

  Name of the security protocol used to create secure connections with the MQ server.

  Valid value: The possible values that this variable can take are PHAOS_SSL and SUN_SSL.

- **SecurityManager(optional)**

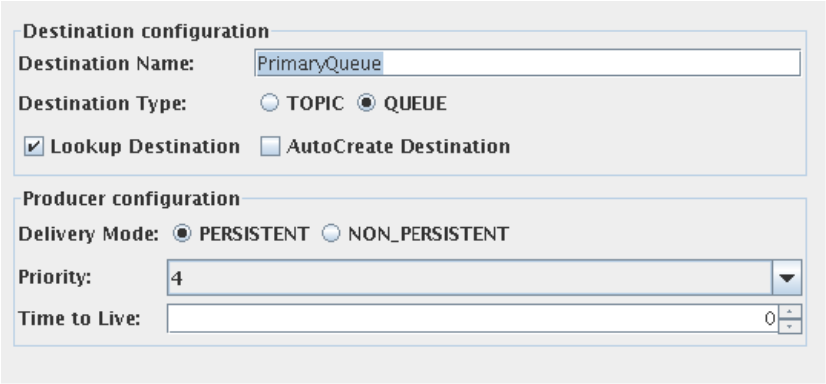
  Note: If this property is not provided then, default security provider will be used.

  The Security Manager implementation used to create secure connections [HTTPS or SSL] with the MQ server. The manager class should be an implementation of the fiorano.jms.runtime.IFMQSecurityManager interface provided by FioranoMQ.

  Valid value: Qualified class name of security Manager class.

4.7.2.1.2 Send Configuration

Send configuration details can be specified in this panel.



Destination Name:

The name of the destination to which the messages have to be sent.

Destination Type:

The JMS type of the destination. This can be chosen as either Topic or Queue.

Lookup Destination:

This option can be chosen if the destination is already present and needs to be looked up using JNDI.

Note: In case of **Active MQ**, to use look up destination, additional JNDI property [topic|queue].<look up name> has to be added as described in the following link. http://activemq.apache.org/jndi-support.html.

AutoCreate Destination:

This option has to be chosen if the destination has to be created, if it doesn't exist.

Note:

- One of **Lookup destination** or **Autocreate destination** is mandatory.
- If both are chosen, the component first tries to lookup the destination and if it fails it will create the destination.

4.7.2.1.3 Producer Configuration

Delivery Mode:

- **PERSISTENT**: The PERSISTENT delivery mode, instructs the JMS provider to take extra care to ensure that a message is not lost in transit in case of a JMS provider failure. A message sent with this delivery mode is logged to stable storage when it is sent.
- **NON-PERSISTENT**: The NON_PERSISTENT delivery mode does not require the JMS provider to store the message or otherwise guarantee that it is not lost if the provider fails.

Priority:

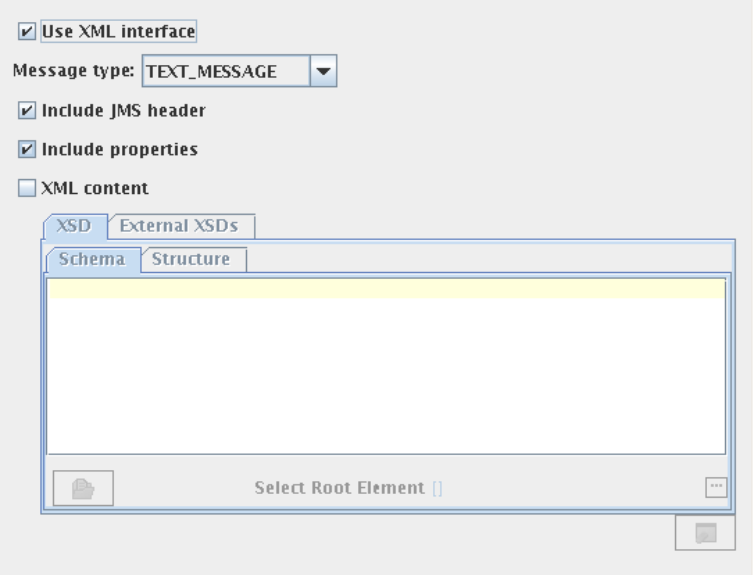
The priority of the message to be sent to the destination.

Time to Live:

The time to live (in milliseconds) of the message to be sent to the destination. After the timeout the message will be discarded.

4.7.2.1.4 Message Definition

Message type configuration can be defined in this panel. The schema of the input port of the component will be determined based on the configuration provided.



Use XML Interface:

This option has to be chosen if the component needs an XML interface. If this is not selected, schema will not be set on input port of the component and the message received on the input port will be sent to the destination as it is.

Message Type:

The type of the message that needs to be sent can be chosen as one of the following.

- **Text message**: Use this option to send a plain text message.
- **Map message**: A Map Message object's message body contains a set of name-value pairs, where names are String objects, and values are Java primitives. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined.
- **Stream message**: A Stream Message object's message body contains a stream of un-interpreted bytes. This message type is for literally encoding a body to match an existing message format.

Include JMS Headers:

This option can be selected to set the JMSHeaders on the message at runtime.

When this option is selected, the fields JMSCorrelationID and JMSDestination will be generated in the input schema of the component.

Include Properties:

This option can be selected to set the JMS properties on the message at runtime.

When this option is selected, the elements **ApplicationContext** and **Property** (ZeroMany) are generated in the input schema of the component.

The properties and application context can be set as shown in the figure.

```
<ns1:Properties>
   <ns1:ApplicationContext><![CDATA[<context>test</context>]]></ns1:ApplicationContext>
   <ns1:Property name="name" type="String">string</ns1:Property>
</ns1:Properties>
```

Depending on the Message Type selected, the appropriate parsing option is shown. Message Type and the corresponding parsing option is explained below.

Text message – XML Content:

Select this option if the content of the input message conforms to a specific schema. The schema can be specified using the schema editor.

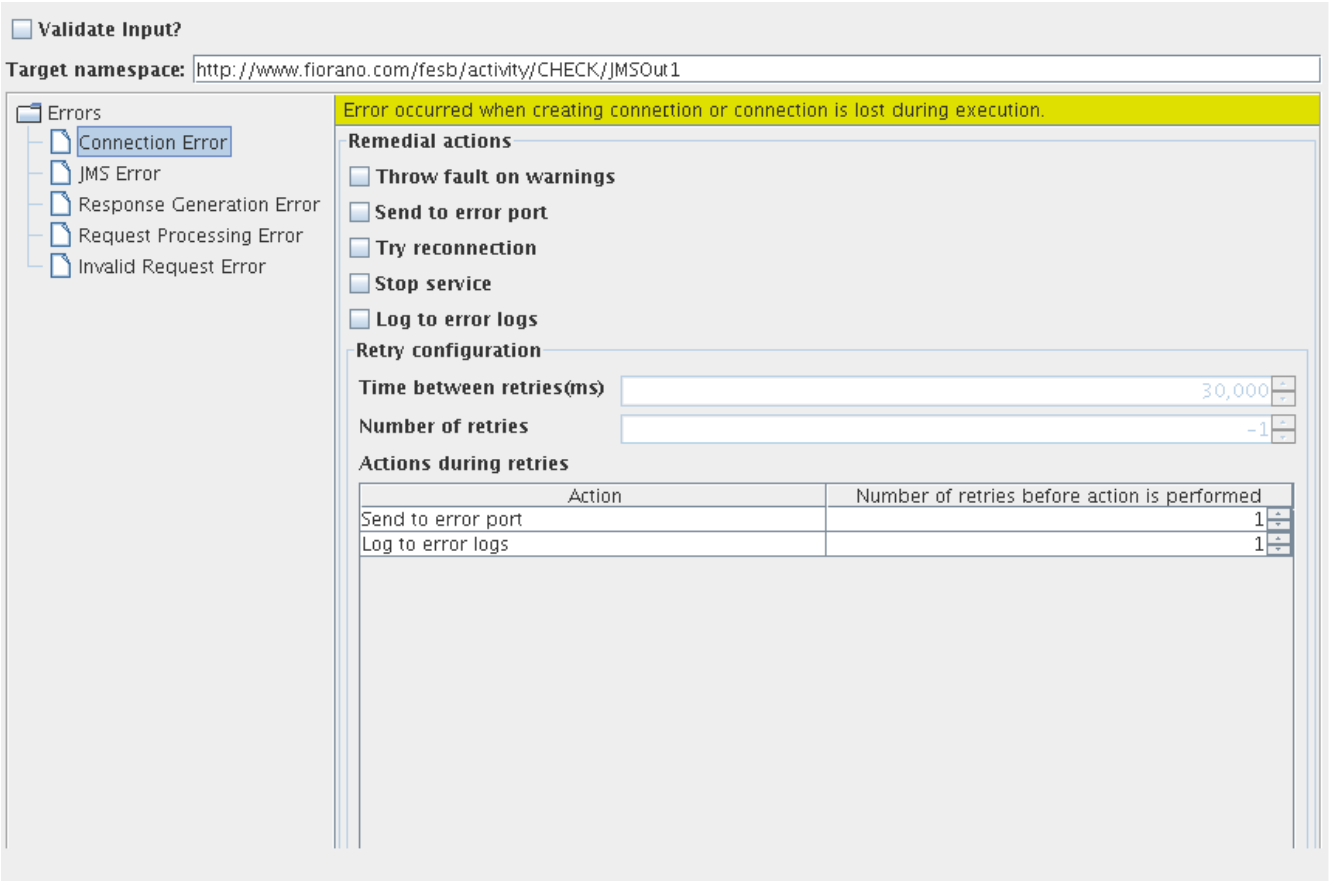
Map message – Define Body:

Select this option to define the body fields of the Map message. The message body fields (name and type of the field) can be added in the table and the input schema will be generated corresponding to the each field defined.

Stream message – Parse Body:

Select this option to set the content of the stream message in input XML. The name, type and length of the field can be added in the table. An element will be added to the schema corresponding to each field with the same name.

4.7.2.1.5 Additional Configuration



Validate Input: Select this option if the input message has to be validated against the schema that is set on the input port.

If a complex schema is used in the component, then enabling this option has an impact on the performance.

Target namespace: The target namespace of the schema can be provided here. The schema on ports will be suffixed with the component name and type of the port. For example, JMSIn/Out.

Error Handling Configuration

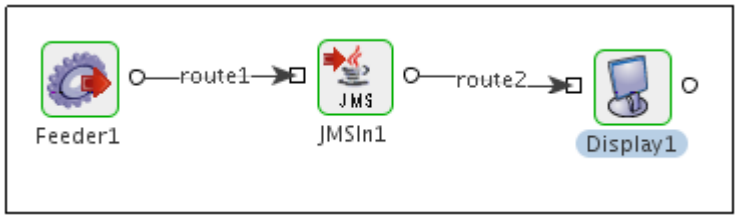
The actions that have to be taken in case of different kinds of errors can be chosen using this panel.

4.7.2.2 Functional Demonstration

Scenario 1: To send a Map message to the destination **PrimaryQueue**.

The component is configured to send a Map message as described in section Message Definition and the option **include JMSHeader** is enabled to set JMS headers on the message.

Feeder and display components are connected to the component to send sample input and check the response respectively.



4.7.2.2.1 Sample Input

```
<ns1:Message xmlns:ns1="http://www.fiorano.com/fesb/activity/JMSIN/JMSIn1/jmsIn/In">
   <ns1:JMSHeader>
      <ns1:JMSCorrelationID>JMSCorrelationID</ns1:JMSCorrelationID>
      <ns1:JMSDestination type="QUEUE">PrimaryQueue</ns1:JMSDestination>
   </ns1:JMSHeader>
   <ns1:Properties>
      <ns1:ApplicationContext><![CDATA[<context>test</context>]]></ns1:ApplicationContext>
      <ns1:Property name="name" type="String">string</ns1:Property>
   </ns1:Properties>
   <ns1:Body>
      <ns1:key type="type">string</ns1:key>
   </ns1:Body>
</ns1:Message>
```

4.7.2.2.2 Sample Output

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:Message xmlns:ns1="http://www.fiorano.com/fesb/activity/JMSIN/JMSIn1/jmsIn/Out">
   <ns1:Body>Message sent successfully to PRIMARYQUEUE</ns1:Body>
   <ns1:SentMessage>&lt;ns1:Message xmlns:ns1="http://www.fiorano.com/fesb/activity/JMSIN/JMSIn1/jmsIn/In"&gt;
   &lt;ns1:JMSHeader&gt;
      &lt;ns1:JMSCorrelationID&gt;JMSCorrelationID&lt;/ns1:JMSCorrelationID&gt;
      &lt;ns1:JMSDestination type="QUEUE"&gt;PrimaryQueue&lt;/ns1:JMSDestination&gt;
   &lt;/ns1:JMSHeader&gt;
   &lt;ns1:Properties&gt;
      &lt;ns1:ApplicationContext&gt;&lt;![CDATA[&lt;context&gt;test&lt;/context&gt;]]&gt;&lt;/ns1:ApplicationContext&gt;
      &lt;ns1:Property name="name" type="String"&gt;string&lt;/ns1:Property&gt;
   &lt;/ns1:Properties&gt;
   &lt;ns1:Body&gt;
      &lt;ns1:key type="type"&gt;string&lt;/ns1:key&gt;
   &lt;/ns1:Body&gt;
&lt;/ns1:Message&gt;</ns1:SentMessage>
</ns1:Message>
```

4.7.2.3 Useful Tips

- Set optimal message age (Time-to-live property) so as to reduce memory overhead, thus improving performance.
- Less message size gives better performance and vice versa. For example, ByteMessage takes less memory than TextMessage, hence choose message type carefully to avoid unnecessary memory overhead.

- Delivery mode defines whether the message can be persistent or non-persistent, this factor has an impact on the performance. Choose non-persistent messages where appropriate.
- Enabling validate input option when a complex schema is used has an impact on the performance of the component.

4.7.3 JMSOut 4.0

The JMSOut component may be used to retrieve messages from a JMS Topic/Queue. Using the Configuration Property Sheet, you can specify the topic or queue from which the message is to be retrieved. The JMSOut component sends the JMS message received from a Topic/Queue to another component. You can create a Subscriber or a Receiver for a Topic or a Queue respectively, and configure the component to retrieve or subscribe to messages from a JMS server at runtime.

This component can be used to receive Text, Bytes or Map messages. The only restriction on Map Messages is that this component does not support Objects in Map Messages.JMSOut is capable of handling following types of messages:

- **Text messages**: A TextMessage object's message body contains a java.lang.String object.
- **Map messages**: A MapMessage object's message body contains a set of name-value pairs, where names are String objects, and values are Java primitives. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined.
- **Bytes Message**: A BytesMessage object's message body contains a stream of uninterrupted bytes. This message type is for literally encoding a body to match an existing message format.

JMSOut uses JMS APIs to process the messages

Points to note

- The only restriction on Map Messages is that this component does not support Objects in Map Messages.
- When adding the Initial Context Factory class for non Fiorano MQ server, the jar file(s) should be added as resource(s) to the JMSAdapters system library.
- For creating Durable Subscriber, use Topic Connection Factory in place of Unified Connection Factory.
- If IsDurableSubscriber property is changed to No or Destination Type is changed from Topic to Queue or ClientID is changed - the durable subscriptions created earlier should be unsubscribed to avoid persisting messages for the durable subscriber in runtimedata. For more information on unsubscribing, please refer to FioranoMQ Handbook - Section 29.3: **Connecting to Web Management Tool - DurableSubscribers view**.
- If a timeout is not specified (left as zero – infinite), and after starting the component, the JMS server (to which the component is connected) crashes, the receive call on the Queue or Topic waits endlessly. Therefore, it is advisable to give a definite timeout value.

- If the component is being used in scheduling mode, the execution timeout for the component should be less than the scheduler interval if the message is being received from a Queue.
- The property 'Cleanup resources (excluding connection) after each document' is set to 'No' by default. If this is set to 'Yes', then only one message is received by JMSOut and after that resources like message consumers are closed and the component will not receive further messages.

4.7.3.1 Configuration and Testing

The JMSOut component connection related properties can be configured in the Managed Connection Factory panel of CPS.

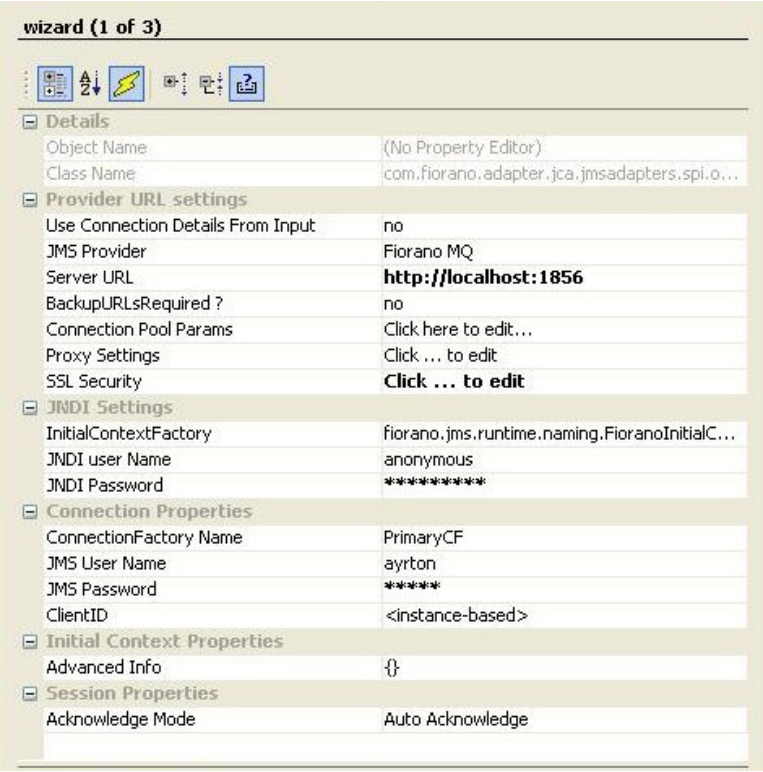


Figure 4.321: Sample JMSOut MCF configuration

Use Connection Details From Input: Parameters to create the connection can be specified in the input message when this property is set to true. If this property is selected the validation errors in the managed connection factory panel of the CPS are treated as warnings. So user can bypass this step without giving valid configuration and complete the configuration of the component. If valid properties are not provided even in the input message exception will be thrown at runtime.

The JMS providers supported are Fiorano MQ, BEA Weblogic, Oracle AQ and Oracle Streams AQ. For working with these JMS providers the jars that should be explicitly added as resources to the component are given below:

BEA WebLogic:

- %BEA_HOME%\server\lib\wlclient.jar
- %BEA_HOME%\server\lib\wljmsclient.jar (required only when the Weblogic server is running on a remote machine)

  Note: For BEA Weblogic 10.0, %BEA_HOME% refers to <BEA WebLogic Installation directory>\wlserver_10.0

Oracle AQ:

- %ORACLE_HOME%\rdbms\jlib\aqapi13.jar (If JDK1.2 / JDK1.1 is used, aqapi12.jar/aqapi11.jar has to be used respectively)
- %ORACLE_HOME%\jdbc\lib\classes12.zip
- %ORACLE_HOME%\jdbc\lib\nls_charset12.jar

  Note: For Oracle Database 9.2.0.1.0, %ORACLE_HOME% refers to <Oracle Installation directory>\ora92

Oracle Streams AQ:

- %ORACLE_HOME%\rdbms\jlib\aqapi.jar
- %ORACLE_HOME%\jdbc\lib\ojdbc14.jar

  Note: For Oracle Database 10.2.0.1.0, %ORACLE_HOME% refers to <Oracle Installation directory>\product\10.2.0\db_1

Note the following:

- If these jars are added to resources of the System library JMSAdapters, the jars are available for JMSIn, JMSOut and JMSRequestor components.
- In case of BEA Weblogic, InitialContext can be created by specifying empty values for JNDI Username and JNDI password as well.

Working with Fiorano MQ HA profiles

When Configuring JMSOut with Fiorano MQ HA profiles we should provide Intial Context Properties in Adavnced Info in the MCF Panel of the cps.

These Properties are "java.naming.provider.url" set to Server Url and "BackupConnectURLs" set to backupUrl's.

Secured Connections to Fiorano MQ (SSL)

If Fiorano MQ server uses SSL then, provide the following properties in Advanced Info Under Initial Context Properties.

- **java.naming.security.protocol**

  Name of the security protocol used to create secure connections with the MQ server.

  Valid value: The possible values that this variable can take are PHAOS_SSL and SUN_SSL.

- **SecurityManager(optional)**

  Note: If this property is not provided then, default security provider will be used.

The Security Manager implementation used to create secure connections [HTTPS or SSL] with the MQ server. The manager class should be an implementation of the fiorano.jms.runtime.IFMQSecurityManager interface provided by FioranoMQ.

Valid value: Qualified class name of security Manager class.

Server connection can be tested from within the CPS by clicking on test in the connection properties panel.

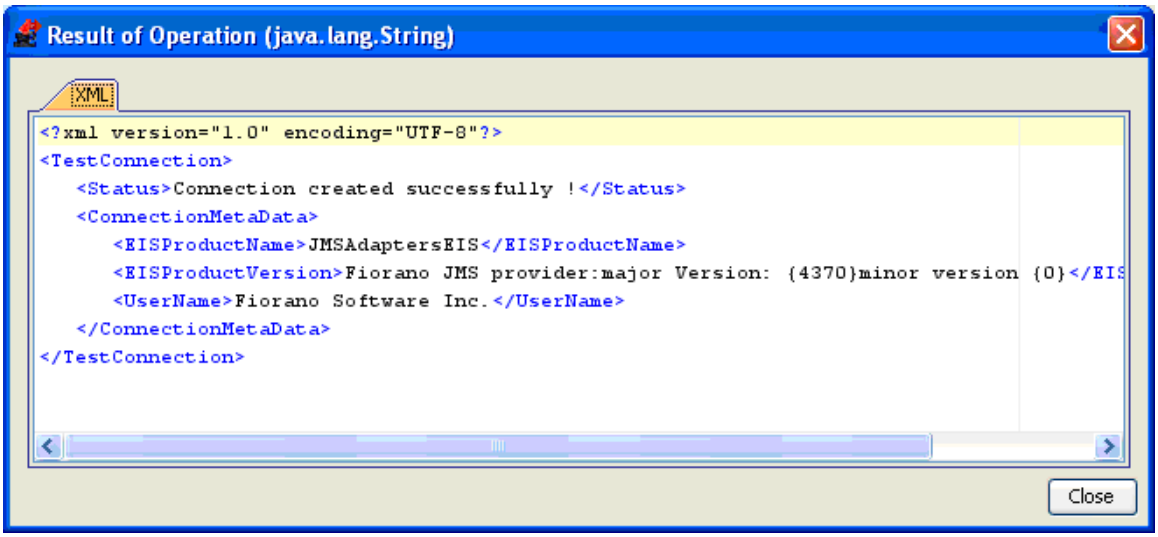


Figure 4.322: Sample connection test result indicating success

4.7.3.1.1 Interaction Configurations

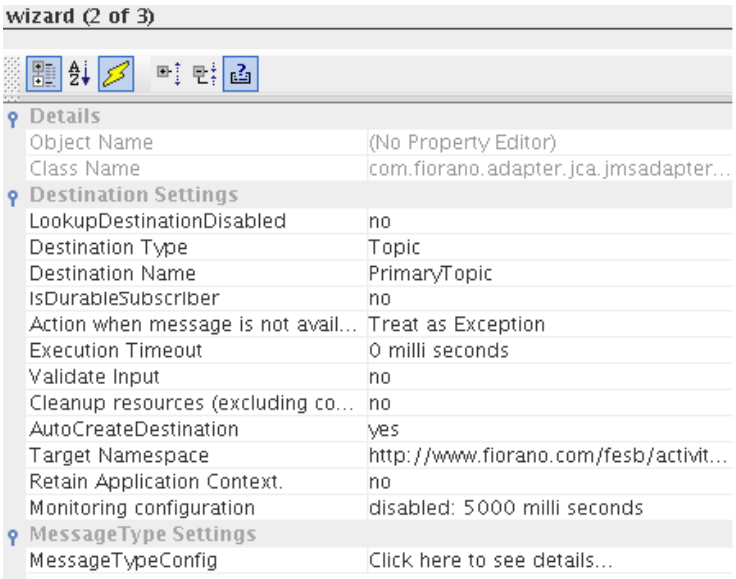


Figure: Interaction Configurations

Destination Settings

LookupDestination Disabled

- **yes** – This property specifies if the destination lookup is disabled
- **no** - specifies if the destinations could be looked up.

Destination Type

- **Topic** - This value is selected when the type of destination is 'Topic'.
- **Queue** – This value is selected when the type of destination is 'Queue'.

Destination Name

Name of the topic or queue to which messages are to be transferred.

Note: For BEA Weblogic, value for Destination Name should be the JNDI name of the destination. Ex: weblogic.examples.jms.exampleTopic.

IsDurableSubscriber

This property is used to create a durable subscription when the destination type is Topic.

Action when message is not available?

Figure: Option to configure when message is not available

This property specifies the action to be taken when there are no messages on the queue within the timeout specified.

- Send Empty Message
- An empty message is sent to output port.
- Treat as Exception
- An exception message is sent to error port and no message is sent to output port.
- No Action
- No message is sent to either output port or error port.

Execution Timeout

Maximum time in milliseconds the component should wait for a message. Default value is 0 which specifies infinite wait time that is, waits until a message is received.

AutoCreate Destination

If destination provided is not present, then it is created dynamically.

Note: Destination specified for the property Destination Name should already exist if BEA Weblogic/Oracle AQ/Oracle Streams AQ is being used. Dynamic creation of destinations is not supported for these providers.

The JMSOut component can be configured using its Custom Proper Sheet wizard.

Destination Settings	
Destination Type	**Queue**
Destination Name	**PrimaryQueue**
AutoCreateDestination	yes
MessageType Settings	
MessageTypeConfig	Click here to see details...

Figure 4.323: Sample JMSOut configuration

Destination specified for the property **Destination Name** should already exist if BEA Weblogic/Oracle AQ/Oracle Streams AQ is being used. Dynamic creation of destinations is not supported for these providers.

For BEA Weblogic, value for **Destination Name** should be the JNDI name of the destination. **Example:** weblogic.examples.jms.exampleTopic.

Above configuration shown in the Figure 4.323 can be tested from within the CPS by clicking on test button in the CPS panel.

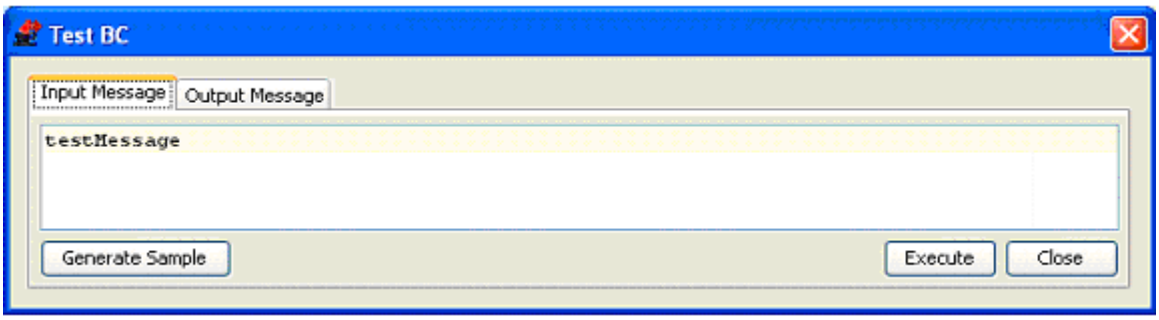


Figure 4.324: Sample JMSOut input message

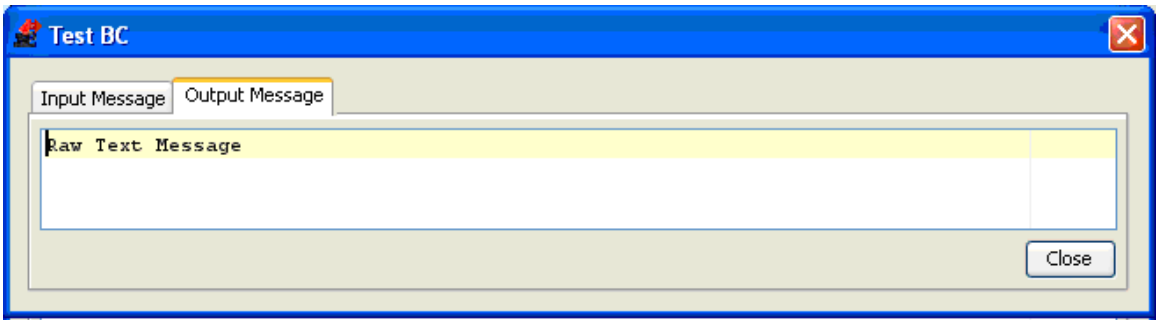


Figure 4.325: Sample JMSOut output message

4.7.3.1.2 Using Named Configurations

Changes are done in eStudio w.r.t to Named Configurations. With the implementation of Named Configurations MCF panel is changed as shown in Figure below.

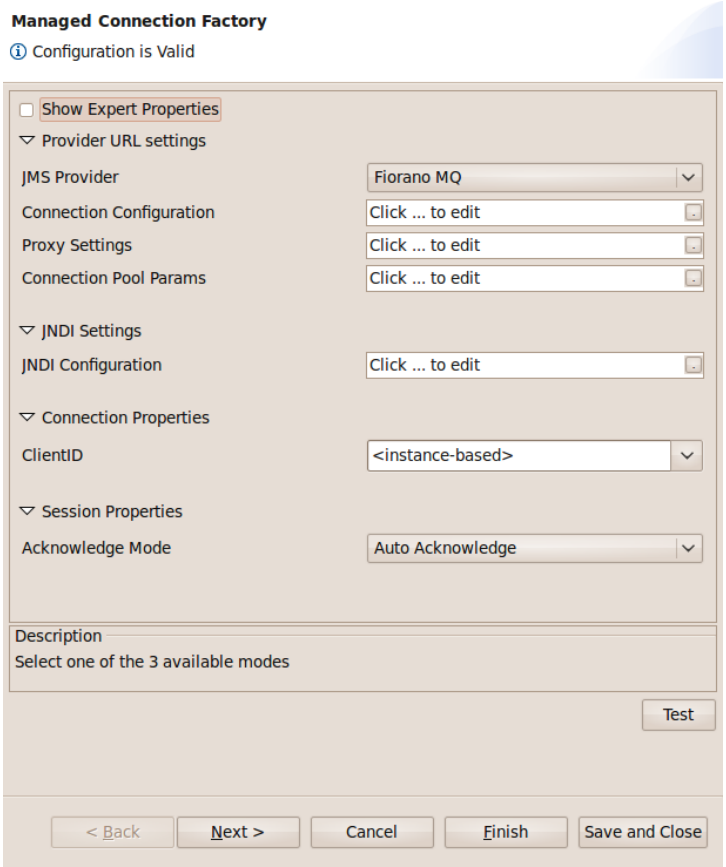


Figure: Modified Managed Connection Factory

Some properties are grouped as one property in the modified configuration panel. ServerURL, BackupURL, CF lookup name, JMS username, JMS password which were present previously in the MCF panel are grouped under the name ConnectionConfiguration.

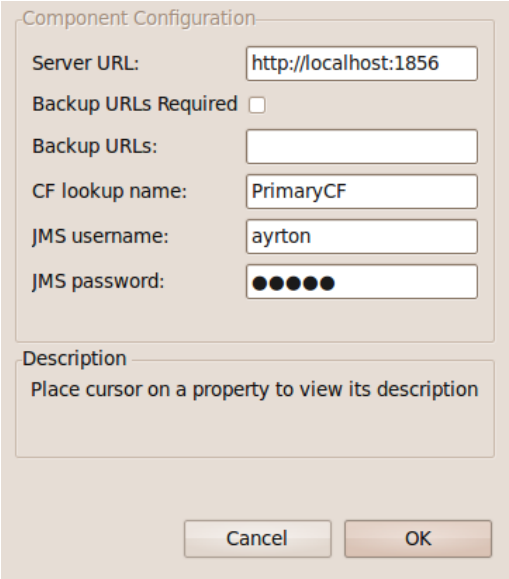


Figure: Connection Configuration

Initial Context Factory, JNDI username, CF lookup name, JNDI password which were present previously in the MCF panel are grouped under the name JNDI Settings.

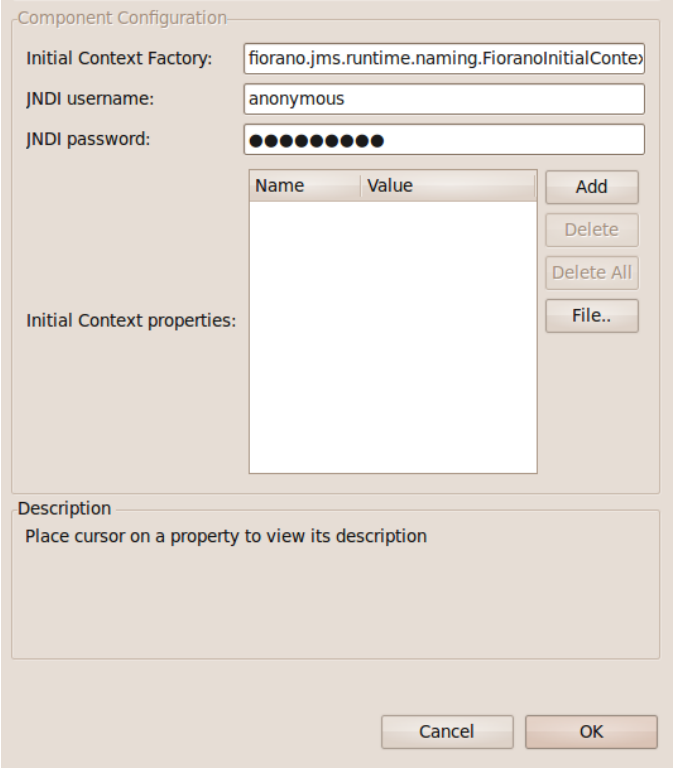


Figure: JNDI Settings

Destination Name, Destination Type, Lookup Destination which were present previously in the Interaction Configurations panel are grouped under the name Destination Configuration.

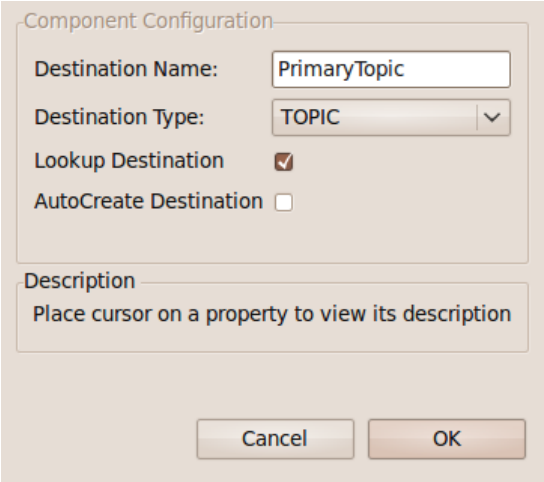


Figure: Destination Configuration

Is Durable and Subscription Name which were present previously in the Interaction Configurations panel are grouped under the name Consumer Configuration.

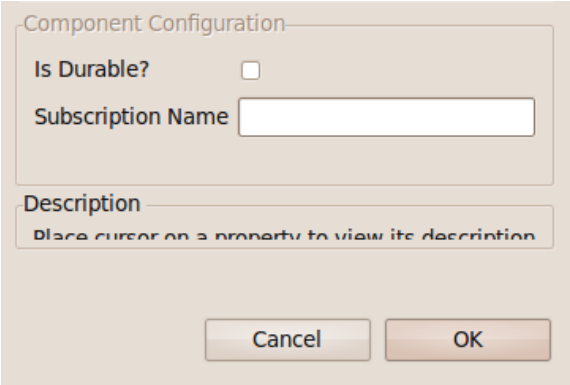


Figure: Consumer Configuration

Similarly SSL Security details and Proxy Settings Panels are modified to use Named Configurations.

4.7.3.2 Input Schema

When the property Use Connection details from input is chosen, the following input schema with the element ConnectionFactorySettings, is generated. Properties that are used to create the connection are present under this element.

XML Item	Value Type
<ns1:Message>	
<ns1:ConnectionFactorySettings>?	
<ns1:ProviderUrlSettings>?	
<ns1:JMSProvider>?	Fiorano MQ | BEA WebLogic | Or...
<ns1:ServerUrl>?	#string
<ns1:BackupUrlsRequired>?	true | false
<ns1:BackupUrls>?	#string
<ns1:JNDISettings>?	
[Choice]	
<ns1:BEAWEBLogic>	
<ns1:IntialContextFactory>?	#string
<ns1:JNDIUserName>?	#string
<ns1:JNDIPassword>?	#string
<ns1:FioranoMQ>	
<ns1:IntialContextFactory>?	#string
<ns1:JNDIUserName>?	#string
<ns1:JNDIPassword>?	#string
<ns1:OracleAQ>	
<ns1:IntialContextFactory>?	#string
<ns1:DestinationType>?	Queue | Topic
<ns1:OracleStreamsAQ>	
<ns1:IntialContextFactory>?	#string
<ns1:Other>	
<ns1:IntialContextFactory>?	#string
<ns1:JNDIUserName>?	#string
<ns1:JNDIPassword>?	#string
<ns1:ConnectionProperties>?	
<ns1:ConnectionFactoryName>?	#string
<ns1:JMSUserName>?	#string
<ns1:JMSPassword>?	#string
<ns1:ClientID>?	#string
<ns1:IntialContextProperties>?	
<ns1:AdvancedInfo>?	#string
<ns1:SessionProperties>?	
<ns1:AcknowledgeMode>?	Auto Acknowledge | Client Ackn...
<ns1:MessageAckCount>?	#int
<ns1:SSLSettings>?	

Figure: Input schema with ConnectionFactorySettings

4.7.3.3 Functional Demonstration

4.7.3.3.1 Scenario 1

Putting a simple Text message on a destination and displaying the response message.

Configure the JMSOut as described in *Configuration and Testing* section and use feeder and display component to send sample input and check the response respectively.

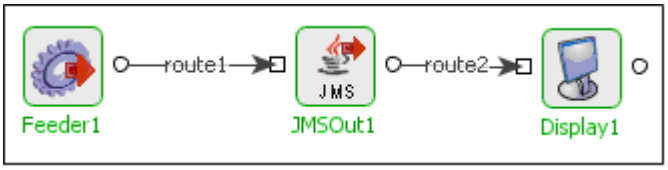


Figure 4.326: Demonstrating Scenario 1 with sample input and output

Sample Input:

```
Input Text
```

Sample Output:

```
Raw Text Message
```

4.7.3.4 Useful Tips

- Choose non-durable messages where appropriate
- When using the durable delivery mode, each message has to be stored by the JMS server either in the database or the file system depending on the vendor before delivery of message to consumer and removed after delivery of message. This has a huge impact on the performance. So as far as possible restrict the use of durable delivery mode unless and until absolutely necessary for your application to avoid the overheads involved.
- Choose proper acknowledgement mode

4.7.4 JMSOut 5.0

The JMSOut component is used to retrieve messages from a JMS destination (topic or queue). The JMSOut component can work either in synchronous or asynchronous mode to fetch the messages from the configured destination.

This component can be used to receive Text, Bytes or Map messages. The only restriction on Map Messages is that this component does not support Objects in Map Messages.

Points to Note:

- The JMS providers supported are Fiorano MQ and BEA Weblogic. For working with these JMS providers the jars that should be explicitly added as resources to the component are given below:

  BEA WebLogic:

  - %BEA_HOME%\server\lib\wlclient.jar
  - %BEA_HOME%\server\lib\wljmsclient.jar (required only when the Weblogic server is running on a remote machine)

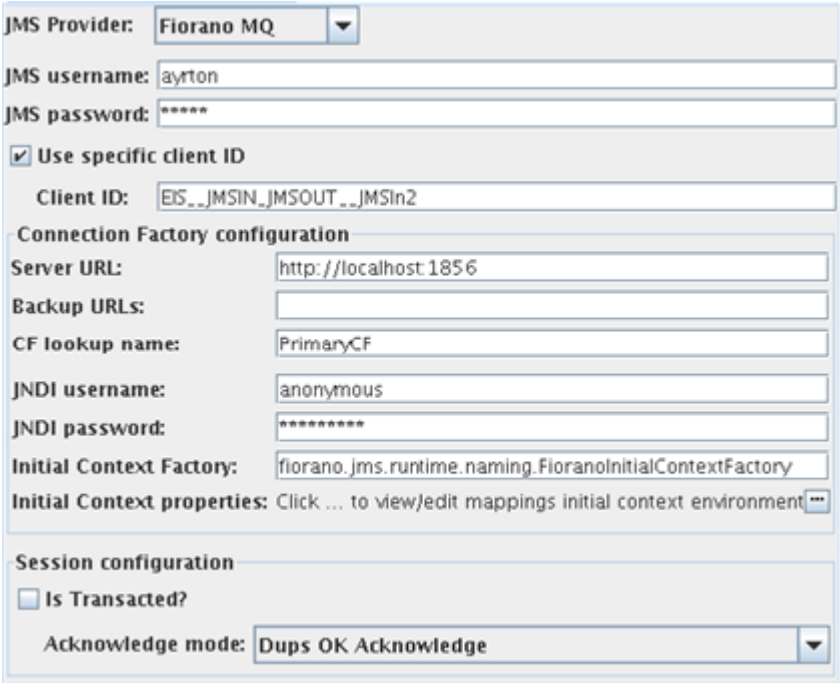
Note: For BEA Weblogic 10.0, %BEA_HOME% refers to <BEA WebLogic Installation directory>\wlserver_10.0. If these jars are added to resources of the System library JMSAdapters, the jars will be available for JMSIn, JMSOut & JMSRequestor components.

- In case of BEA Weblogic, InitialContext can be created by specifying empty values for JNDI Username and JNDI password as well.
- When using the Durable Subscriber mode, each message has to be stored by the JMS server either in the database or the file system depending on the vendor before delivery of message to consumer and removed after delivery of message. This has a huge impact on the performance. So as far as possible restrict the use of durable delivery mode unless it is necessary for your application to avoid the overheads involved.

4.7.4.1 Configuration

4.7.4.1.1 Connection Configuration

Connection configuration details can be specified in this panel.



JMS Implementation

Choose the MQ server from the dropdown list provided.

JMS Username: The user name with which the client connects to the MQ server.

Note: When the username is not provided, the password is ignored too and JMS connection creation is attempted without username and password.

JMS Password: The password for the username that is provided.

Use specific client ID: Select this option to specify a user defined Client ID. If it is not selected client ID is generated automatically at runtime.

> **Note**: If user defined client ID is used in the component and if durable subscription is enabled, while reconfiguring the component if the durable subscription changes, then the previously created durable subscription will also receive the messages. To avoid this there is an option to remove the previously created durable subscription. Refer "Useful tips" for more details.

Client ID: The client ID that will be set on the connection with the MQ server.

> **Note**: This will be used only when the option "Use specific client ID" is selected.

Server URL: The URL of the server to which the component connects.

Backup URLs: The backup URLs to which the component tries to connect if the server specified by the property "server URL" is down. Multiple backup URLs can be specified by separating them with ";".

CF lookup name: The lookup name of the connection factory.

JNDI Username: The name with which the user connects to the JNDI server to perform lookup operations.

JNDI Password: The password for the JNDI username.

Initial context factory: The name of the initial context factory.

Initial context properties: Additional properties which can be used for creating an Initial Context can be specified here. For instance, if JMSAdapter has to connect to the FMQ server using HTTP protocol instead of the default TCP protocol, then a property 'TransportProtocol' can be specified with the value 'HTTP'.

- **Is transacted**: Select this option if session to be created should be transacted. If this is not selected appropriate Acknowledge mode has to be chosen.
- **Acknowledgement mode**: The type of Acknowledgement mode can be specified here. A session retains messages it consumes until they have been acknowledged.
    - **Dups OK Acknowledge**: This acknowledgment mode instructs the session to lazily acknowledge the delivery of messages.
    - **Auto Acknowledge**: The session automatically acknowledges a client's receipt of a message either when the session has successfully returned from a call to receive or when the message listener the session has called to process the message successfully returns.
    - **Client Acknowledge**: With this acknowledgment mode, the client acknowledges a consumed message.

> **Note**: This option is relevant only when "isTransacted" is not selected.

Secured Connections to Fiorano MQ (SSL)

If Fiorano MQ server uses SSL then, provide the following properties in Initial Context Properties.

- **java.naming.security.protocol**

  Name of the security protocol used to create secure connections with the MQ server.

  Valid value: The possible values that this variable can take are PHAOS_SSL and SUN_SSL.

- **SecurityManager(optional)**

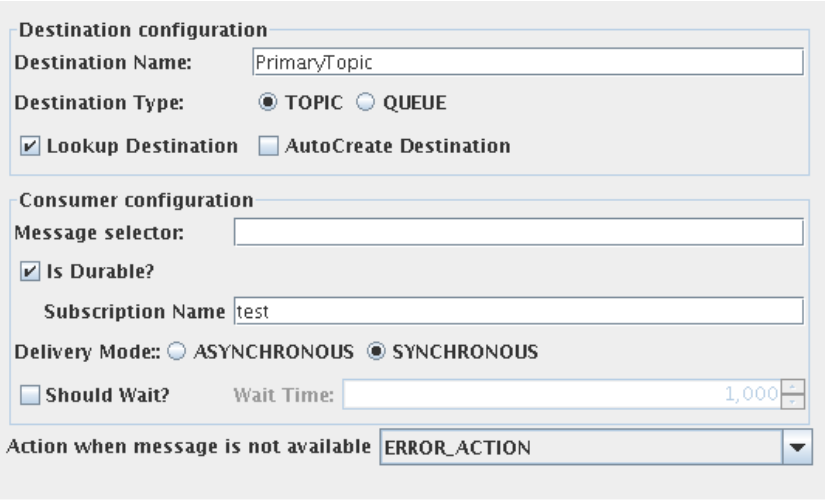
  Note: If this property is not provided then, default security provider will be used.

  The Security Manager implementation used to create secure connections [HTTPS or SSL] with the MQ server. The manager class should be an implementation of the fiorano.jms.runtime.IFMQSecurityManager interface provided by FioranoMQ.

  Valid value: Qualified class name of security Manager class.

4.7.4.1.2 Receive Configuration

The details corresponding to the retrieval of messages from the server can be configured in this panel.



Destination Name: The name of the destination from which the messages have to be retrieved.

Destination Type: The JMS type of the destination. This can be chosen as either Topic or Queue.

Lookup Destination: This option can be chosen if the destination is already present and needs to be looked up using JNDI.

Note: In case of **Active MQ**, to use look up destination, additional JNDI property [topic|queue].<look up name> has to be added as described in the following link. http://activemq.apache.org/jndi-support.html.

AutoCreate Destination:

This option has to be chosen if the destination has to be created.

Note:

- One of "Lookup destination" or "Autocreate destination" is mandatory.
- If both are chosen, the component first tries to lookup the destination and if it fails it will create the destination.

Message Selector: The message selector for the messages to be received by this consumer. More information on message selectors is available at http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/jms/Message.html

Is Durable: This option has to be chosen if a durable subscriber has to be created on the destination.

Note:

- If a durable subscriber is already created by this component and the checkbox is deselected, when the CPS is finished, User is provided with an option to remove durable subscription that is created previously. This can be done by providing the administrator credentials of the JMS Server.
- This option is relevant only when the destination type is selected as "TOPIC" in the destination configuration.
- Selecting this option enables the user to specify the subscription name.

Subscription Name: The name of the subscription corresponding to the durable subscriber that is created by the component.

> **Note**: This option is relevant only when the destination type is selected as "TOPIC" in the destination configuration and the property "Is Durable" is selected.

Delivery Mode:

- **ASYNCHRONOUS**: When this option is selected, the component retrieves messages asynchronously by registering a listener to the destination.
- **SYNCHRONOUS**: When this option is selected, the component retrieves messages when a message is sent on to the input port. Selecting this option enables the property "Should wait"

Should Wait: If this option is selected, the component waits for the time specified by the property "wait time" for the retrieval of messages.

> **Note**: This option is relevant only when the delivery mode is chosen as "SYNCHRONOUS".

Wait time: The time in milliseconds for which the component waits for the message after sending request in synchronous mode.

> **Note**: This option is relevant only when the delivery mode is chosen as "SYNCHRONOUS" and "Should Wait" is selected.

Action when no message is available: The action that has to be taken if the message is not available on the destination in case of synchronous mode. If "Should wait" is enabled, the component waits for the specified timeout before taking the chosen action. Otherwise the action will be done immediately. The action can be chosen as one of the following

- **EMPTY_MESSAGE_ACTION**: An empty message is sent onto the output port of the component.
- **NO_ACTION**: Nothing is done if no message is received.
- **ERROR_ACTION**: The situation will be treated as a request execution error and action will be taken based on the configuration of Error Actions defined in Additional Configuration page.

4.7.4.1.3 Message Definition Configuration

Use XML Interface:

Select this option if the message that is received has to be converted to XML format before sending to the output port of the component.

Message Type:

The type of the message that has to be received can be chosen as one of the following.

- **Text message** - Use this option to receive a plain text message.
- **Map message** - A MapMessage object's message body contains a set of name-value pairs, where names are String objects, and values are Java primitives. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined.
- **Stream message** - A Stream Message object's message body contains a stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format.

Include JMS Headers:

This option can be selected to include JMSHeader information of message in the output XML. An element "JMSHeader" will be added to the output schema and all relevant JMS headers appear as child elements in the output schema as shown in figure.

<ns1:Message>	
<ns1:JMSHeader>	
<ns1:JMSCorrelationID>?	#string
<ns1:JMSDeliveryMode>?	#string
<ns1:JMSDestination>?	
@type	TOPIC | QUEUE
#string	
<ns1:JMSExpiration>?	#string
<ns1:JMSMessageID>?	#string
<ns1:JMSPriority>?	#string
<ns1:JMSRedelivered>?	#string
<ns1:JMSReplyTo>?	#string
<ns1:JMSTimeStamp>?	#string
<ns1:JMSType>?	#string
<ns1:Properties>	
<ns1:ApplicationContext>?	#string
<ns1:Property>*	
@name	#string

Include Properties:

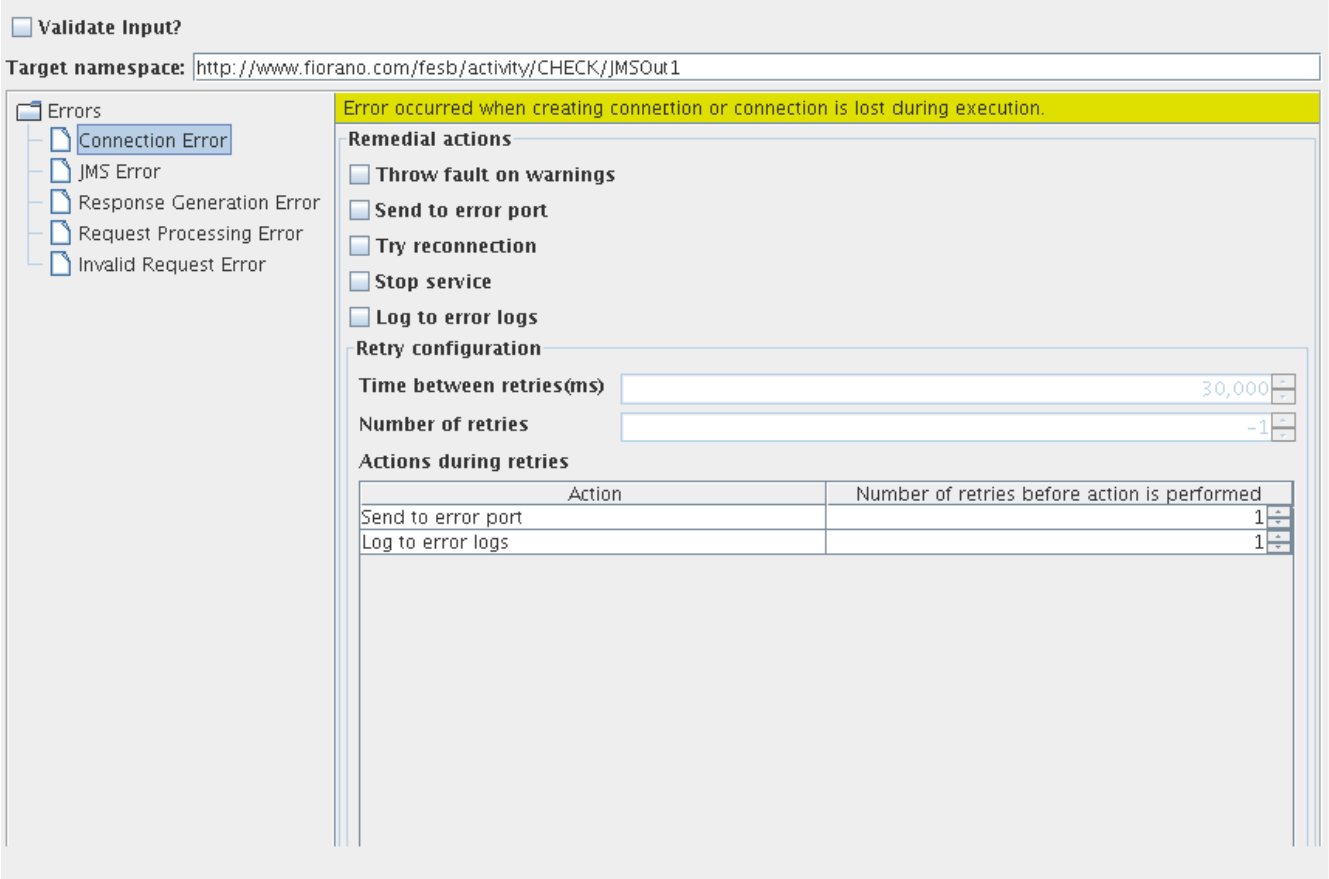
This option can be selected to retrieve the JMS properties on the message.

When this option is selected, the elements "ApplicationContext" and "Property" (ZeroMany) will be generated in the schema of output port.

Depending on the type of the message, the parsing of the message content can be chosen.

- **Text message – XML Content:** This option can be chosen if the content of the output message conforms to a specific schema. The schema can be specified using the schema editor. This schema will be set as the schema of the body element. Setting the schema enables doing necessary transformations.
- **Map message – Define Body:**This option can be chosen to retrieve the body fields of the map message. The message body fields (name and type of the field) can be added in the table and the output schema will be generated corresponding to the each field defined.
- **Stream message – Parse Body:**This option can be chosen to retrieve the content of the stream message. The name, type and length of the field can be added in the table. An element will be added to the schema corresponding to each field with the same name.

4.7.4.1.4 Additional Configuration



Validate Input: This option has to be chosen if the message has to be validated against the schema that is set on the input port.

Target namespace: The target namespace of the schema can be provided here. The schema on ports will be suffixed with the component name and type of the port. Eg. /JMSIn/In

Error Handling Configuration:

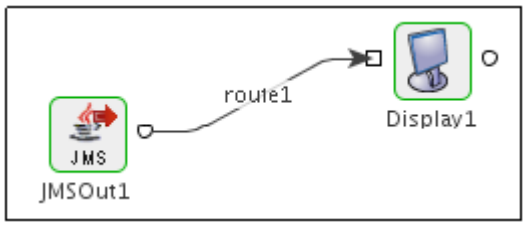
The actions that have to be taken in case of different kinds of errors can be chosen using this panel.

4.7.4.2 Functional Demonstration

4.7.4.2.1 Scenario 1

The component is configured to receive message from "PrimaryQueue" in asynchronous mode. Display component is connected to the component to check the response.

Sample Output:




```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <fns:Message xmlns:fns="http://www.fiorano.com/fesb/activity/JMSOUT/JMSOut1/jmsOut/Out">
3    <fns:Properties>
4       <fns:Property name="name" type="String">String</fns:Property>
5    </fns:Properties>
6    <fns:Body>
7       <fns:key type="unkown"/>
8    </fns:Body>
9 </fns:Message>
```

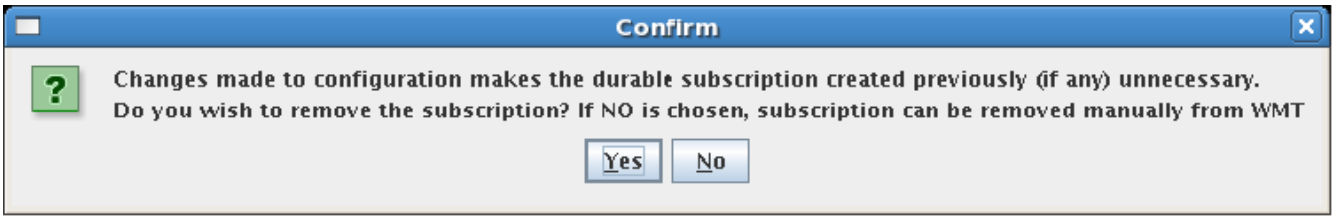
4.7.4.3 Useful Tips

- When using the Durable Subscriber mode, each message has to be stored by the JMS server either in the database or the file system depending on the vendor before delivery of message to consumer and removed after delivery of message. This has a huge impact on the performance. So as far as possible restrict the use of durable delivery mode unless it is necessary for your application to avoid the overheads involved.
- The durable subscription created by this component earlier will become unnecessary if the following gets changed while reconfiguring the component.
    - IsDurable property is deselected in consumer configuration

- Subscription name is changed.
- Response Destination Type is changed from Topic to Queue
- ClientID is changed

Therefore the previous subscription has to be removed to avoid persisting messages in runtimedata.

If the "use specific client ID" is selected in connection configuration step, a dialog **Confirm** will appear prompting the user if the durable subscription has to be removed. The user can select "yes" and then give the admin credentials to remove the previous durable subscription.



Otherwise, the subscription has to be removed manually. For more information on unsubscribing, please refer to Section 29.3 **Connecting to Web Management Tool - DurableSubscribers view** of FioranoMQ Handbook.

4.7.5 JMSReplier

The JMS Replier component is used to retrieve messages from a JMS Topic / Queue and send the response to a configured destination after the message is processed by the flow. Using the CPS, the topic or queue where from the message is to be retrieved can be specified. The message received from the destination should have a JMSReplyTo destination set in the JMS message header properties; else the message not be processed by JMS Replier. The JMS Replier component sends the JMS message received from a Topic / Queue to another component.

After the message is processed by the downstream components, the JMS message should return to the Input Port of the JMSReplier which sends the processed message to the JMSReplyTo destination set in the JMS message header properties. In case an error occurs while processing the message, the message is send to the error destination. In case the error destination is not provided, then the message is send to the JMSReplyTo destination set in the JMS message header properties (which may be a temporary destination).

This component can be used to receive Text, Byte or Map messages.

Note: The only restriction on Map Messages is that this component does not support Objects in Map Messages.

Secured Connections to Fiorano MQ (SSL)

If Fiorano MQ server uses SSL then, provide the following properties in Initial Context Properties.

- **java.naming.security.protocol**

  Name of the security protocol used to create secure connections with the MQ server.

Valid value: The possible values that this variable can take are PHAOS_SSL and SUN_SSL.

- **SecurityManager(optional)**

  Note: If this property is not provided then, default security provider will be used.

  The Security Manager implementation used to create secure connections [HTTPS or SSL] with the MQ server. The manager class should be an implementation of the fiorano.jms.runtime.IFMQSecurityManager interface provided by FioranoMQ.

  Valid value: Qualified class name of security Manager class.

4.7.5.1 Configuration and Testing

The fields that make JMS connection etc. can be configured in the CPS.

Attributes	
AdvancedInfo	{}
Server URL	http://localhost:1956
Destination Type	Topic
IsBackupURLsRequired	no
Destination Name	PrimaryTopic
JNDI user Name	anonymous
JNDI Password	**********
Time-to-live	0
InitialContextFactory	fiorano.jms.runtime.naming.FioranoInitialContextFactory
InputMessageConfig	Click here to see details...
ConnectionFactoryName	PrimaryCF
OutputMessageConfig	Click here to see details...
JMS User Name	ayrton
JMS Password	******
AutoCreateDestination	yes
ClientID	
Delivery Mode of message	Non-Persistent Mode
Priority of message	4
IsDurableSubscriber	no
Acknowledge Mode	Auto Acknowledge
Message Ack Count	1
Proxy Settings	Click ... to edit

Figure 4.327: Sample JMS Replier component configuration

4.7.5.1.1 Using Named Configurations

Changes are done in eStudio w.r.t to Named Configurations. With the implementation of Named Configurations, Component Configuration Panel is changed as shown in Figure below.

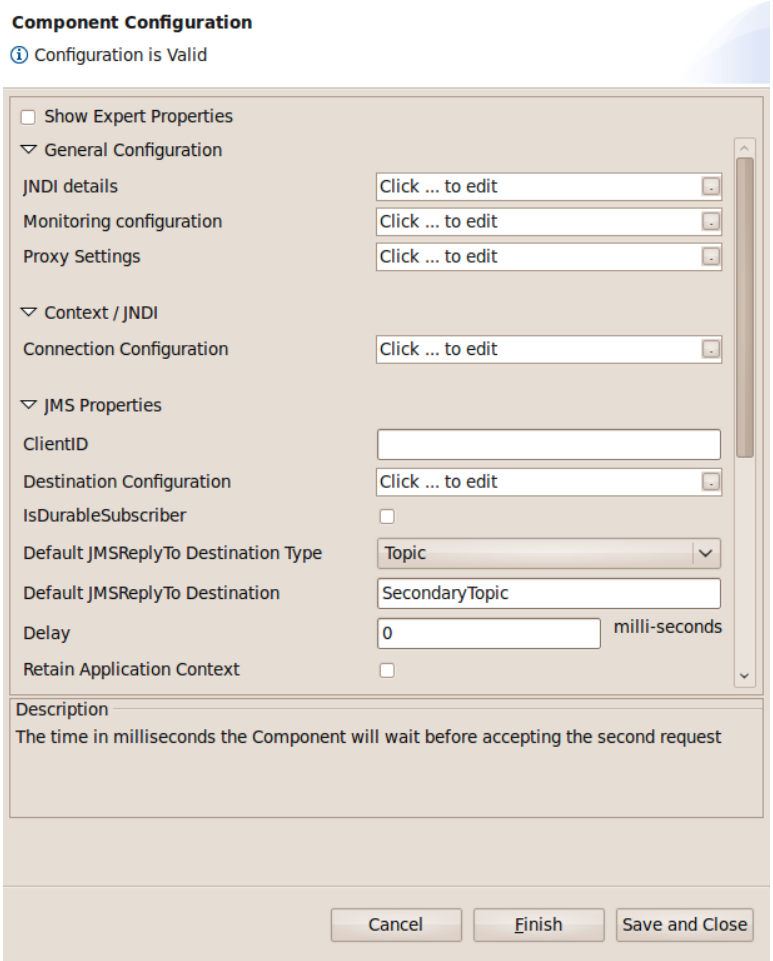


Figure: Modified Managed Connection Factory

Some properties are grouped as one property in the modified configuration panel. ServerURL, BackupURL, CF lookup name, JMS username, JMS password which were present previously in the Component Configuration panel are grouped under the name Connection Configuration.

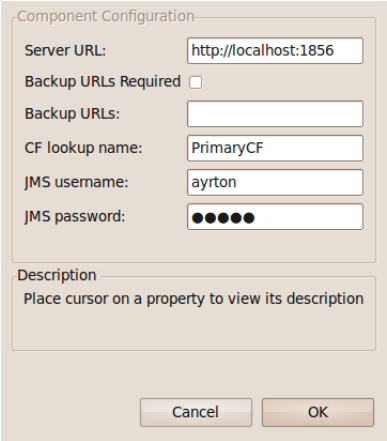


Figure: Connection Configuration

Initial Context Factory, JNDI username, CF lookup name, JNDI password which were present previously in the Component Configuration panel are grouped under the name JNDI Settings.

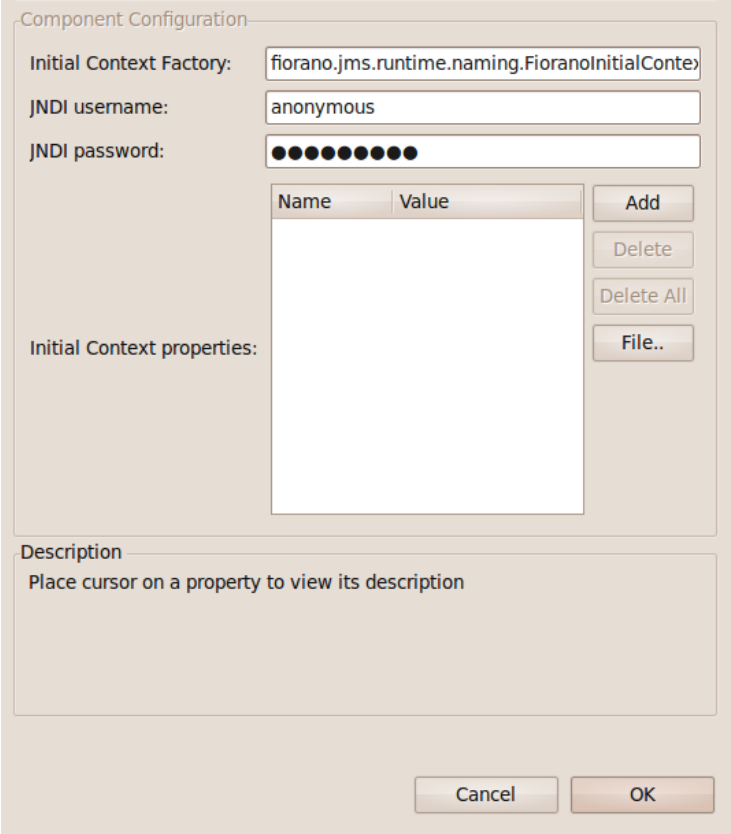


Figure: JNDI Settings

Destination Name, Destination Type, Lookup Destination which were present previously in the Component Configuration panel are grouped under the name Destination Configuration.

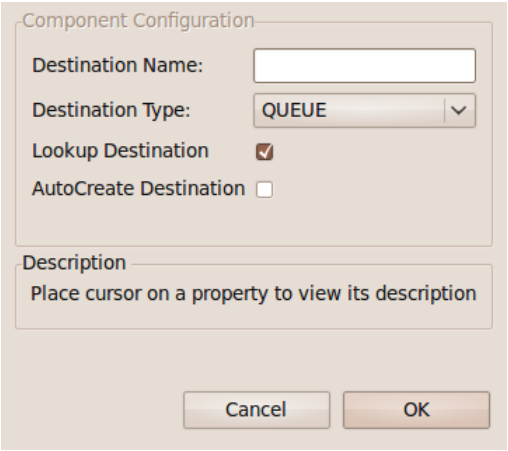


Figure: Destination Configuration

Delivery Mode, Time to Live, Priority which were present previously in the Component Configuration panel are grouped under the Producer Configuration.

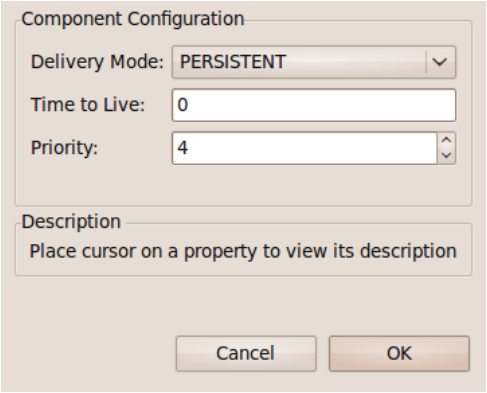


Figure: Producer Configuration

Similarly SSL Security details and Proxy Settings Panels are modified to use Named Configurations.

4.7.5.2 Input Schema

The input schemas are auto generated based on the configuration provided (type of message). For the configuration shown above, the schemas would be

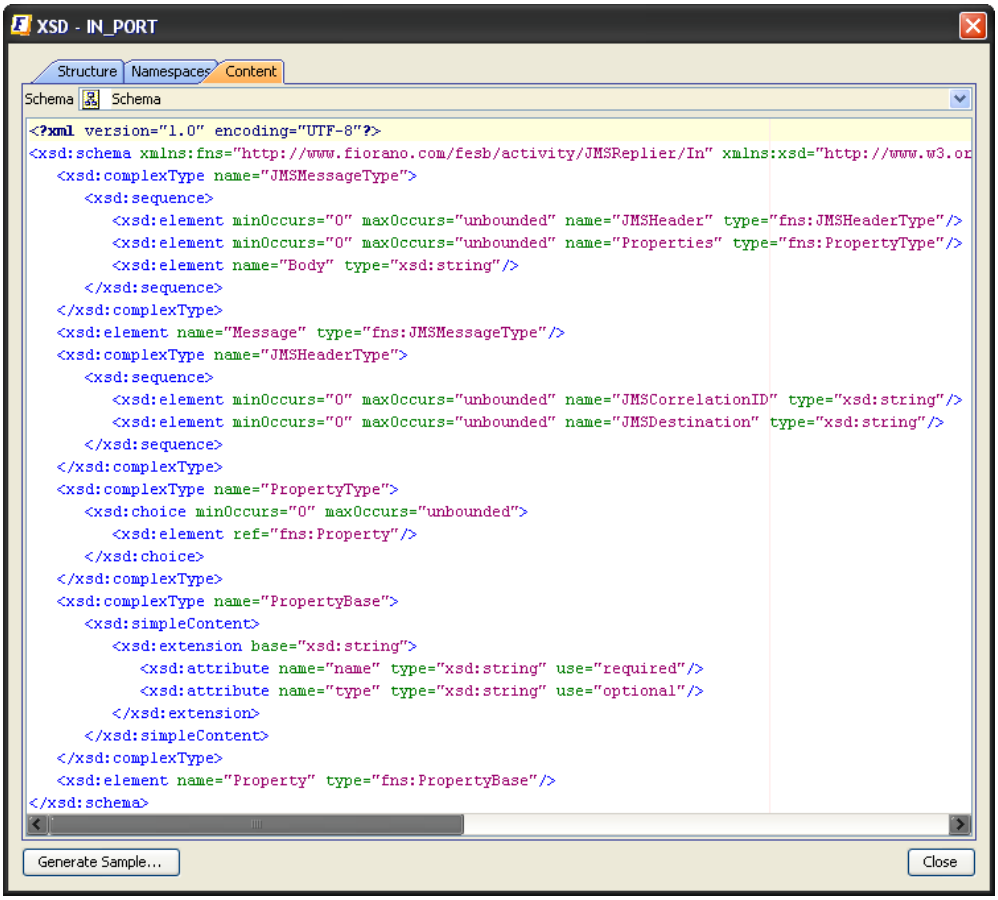
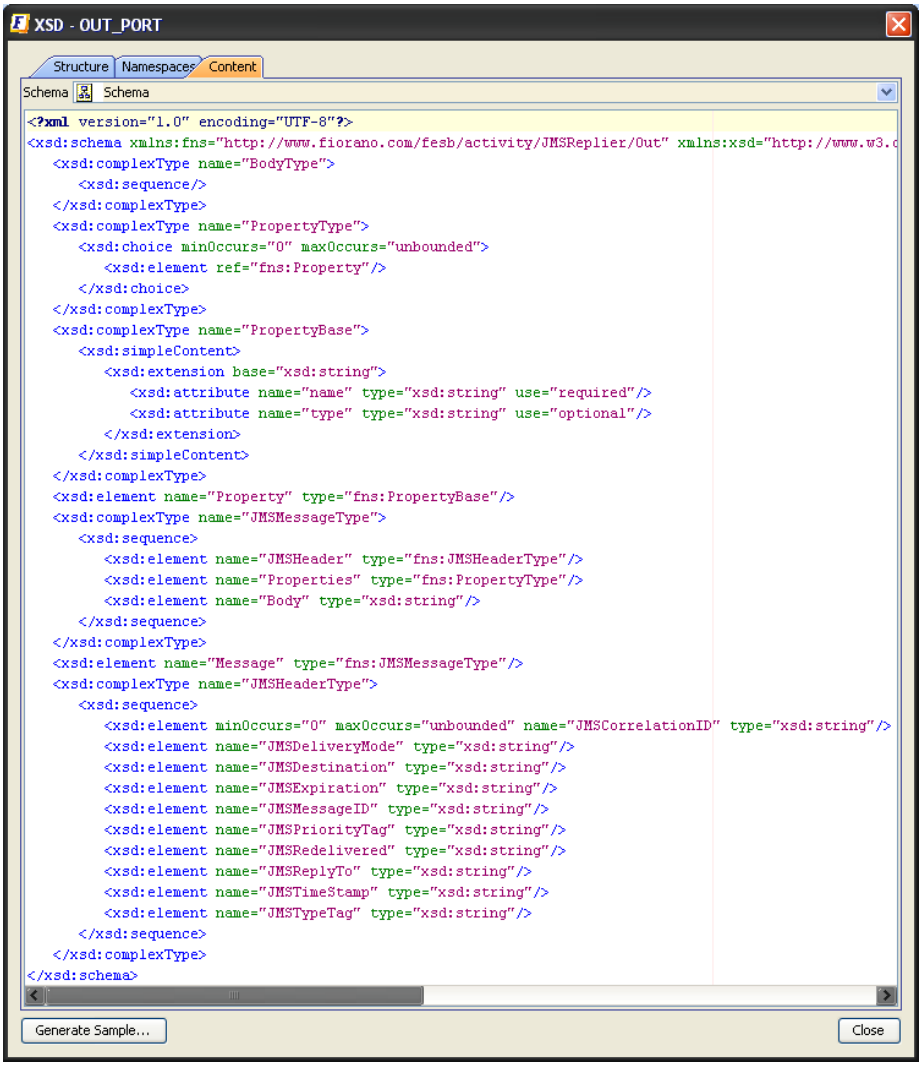


Figure 4.328: Input Schema

4.7.5.3 Output Schema

The output schema is auto generated based on the configuration provided (type of message). For the configuration shown above, the schema would be



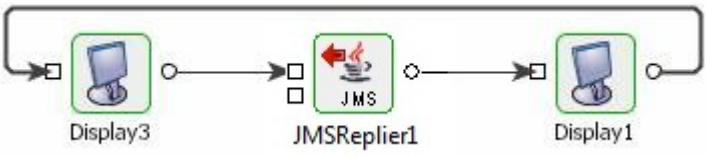

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:fns="http://www.fiorano.com/fesb/activity/JMSReplier/Out" xmlns:xsd="http://www.w3.c
   <xsd:complexType name="BodyType">
      <xsd:sequence/>
   </xsd:complexType>
   <xsd:complexType name="PropertyType">
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
         <xsd:element ref="fns:Property"/>
      </xsd:choice>
   </xsd:complexType>
   <xsd:complexType name="PropertyBase">
      <xsd:simpleContent>
         <xsd:extension base="xsd:string">
            <xsd:attribute name="name" type="xsd:string" use="required"/>
            <xsd:attribute name="type" type="xsd:string" use="optional"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
   <xsd:element name="Property" type="fns:PropertyBase"/>
   <xsd:complexType name="JMSMessageType">
      <xsd:sequence>
         <xsd:element name="JMSHeader" type="fns:JMSHeaderType"/>
         <xsd:element name="Properties" type="fns:PropertyType"/>
         <xsd:element name="Body" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:element name="Message" type="fns:JMSMessageType"/>
   <xsd:complexType name="JMSHeaderType">
      <xsd:sequence>
         <xsd:element minOccurs="0" maxOccurs="unbounded" name="JMSCorrelationID" type="xsd:string"/>
         <xsd:element name="JMSDeliveryMode" type="xsd:string"/>
         <xsd:element name="JMSDestination" type="xsd:string"/>
         <xsd:element name="JMSExpiration" type="xsd:string"/>
         <xsd:element name="JMSMessageID" type="xsd:string"/>
         <xsd:element name="JMSPriorityTag" type="xsd:string"/>
         <xsd:element name="JMSRedelivered" type="xsd:string"/>
         <xsd:element name="JMSReplyTo" type="xsd:string"/>
         <xsd:element name="JMSTimeStamp" type="xsd:string"/>
         <xsd:element name="JMSTypeTag" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

Figure 4.329: Output Schema

4.7.5.4 Functional Demonstration

4.7.5.4.1 Scenario 1

Replying on a temporary destination set by request / reply invocation



Configure the JMS Replier component as described in *Configuration and Testing section* and use display component to receive the sample message receive and response sent back.

Figure 4.330: Demonstrating Scenario 1 with sample input and output

4.7.5.5 Use Case Scenario

In an order entry system scenario, the JMS Replier can be used for receiving and acknowledging purchase orders.

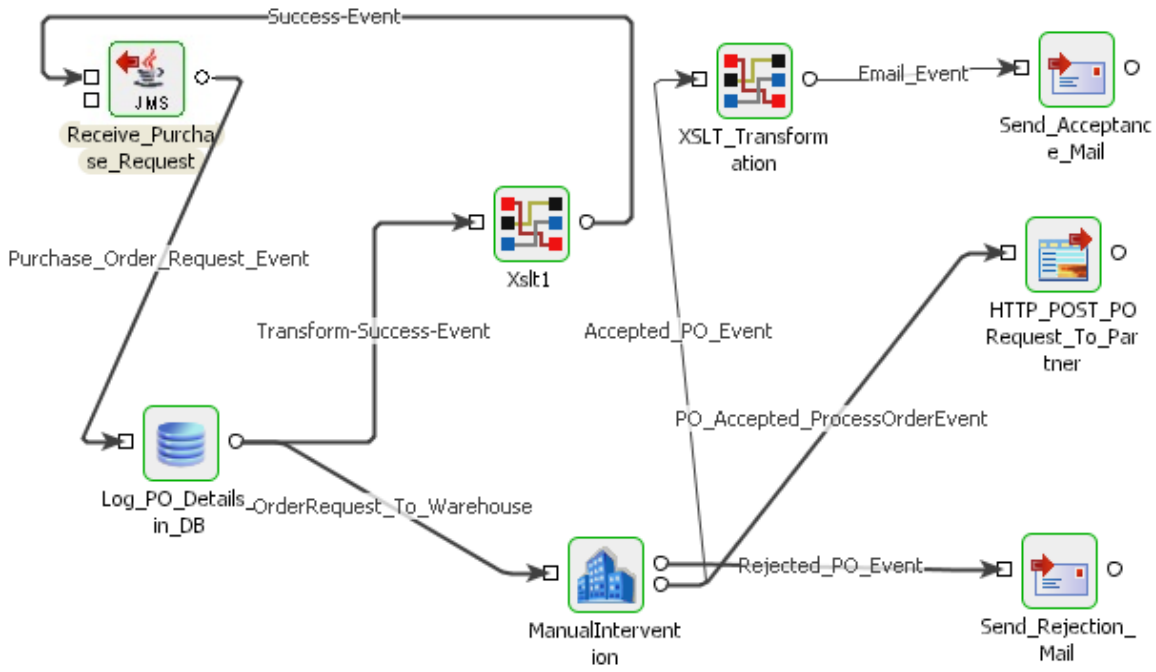


Figure 4.331: Order entry system scenario

The event process demonstrating this scenario is bundled with the installer. The JMS Replier has been used instead of the HTTP Receive adapter as we are replacing the transport from HTTP to JMS.

Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Studio.

4.7.5.6 Useful Tips

- The only restriction on Map Messages is that this component does not support Objects in Map Messages.
- If IsDurableSubscriber property is changed to No or Destination Type is changed from Topic to Queue or ClientID is changed - the durable subscriptions created earlier should be unsubscribed to avoid persisting messages for the durable subscriber in runtimedata. For more information on unsubscribing, please refer to **FioranoMQ Handbook - Chapter29: FioranoMQ Web Management Tool** - Section 29.3: Connecting to Web Management Tool - DurableSubscribers view.

4.7.6 JMSRequestor 4.0

The JMSRequestor component is used to send messages to a JMS Topic/Queue and wait till a message reply is received from the same. After sending the message to the Destination set in the CPS, JMSRequestor waits for the response message on the Response Destination. If Response Destination is not specified in the CPS, then a temporary destination is created on which the response message is expected. User can also specify an Error Destination on which JMSRequestor receives the error message.

JMSRequestor is capable of handling following types of messages:

- Text messages
- A TextMessage object's message body contains a java.lang.String object.

Text Message

A TextMessage object's message body contains a java.lang.String object.

Map messages

A MapMessage object's message body contains a set of name-value pairs, where names are String objects, and values are Java primitives. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined.

Bytes Message

A BytesMessage object's message body contains a stream of uninterrupted bytes. This message type is for literally encoding a body to match an existing message format.

JMSRequestor uses JMS APIs to process the messages.

Points to note

- The only restriction on Map Messages is that this component does not support Objects in Map Messages.
- When adding the Initial Context Factory class for non Fiorano MQ server, the jar file(s) should be added as resource(s) to the JMSAdapters system library.
- JMSRequestor will not process further messages it receives on its input port till it receives the response for the current message.
- If IsDurableSubscriber property is changed to No or Destination Type is changed from Topic to Queue or ClientID/Subscriber ID is changed - the durable subscriptions created earlier should be unsubscribed to avoid persisting messages for the durable subscriber in runtimedata. For more information on unsubscribing, please refer to **FioranoMQ Handbook - Chapter29: FioranoMQ Web Management Tool** - Section *29.3 Connecting to Web Management Tool - DurableSubscribers view*.

4.7.6.1 Configuration and Testing

The JMSRequestor component connection related properties can be configured in the Managed Connection Factory panel of CPS.

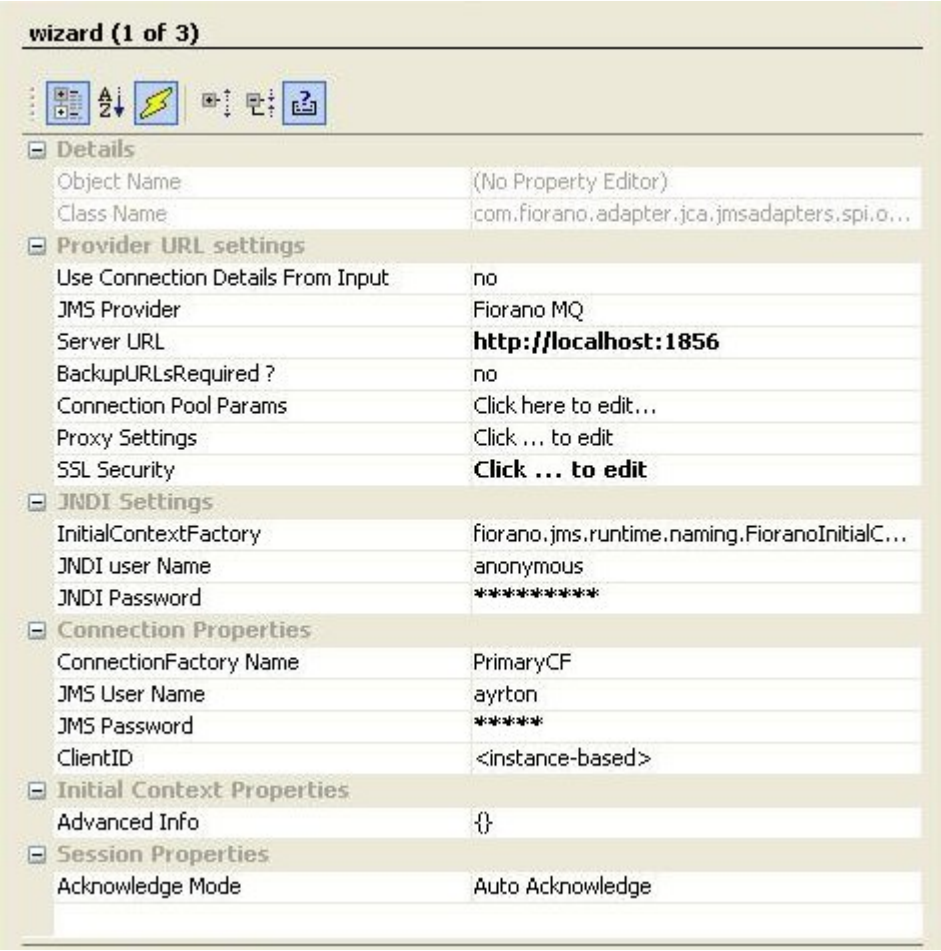


Figure 4.332: Sample JMSRequestor MCF configuration

Use Connection Details From Input: Parameters to create the connection can be specified in the input message when this property is set to true. If this property is selected the validation errors in the managed connection factory panel of the CPS are treated as warnings. So user can bypass this step without giving valid configuration and complete the configuration of the component. If valid properties are not provided even in the input message exception will be thrown at runtime.

The JMS providers supported are Fiorano MQ, BEA Weblogic, Oracle AQ and Oracle Streams AQ. For working with these JMS providers the jars that should be explicitly added as resources to the component are given below:

BEA WebLogic:

- %BEA_HOME%\server\lib\wlclient.jar
- %BEA_HOME%\server\lib\wljmsclient.jar (required only when the Weblogic server is running on a remote machine)

Note: For BEA Weblogic 10.0, %BEA_HOME% refers to <BEA WebLogic Installation directory>\wlserver_10.0

Oracle AQ:

- %ORACLE_HOME%\rdbms\jlib\aqapi13.jar (If JDK1.2 / JDK1.1 is used, aqapi12.jar/aqapi11.jar has to be used respectively)
- %ORACLE_HOME%\jdbc\lib\classes12.zip
- %ORACLE_HOME%\jdbc\lib\nls_charset12.jar

  Note: For Oracle Database 9.2.0.1.0, %ORACLE_HOME% refers to <Oracle Installation directory>\ora92

Oracle Streams AQ:

- %OR ACLE_HOME%\rdbms\jlib\aqapi.jar
- %ORACLE_HOME%\jdbc\lib\ojdbc14.jar

  Note: For Oracle Database 10.2.0.1.0, %ORACLE_HOME% refers to <Oracle Installation directory>\product\10.2.0\db_1

Note the following:

- If these jars are added to resources of the System library JMSAdapters, the jars is available for JMSIn, JMSOut and JMSRequestor components.
- In case of BEA Weblogic, InitialContext can be created by specifying empty values for JNDI Username and JNDI password as well.

Secured Connections to Fiorano MQ (SSL)

If Fiorano MQ server uses SSL then, provide the following properties in Advanced Info Under Initial Context Properties.

- **java.naming.security.protocol**

  Name of the security protocol used to create secure connections with the MQ server.

  Valid value: The possible values that this variable can take are PHAOS_SSL and SUN_SSL.

- **SecurityManager(optional)**

  Note: If this property is not provided then, default security provider will be used.

  The Security Manager implementation used to create secure connections [HTTPS or SSL] with the MQ server. The manager class should be an implementation of the fiorano.jms.runtime.IFMQSecurityManager interface provided by FioranoMQ.

  Valid value: Qualified class name of security Manager class.

Server connection can be tested from within the CPS by clicking on **test** in the connection properties panel.

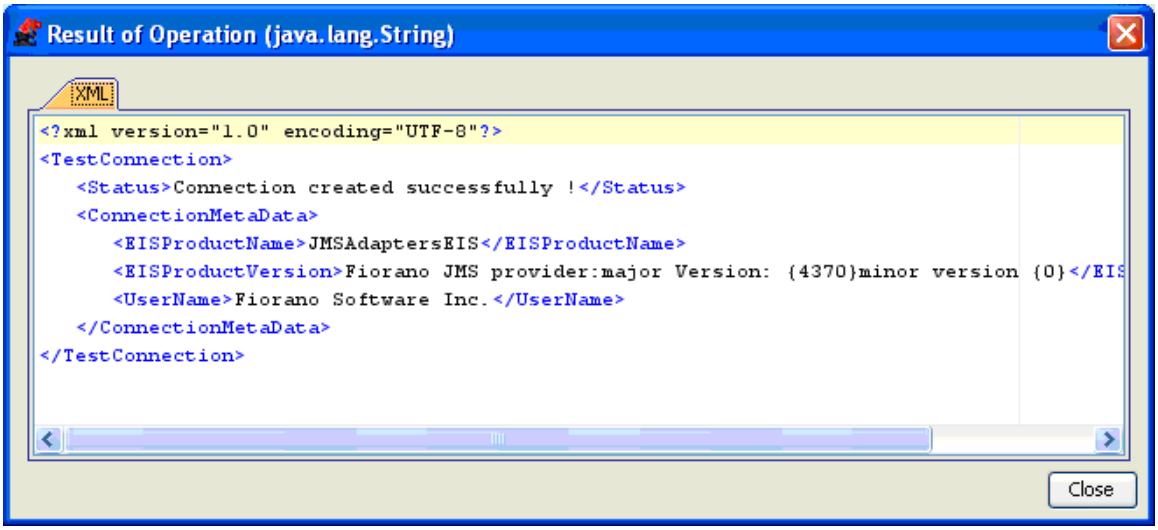


Figure 4.333: Sample connection test result indicating success

The JMSRequestor component can be configured using its Custom Proper Sheet wizard.

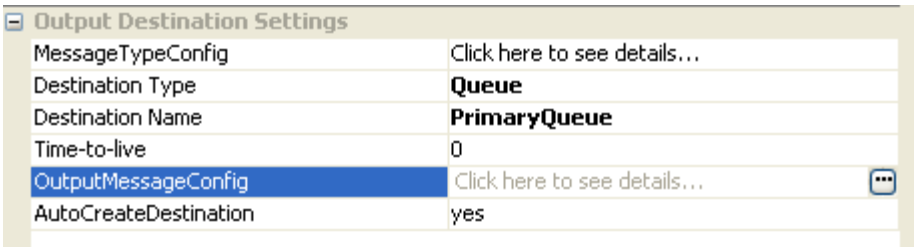


Figure 4.334: Sample JMSRequestor configuration

Destination specified for the property **Destination Name** should already exist if BEA Weblogic/Oracle AQ/Oracle Streams AQ is being used. Dynamic creation of destinations is not supported for these providers.

For BEA Weblogic, value for **Destination Name** should be the JNDI name of the destination. **Example:** weblogic.examples.jms.exampleTopic.

Above configuration shown in the **Figure 4.334** can be tested from within the CPS by clicking on test button in the CPS panel.

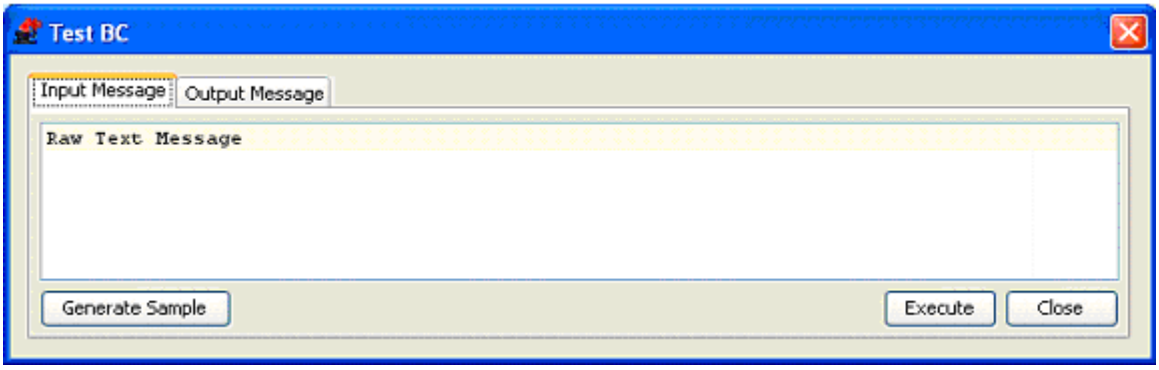


Figure 4.335: Sample JMSRequestor input message

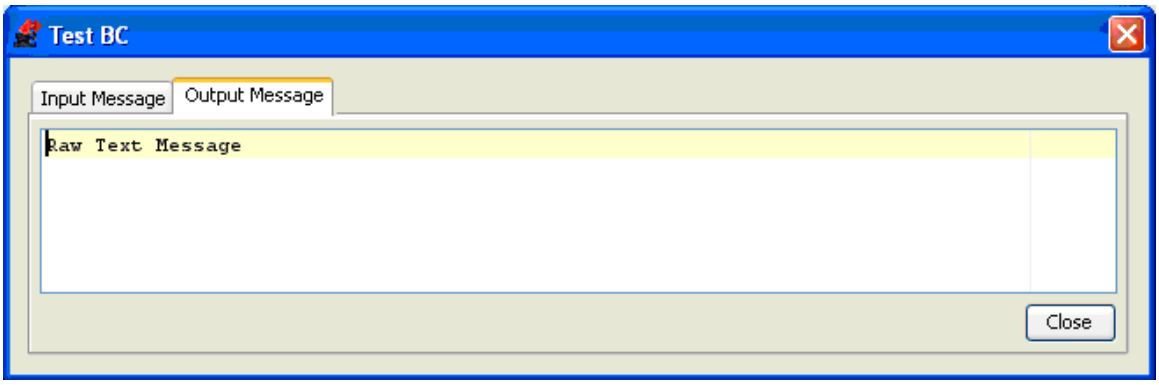


Figure 4.336: Sample JMSRequestor output message

Input Schema

When the property Use Connection details from input is chosen, the following input schema with the element ConnectionFactorySettings, is generated. Properties that are used to create the connection are present under this element.

XML Item	Value Type
<ns1:Message>	
<ns1:ConnectionFactorySettings>?	
<ns1:ProviderUrlSettings>?	
<ns1:JMSProvider>?	Fiorano MQ | BEA WebLogic | Or...
<ns1:ServerUrl>?	#string
<ns1:BackupUrlsRequired>?	true | false
<ns1:BackupUrls>?	#string
<ns1:JNDISettings>?	
[Choice]	
<ns1:BEAWEBLogic>	
<ns1:IntialContextFactory>?	#string
<ns1:JNDIUserName>?	#string
<ns1:JNDIPassword>?	#string
<ns1:FioranoMQ>	
<ns1:IntialContextFactory>?	#string
<ns1:JNDIUserName>?	#string
<ns1:JNDIPassword>?	#string
<ns1:OracleAQ>	
<ns1:IntialContextFactory>?	#string
<ns1:DestinationType>?	Queue | Topic
<ns1:OracleStreamsAQ>	
<ns1:IntialContextFactory>?	#string
<ns1:Other>	
<ns1:IntialContextFactory>?	#string
<ns1:JNDIUserName>?	#string
<ns1:JNDIPassword>?	#string
<ns1:ConnectionProperties>?	
<ns1:ConnectionFactoryName>?	#string
<ns1:JMSUserName>?	#string
<ns1:JMSPassword>?	#string
<ns1:ClientID>?	#string
<ns1:IntialContextProperties>?	
<ns1:AdvancedInfo>?	#string
<ns1:SessionProperties>?	
<ns1:AcknowledgeMode>?	Auto Acknowledge | Client Ackn...
<ns1:MessageAckCount>?	#int
<ns1:SSLSettings>?	

Figure: Input schema with ConnectionFactorySettings

4.7.6.1.1 Using Named Configurations

Changes are done in eStudio w.r.t to Named Configurations.With the implementation of Named Configurations MCF panel is changed as shown in Figure below.

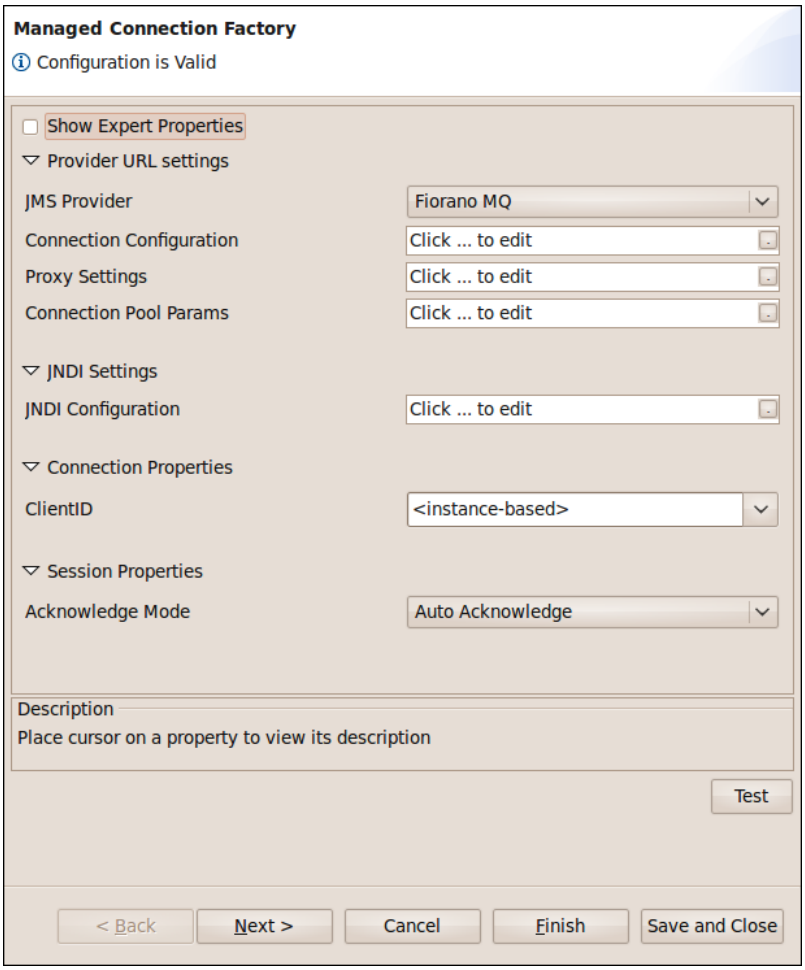


Figure: Modified Managed Connection Factory

Some properties are grouped as one property in the modified configuration panel. ServerURL, BackupURL, CF lookup name, JMS username, JMS password which were present previously in the MCF panel are grouped under the name ConnectionConfiguration.

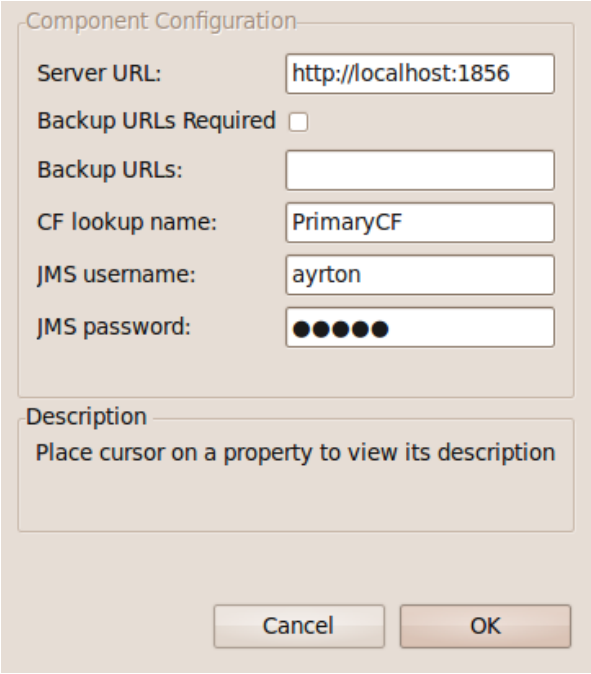


Figure: Connection Configuration

Initial Context Factory, JNDI username, CF lookup name, JNDI password which were present previously in the MCF panel are grouped under the name JNDI Settings.

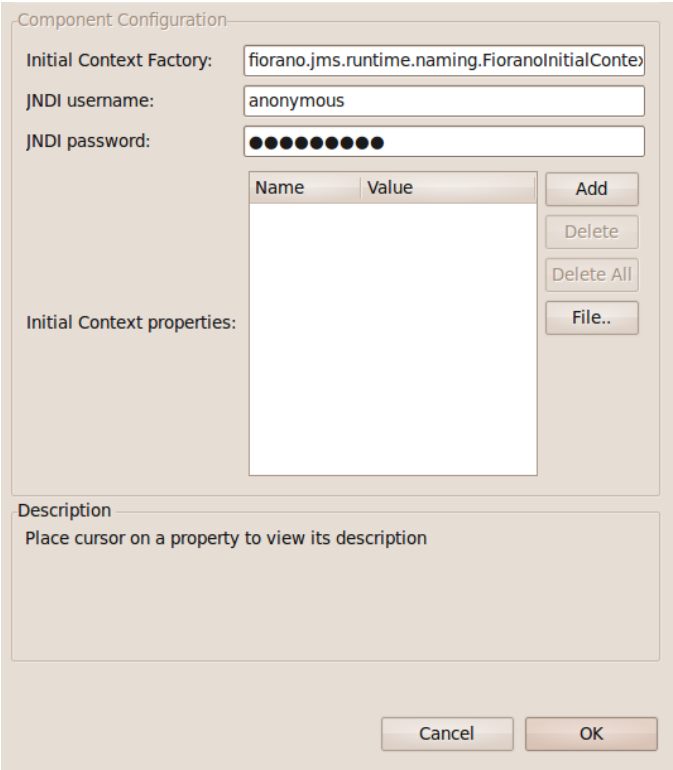


Figure: JNDI Settings

Destination Name, Destination Type, Lookup Destination which were present previously in the Interaction Configurations panel are grouped under the name Destination Configuration.

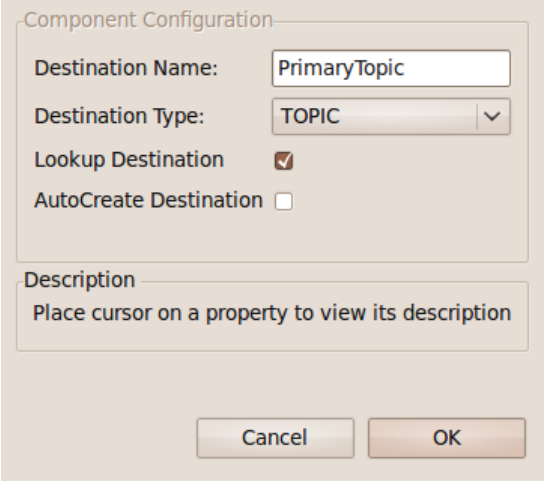


Figure: Destination Configuration

Delivery Mode, Time to Live, Priority which were present previously in the Interaction Configurations panel are grouped under the Producer Configuration.

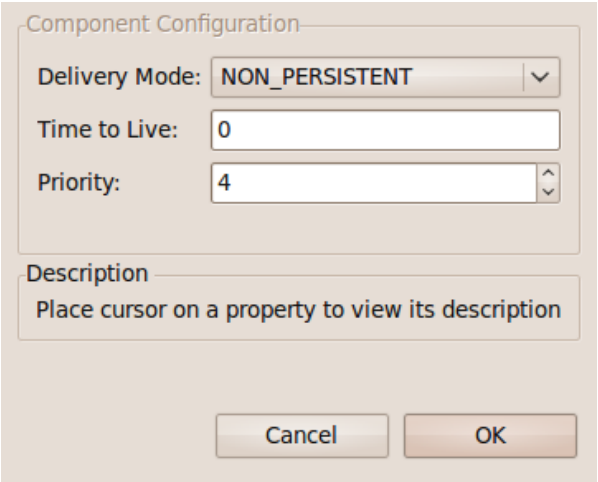


Figure: Producer Configuration

Is Durable and Subscription Name which were present previously in the Interaction Configurations panel are grouped under the name Consumer Configuration.

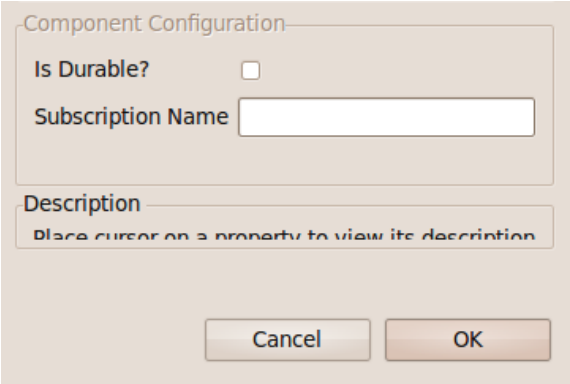


Figure: Consumer Configuration

Similarly SSL Security details and Proxy Settings Panels are modified to use Named Configurations.

4.7.6.2 Functional Demonstration

4.7.6.2.1 Scenario 1

Putting a simple Text message on a destination and displaying the response message.

Configure the JMSRequestor as described in *Configuration and Testing* section and use feeder and display component to send sample input and check the response respectively.

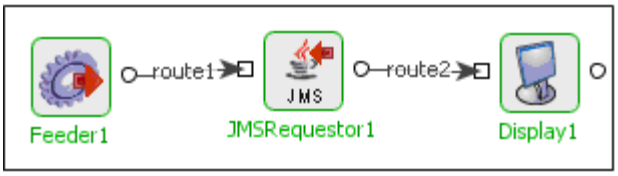


Figure 4.337: Demonstrating Scenario 1 with sample input and output

Sample Input:

```
Input Text
```

Sample Output:

```
Raw Text Message
```

4.7.7 JMSRequestor 5.0

The JMSRequestor component is used to send messages to a JMS Topic/Queue and wait till a message reply is received from the same. After sending the message to the Destination set in the CPS, JMSRequestor waits for the response message on the Response Destination. If Response Destination is not specified in the CPS, then a temporary destination is created on which the response message is expected. User can also specify an Error Destination on which JMSRequestor receives the error message.

Points to Note:

- The JMS providers supported are Fiorano MQ and BEA Weblogic. For working with these JMS providers the jars that should be explicitly added as resources to the component are given below:

  BEA WebLogic:

  - %BEA_HOME%\server\lib\wlclient.jar
  - %BEA_HOME%\server\lib\wljmsclient.jar (required only when the Weblogic server is running on a remote machine)

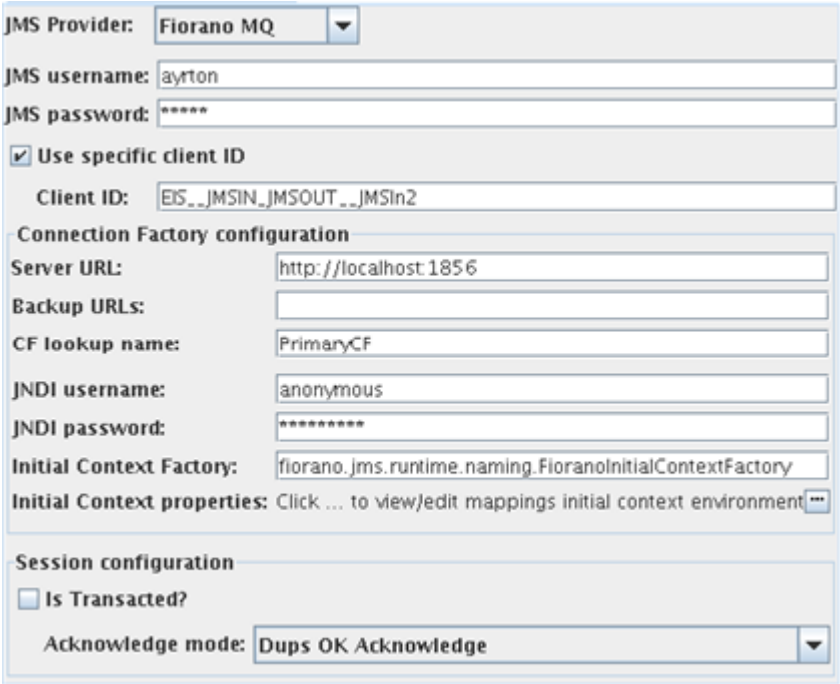
  Note: For BEA Weblogic 10.0, %BEA_HOME% refers to <BEA WebLogic Installation directory>\wlserver_10.0. If these jars are added to resources of the System library JMSAdapters, the jars will be available for JMSIn, JMSOut & JMSRequestor components.

- In case of BEA Weblogic, InitialContext can be created by specifying empty values for JNDI Username and JNDI password as well.
- When using the Durable Subscriber mode, each message has to be stored by the JMS server either in the database or the file system depending on the vendor before delivery of message to consumer and removed after delivery of message. This has a huge impact on the performance. So as far as possible restrict the use of durable delivery mode unless it is necessary for your application to avoid the overheads involved.

4.7.7.1 Configuration

4.7.7.1.1 Connection Configuration

Connection configuration details can be specified in this panel.



JMS Implementation

Choose the MQ server from the dropdown list provided.

JMS Username:

The user name with which the client connects to the MQ server.

Note: When the username is not provided, the password is ignored too and JMS connection creation is attempted without username and password.

JMS Password:

The password for the username that is provided.

Use specific client ID:

Select this option to specify a user defined Client ID. If it is not selected client ID is generated automatically at runtime.

Note: If user defined client ID is used in the component and if durable subscription is enabled, while reconfiguring the component if the durable subscription changes, then the previously created durable subscription will also receive the messages. To avoid this there is an option to remove the previously created durable subscription. Refer "Useful tips" for more details.

Client ID:

The client ID that will be set on the connection with the MQ server.

Note: This will be used only when the option "Use specific client ID" is selected.

Server URL:

The URL of the server to which the component connects.

Backup URLs:

The backup URLs to which the component tries to connect if the server specified by the property "server URL" is down. Multiple backup URLs can be specified by separating them with ";".

CF lookup name:

The lookup name of the connection factory.

JNDI Username:

The name with which the user connects to the JNDI server to perform lookup operations.

JNDI Password:

The password for the JNDI username.

Initial context factory:

The name of the initial context factory.

Initial context properties:

Additional properties which can be used for creating an Initial Context can be specified here. For instance, if JMSAdapter has to connect to the FMQ server using HTTP protocol instead of the default TCP protocol, then a property 'TransportProtocol' can be specified with the value 'HTTP'.

- **Is transacted**: Select this option if session to be created should be transacted. If this is not selected appropriate Acknowledge mode has to be chosen.
- **Acknowledgement mode**: The type of Acknowledgement mode can be specified here. A session retains messages it consumes until they have been acknowledged.
    - **Dups OK Acknowledge**: This acknowledgment mode instructs the session to lazily acknowledge the delivery of messages.
    - **Auto Acknowledge**: The session automatically acknowledges a client's receipt of a message either when the session has successfully returned from a call to receive or when the message listener the session has called to process the message successfully returns.
    - **Client Acknowledge**: With this acknowledgment mode, the client acknowledges a consumed message.

Note: This option is relevant only when "isTransacted" is not selected.

Secured Connections to Fiorano MQ (SSL)

If Fiorano MQ server uses SSL then, provide the following properties in Initial Context Properties.

- **java.naming.security.protocol**

  Name of the security protocol used to create secure connections with the MQ server.

  Valid value: The possible values that this variable can take are PHAOS_SSL and SUN_SSL.

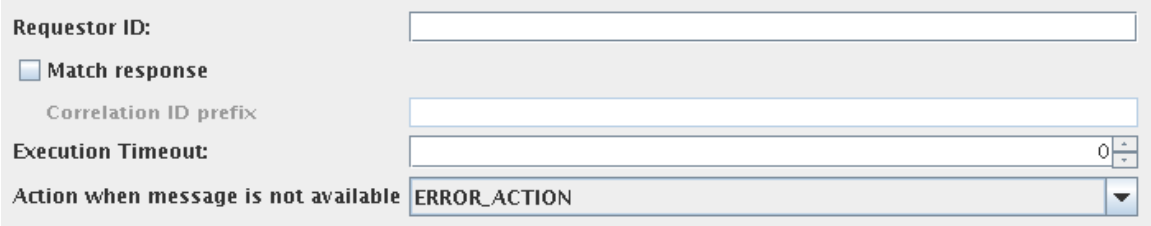
- **SecurityManager(optional)**

  Note: If this property is not provided then, default security provider will be used.

  The Security Manager implementation used to create secure connections [HTTPS or SSL] with the MQ server. The manager class should be an implementation of the fiorano.jms.runtime.IFMQSecurityManager interface provided by FioranoMQ.

  Valid value: Qualified class name of security Manager class.

4.7.7.1.2 Requestor Configuration



Requestor ID: The ID specific to an instance of requestor. This is used to set the message selector on the response destination. Providing a value is optional. If no value is provided then a unique requestor ID will be automatically generated by the component.

Match Response: This option has to be chosen if the response that is received from the response destination has to be matched with the request that is sent based on the header JMSCorrelationID. Choosing this option enables the property "CorrelationID prefix".

CorrelationID prefix: The String that has to be prefixed to the correlationID that is generated. Providing value is optional. If no value is provided, a default value "Correlation_" will be generated and used by the component.

Execution Timeout: The time in milliseconds for which the component has to wait for a response from the response destination after sending a request. A value of 0 indicates infinite timeout.

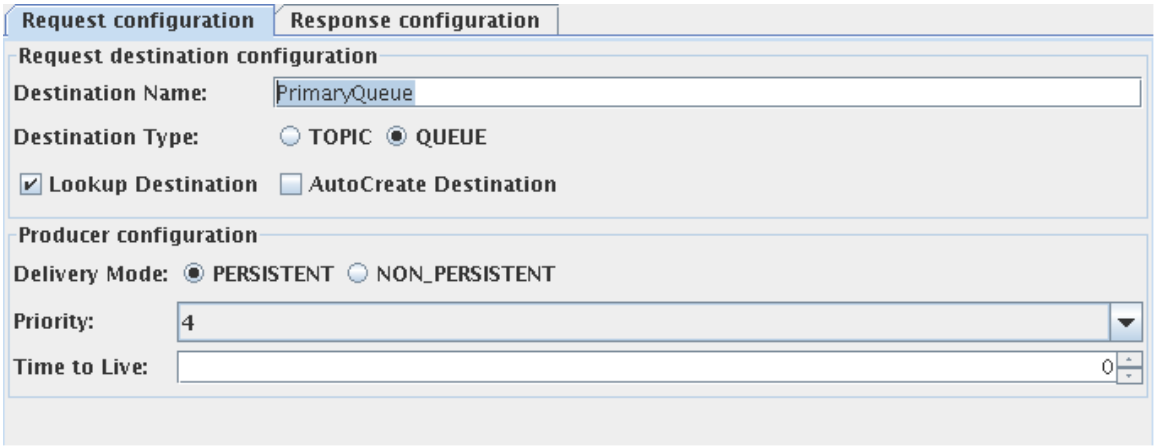
Action when message is not available: The action that has to be performed when no message is received within the timeout specified by the property "Execution Timeout".

The action can be chosen as one of the following

- **EMPTY_MESSAGE_ACTION**: An empty message is sent onto the output port of the component.
- **NO_ACTION**: Nothing is done if no message is received.

- **ERROR_ACTION**: The situation will be treated as a request execution error and action will be taken based on the configuration of Error Actions defined in Additional Configuration page.

4.7.7.1.3 Request Configuration



The details of the destination to which the request message from input port has to be sent can be configured using the Request configuration tab.

Destination Name: The name of the destination to which the messages have to be sent.

Destination Type: The JMS type of the destination. This can be chosen as either Topic or Queue.

Lookup Destination: This option can be chosen if the destination is already present and needs to be looked up using JNDI.

Note: In case of Active MQ, to use look up destination, additional JNDI property [topic|queue]. <look up name> has to be added as described in the following link. http://activemq.apache.org/jndi-support.html.

AutoCreate Destination: This option has to be chosen if the destination has to be created, if it doesn't exist.

Note:

- One of "Lookup destination" or "Autocreate destination" is mandatory.
- If both are chosen, the component first tries to lookup the destination and if it fails it will create the destination.

4.7.7.1.4 Producer Configuration

Delivery Mode:

- **PERSISTENT**: The PERSISTENT delivery mode, instructs the JMS provider to take extra care to ensure that a message is not lost in transit in case of a JMS provider failure. A message sent with this delivery mode is logged to stable storage when it is sent.
- **NON-PERSISTENT**: The NON_PERSISTENT delivery mode does not require the JMS provider to store the message or otherwise guarantee that it is not lost if the provider fails.

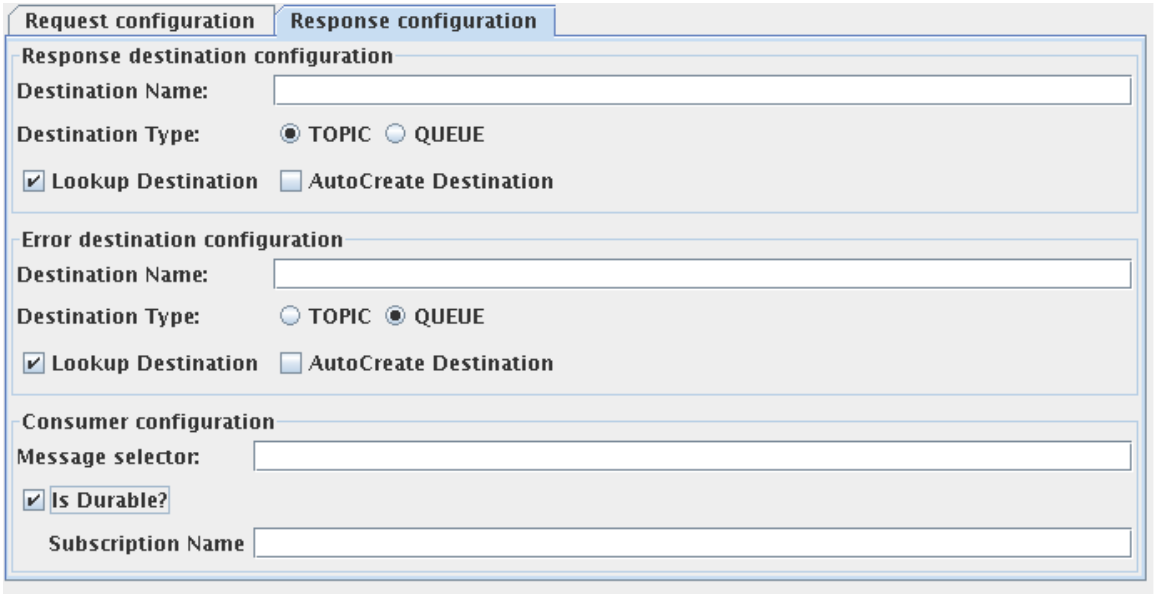
Priority:

The priority of the message to be sent to the destination.

Time to Live:

The time to live (in milliseconds) of the message to be sent to the destination. After the timeout the message will be discarded.

4.7.7.1.5 Response Configuration



Response Destination Configuration:

The details of the destination to which the response is sent can be specified here.

Destination Name:

The name of the destination on which the response for the request will be received. This is optional. If the destination name is not defined, then a temporary destination will be created and will be set as the JMSReplyTo Destination on the request message. The type of the temporary destination created will be same as the type of the request destination.

Destination Type:

The JMS type of the destination. This can be chosen as either Topic or Queue. This option is relevant only when the destination name is not empty.

Lookup Destination:

This option can be chosen if the destination is already present and needs to be looked up using JNDI.

Note: In case of **Active MQ**, to use look up destination, additional JNDI property [topic|queue].<look up name> has to be added as described in the following link. http://activemq.apache.org/jndi-support.html.

AutoCreate Destination:

This option has to be chosen if the destination has to be created.

Note:

- One of "Lookup destination" or "Autocreate destination" is mandatory.
- If both are chosen, the component first tries to lookup the destination and if it fails it will create the destination.
- Both options are relevant only when the destination name is not empty.

4.7.7.1.6 Error Destination Configuration

The details of the destination to which the errors are sent can be specified here.

Destination Name:

The name of the destination on which the errors occurred after sending the message will be received. This is optional. If the destination name is not defined, then a temporary destination will be created whose type is same as the type of the request destination.

Destination Type:

The JMS type of the destination. This can be chosen as either Topic or Queue. This option is relevant only when the destination name is not empty.

Lookup Destination:

This option can be chosen if the destination is already present and needs to be looked up using JNDI.

Note: In case of **Active MQ**, to use look up destination, additional JNDI property [topic|queue].<look up name> has to be added as described in the following link. http://activemq.apache.org/jndi-support.html.

AutoCreate Destination:

This option has to be chosen if the destination has to be created.

Note:

- One of "Lookup destination" or "Autocreate destination" is mandatory.
- If both are chosen, the component first tries to lookup the destination and if it fails it will create the destination.
- Both options are relevant only when the destination name is not empty.

4.7.7.1.7 Consumer Configuration

The details of the consumers that has to be created for the response and error destinations can be configured here. The configuration provided is relevant only when destination name is provided either for response destination or error destination.

Message Selector: The message selector for the messages to be received by this consumer. More information on message selectors is available at http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/jms/Message.html

Is Durable: This option has to be chosen if a durable subscriber has to be created on the destination.

Note:

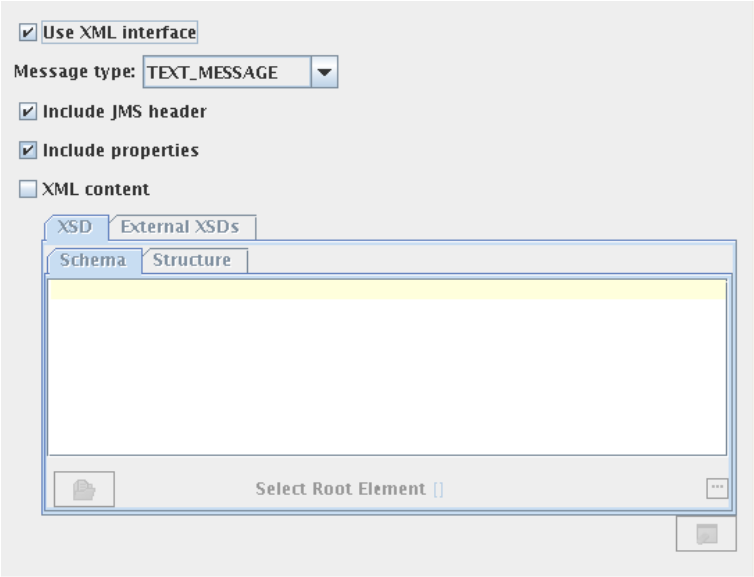
- If a durable subscriber is already created by this component and the checkbox is deselected, when the CPS is finished, User is provided with an option to remove durable subscription that is created previously. This can be done by providing the administrator credentials of the JMS Server.
- This option is relevant only when the destination type is selected as "TOPIC" in the response destination configuration.
- Selecting this option enables the user to specify the subscription name.

Subscription Name: The name of the subscription corresponding to the durable subscriber that is created by the component.

Note: This option is relevant only when the response destination type is selected as "TOPIC" in the destination configuration and the property "Is Durable" is selected.

Request Message Definition

Request Message type configuration can be defined in this panel. The schema of the input port of the component will be determined based on the configuration provided.



Use XML Interface:

This option has to be chosen if the component needs an XML interface. If this is not selected, schema will not be set on input port of the component and the message received on the input port will be sent to the destination as it is.

Message Type:

The type of the message that needs to be sent can be chosen as one of the following.

- **Text message** - Use this option to send a plain text message.
- **Map message** - A Map Message object's message body contains a set of name-value pairs, where names are String objects, and values are Java primitives. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined.
- **Stream message** - A Stream Message object's message body contains a stream of uninterrupted bytes. This message type is for literally encoding a body to match an existing message format.

Include JMS Headers:

This option can be selected to set the JMSHeaders on the message at runtime.

When this option is selected, the fields JMSCorrelationID and JMSDestination will be generated in the input schema of the component.

Include Properties:

This option can be selected to set the JMS properties on the message at runtime.

When this option is selected, the elements “ApplicationContext” and “Property” (ZeroMany) are generated in the input schema of the component.

The properties and application context can be set as shown in the figure.

```
<ns1:Properties>
   <ns1:ApplicationContext><![CDATA[<context>test</context>]]></ns1:ApplicationContext>
   <ns1:Property name="name" type="String">string</ns1:Property>
</ns1:Properties>
```

Depending on the Message Type selected, the appropriate parsing option is shown. Message Type and the corresponding parsing options are explained below.

Text message – XML Content: Select this option if the content of the input message conforms to a specific schema. The schema can be specified using the schema editor.

Map message – Define Body: Select this option to define the body fields of the Map message. The message body fields (name and type of the field) can be added in the table and the input schema will be generated corresponding to the each field defined.

Stream message – Parse Body: Select this option to set the content of the stream message in input XML. The name, type and length of the field can be added in the table. An element will be added to the schema corresponding to each field with the same name.

Response Message Definition

Use XML Interface: Select this option if the response message has to be converted to XML format before sending to the output port of the component.

Message Type:

The type of the message that has to be received can be chosen as one of the following.

- **Text message** - Use this option to receive a plain text message.
- **Map message** - A Map Message object's message body contains a set of name-value pairs, where names are String objects, and values are Java primitives. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined.
- **Stream message** - A Stream Message object's message body contains a stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format.

Include JMS Headers: This option can be selected to include JMSHeader information of message in the output XML. An element "JMSHeader" will be added to the output schema and all relevant JMS headers appear as child elements in the output schema as shown in figure.

<ns1:Message>	
<ns1:JMSHeader>	
<ns1:JMSCorrelationID>?	#string
<ns1:JMSDeliveryMode>?	#string
<ns1:JMSDestination>?	
@type	TOPIC | QUEUE
#string	
<ns1:JMSExpiration>?	#string
<ns1:JMSMessageID>?	#string
<ns1:JMSPriority>?	#string
<ns1:JMSRedelivered>?	#string
<ns1:JMSReplyTo>?	#string
<ns1:JMSTimeStamp>?	#string
<ns1:JMSType>?	#string
<ns1:Properties>	
<ns1:ApplicationContext>?	#string
<ns1:Property>*	
@name	#string

Include Properties:

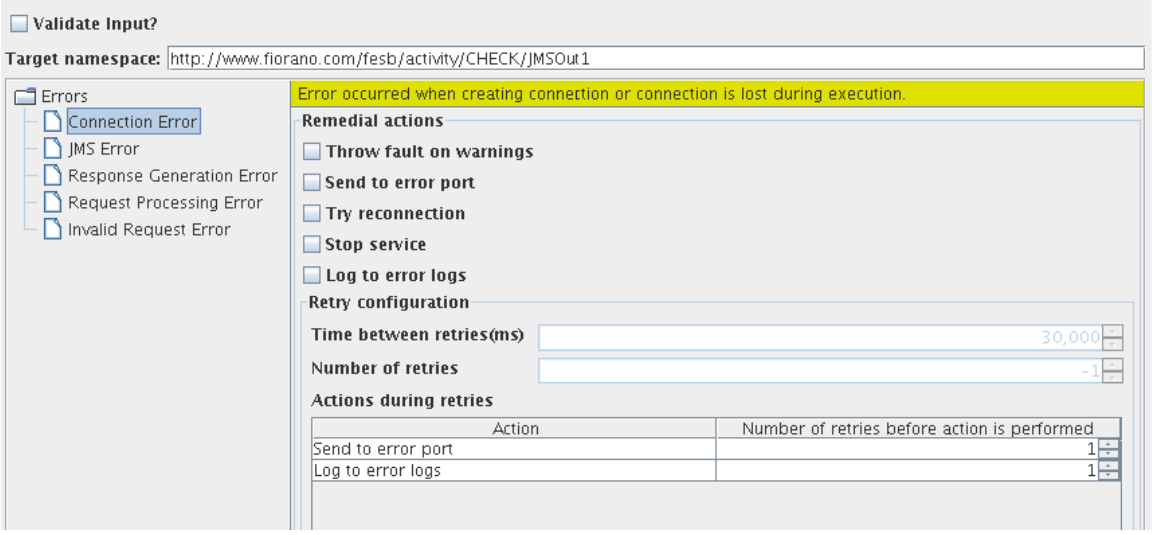
This option can be selected to retrieve the JMS properties on the message.

When this option is selected, the elements "ApplicationContext" and "Property" (ZeroMany) will be generated in the schema of output port.

Depending on the type of the message, the parsing of the message content can be chosen.

- **Text message – XML Content**: This option can be chosen if the content of the output message conforms to a specific schema. The schema can be specified using the schema editor. This schema will be set as the schema of the body element. Setting the schema enables doing necessary transformations.
- **Map message – Define Body**: This option can be chosen to retrieve the body fields of the map message. The message body fields (name and type of the field) can be added in the table and the output schema will be generated corresponding to the each field defined.
- **Stream message – Parse Body**: This option can be chosen to retrieve the content of the stream message. The name, type and length of the field can be added in the table. An element will be added to the schema corresponding to each field with the same name.

4.7.7.1.8 Additional Configuration



Validate Input: This option has to be chosen if the input message has to be validated against the schema that is set on the input port.

Target namespace: The target namespace of the schema can be provided here. The schema on ports will be suffixed with the component name and type of the port. Eg. /JMSIn/In

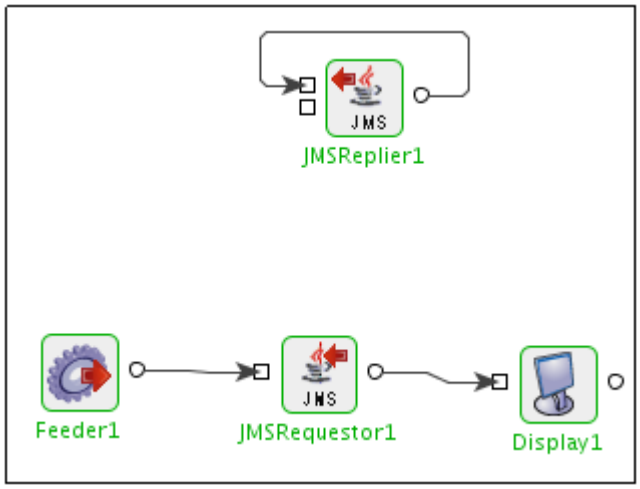
Error Handling Configuration: The actions that have to be taken in case of different kinds of errors can be chosen using this panel.

4.7.7.2 Functional Demonstration

4.7.7.2.1 Scenario 1

The component is configured to send message to "PrimaryTopic". A replier is used to resend the request that is send to it as shown in figure.

Sample Input




```
<ns1:Message xmlns:ns1="http://www.fiorano.com/fesb/activity/JMSREQUESTOR/JMSRequestor1/ReqestReply/In">
   <ns1:JMSHeader>
      <ns1:JMSCorrelationID>JMSCorrelationID</ns1:JMSCorrelationID>
   </ns1:JMSHeader>
   <ns1:Properties>
      <ns1:ApplicationContext><![CDATA[<context>test</context>]]></ns1:ApplicationContext>
      <ns1:Property name="name" type="String">string</ns1:Property>
   </ns1:Properties>
   <ns1:Body>Body</ns1:Body>
</ns1:Message>
```

Sample Output

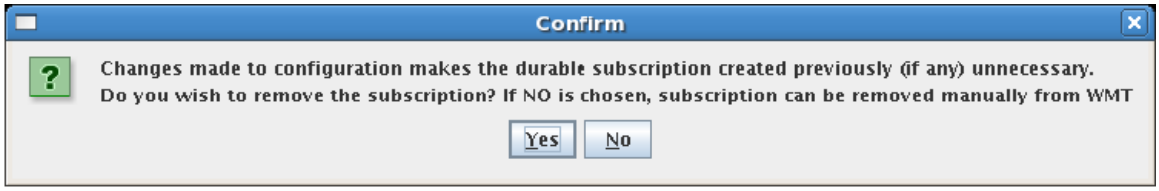
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <fns:Message xmlns:fns="http://www.fiorano.com/fesb/activity/JMSREQUESTOR/JMSRequestor1/Requestor/Out">
3     <fns:Properties>
4        <fns:Property name="ESBX__SYSTEM__PORT_NAME" type="String">IN_PORT</fns:Property>
5        <fns:Property name="ESBX__SYSTEM__COMP_INST_NAME" type="String">JMSReplier1</fns:Property>
6        <fns:Property name="name" type="String">string</fns:Property>
7        <fns:Property name="ESBX__SYSTEM__EVENT_PROCESS_NAME" type="String">JMSREQUESTOR</fns:Property>
8        <fns:Property name="RequestorID" type="String">RequestorID_1251384828672</fns:Property>
9     </fns:Properties>
10    <fns:Body>Body</fns:Body>
11 </fns:Message>
```

4.7.7.3 Useful Tips

- When using the Durable Subscriber mode, each message has to be stored by the JMS server either in the database or the file system depending on the vendor before delivery of message to consumer and removed after delivery of message. This has a huge impact on the performance. So as far as possible restrict the use of durable delivery mode unless it is necessary for your application to avoid the overheads involved.
- The durable subscription created by this component earlier will become unnecessary if the following gets changed while reconfiguring the component.
- IsDurable property is deselected in consumer configuration; Subscription name is changed.
- Response Destination Type is changed from Topic to Queue; ClientID is changed.

Therefore the previous subscription has to be removed to avoid persisting messages in runtimedata.

If the "use specific client ID" is selected in connection configuration step, a dialog **Confirm** will appear prompting the user if the durable subscription has to be removed. The user can select "yes" and then give the admin credentials to remove the previous durable subscription.



Otherwise, the subscription has to be removed manually. For more information on unsubscribing, please refer to Section 29.3 **Connecting to Web Management Tool - DurableSubscribers view** of FioranoMQ Handbook.

4.7.8 MQSeriesIn

The MQSeriesIn component provides an interface to queues on IBM WebSphere MQ 5.3 and above using MQSeries client for Java. The component sends messages that are received to queues on the MQSeries Server. The input message contains details of the message to be sent to the queue.

4.7.8.1 Configuration and Testing

4.7.8.1.1 Creating queues on IBM WebSphere MQ using WebSphere MQ explorer

Required queue should be created on IBM WebSphere MQ prior to configuring the component. This section can be skipped if the queue to which messages are sent is already created.

Steps for creating a queue on IBM Websphere MQ:

1. Start WebSphereMQ Explorer.
2. In the WebSphereMQ Explorer – Navigator pane, expand the IBM WebSphere MQ and right-click **Queue Managers** node.
3. From the pop-up menu, select **New** and click **Queue Manager...** (shown in Figure 1)

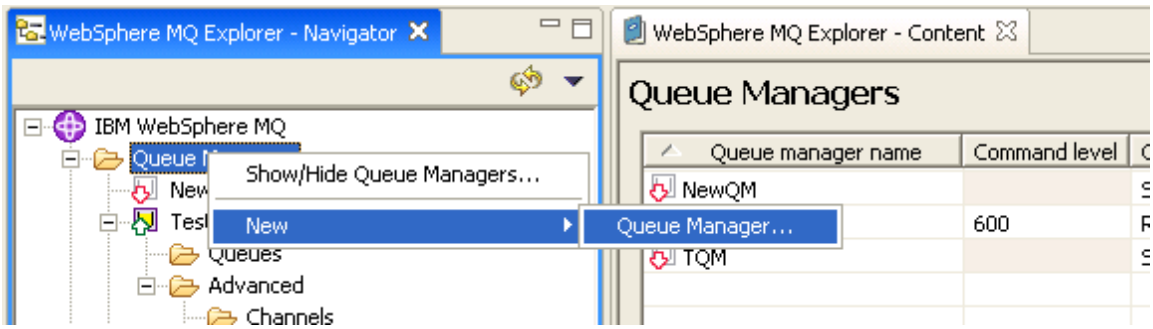


Figure 1: Adding new queue manager

4. Enter the Queue manager name with required name in Enter basic values (Step 1).

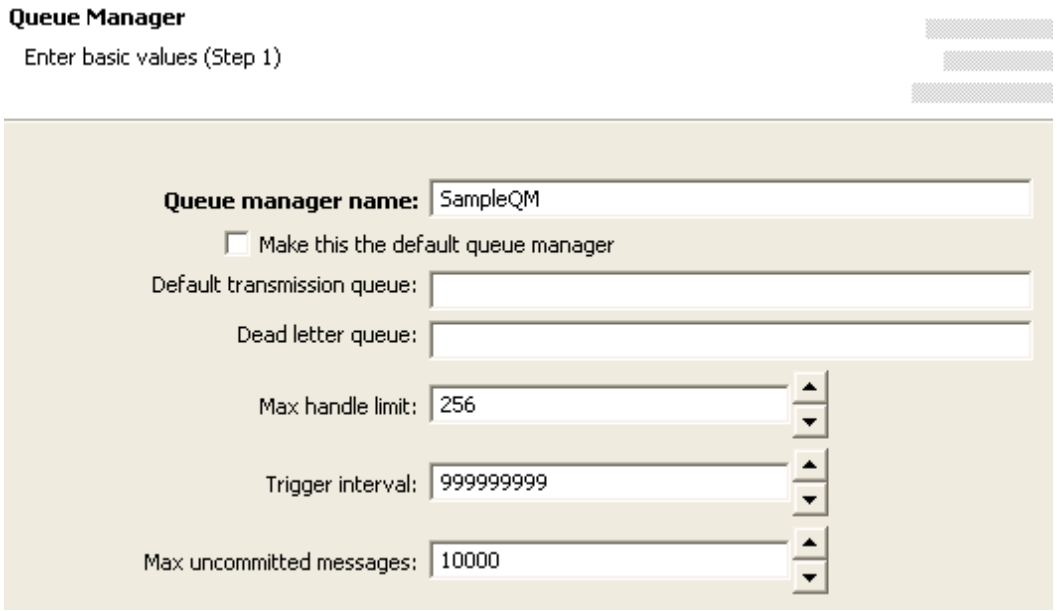


Figure 2: Providing a name for queue manager

5. Proceed to **Enter listener options (Step 4)** wizard page.
6. Provide a port number which is not used by any other application or any other Queue Manager in **IBM WebSphere MQ** as shown in Figure 3.

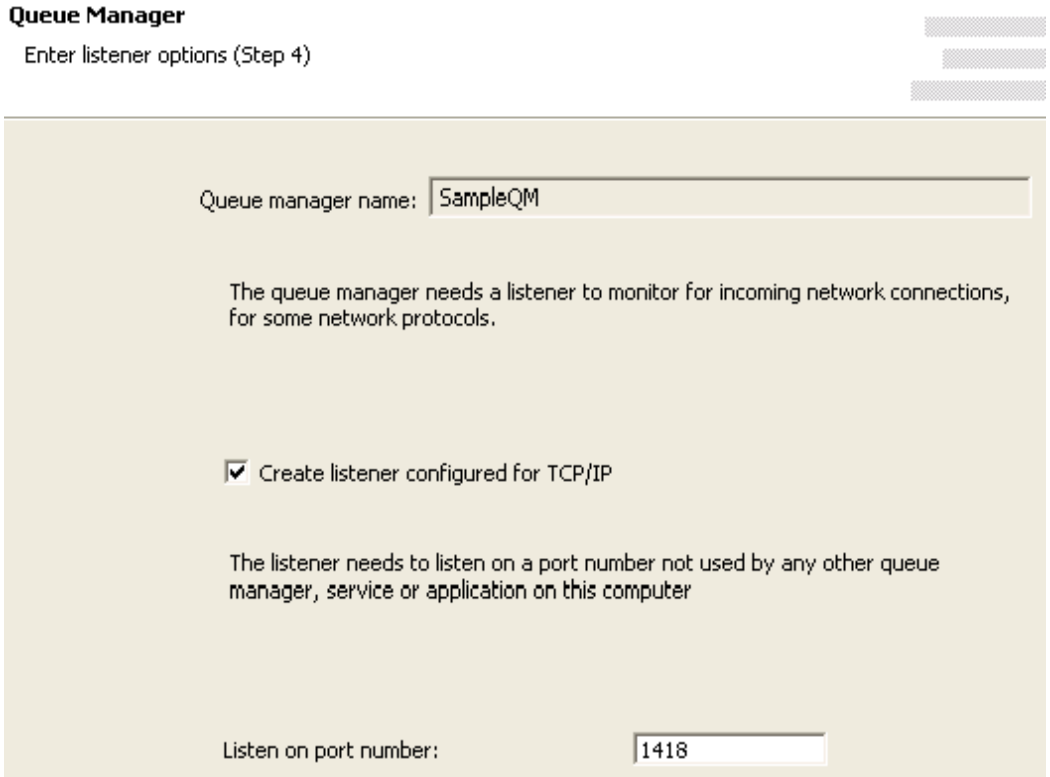


Figure 3: Providing port number

7. Click **Finish**. A new Queue Manager with given name is created and shown in **WebSphereMQ Explorer – Navigator**.

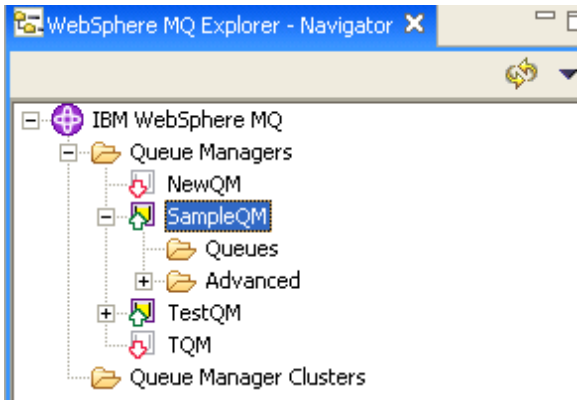


Figure 4: New Queue Manager Sample QM

8. In the **WebSphereMQ Explorer – Navigator**, expand IBM WebSphere MQ > Queue Managers > SampleQM > **Advanced** and right-click **Channels** node as shown in Figure 5.

9. From the pop-up menu, select **New** and click **Server-connection Channel** to add a server-connection channel as shown in Figure 5.

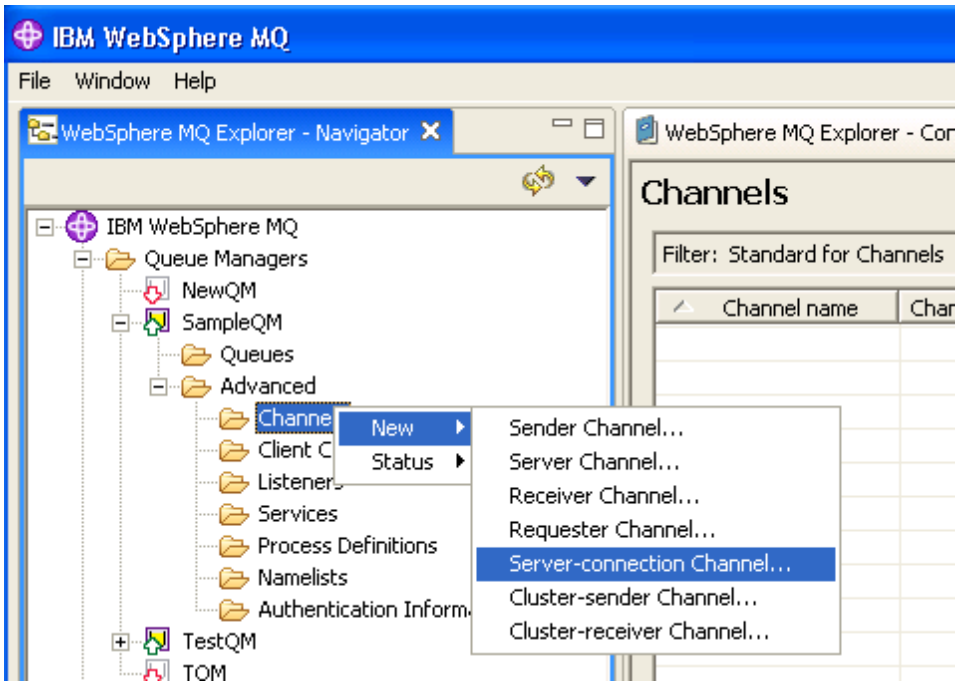


Figure 5: Adding a server-connection channel

10. Enter the **Name** in **Create a Server-connection Channel** step, as shown in Figure 6, and click **Finish**.

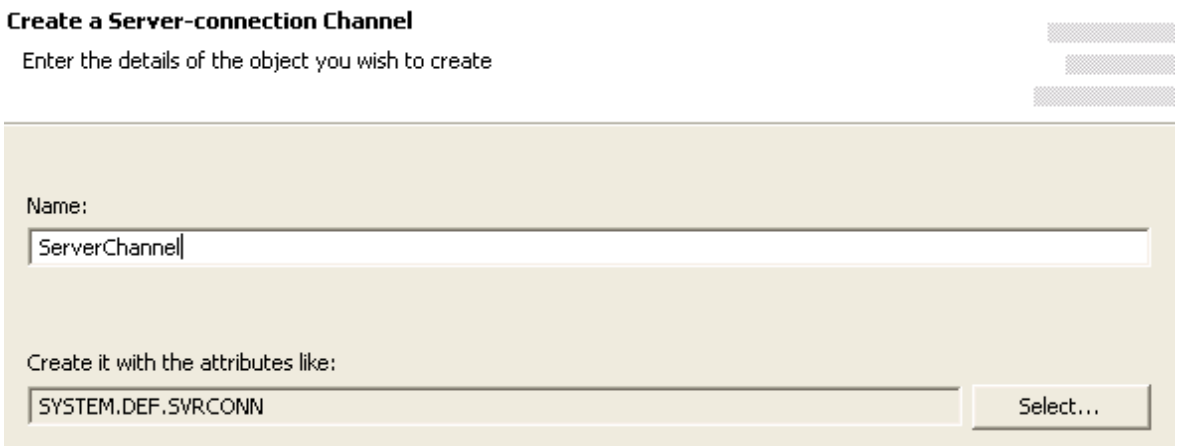


Figure 6: Configuring Server-connection Channel with required name

11. On successful completion, newly added server-connection channel is shown when **IBM WebSphere MQ** > **Queue Managers > SampleQM > Advanced > Channels** node is expanded as shown in Figure 7.

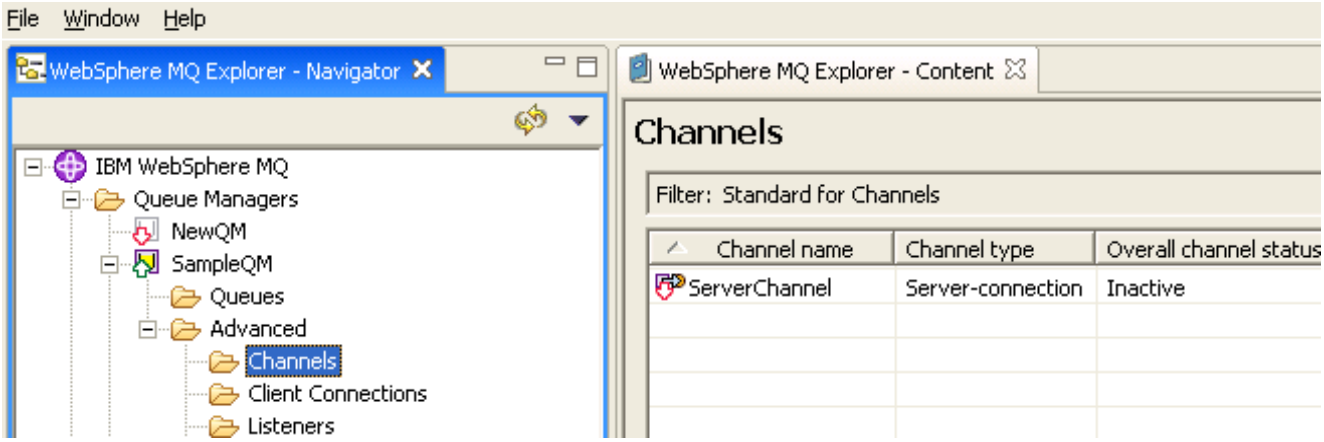


Figure 7: Newly added server connection channel

12. Right-click the newly added **Server-connection Channel** and click **Start** option from the pop-up menu as shown in Figure 8.

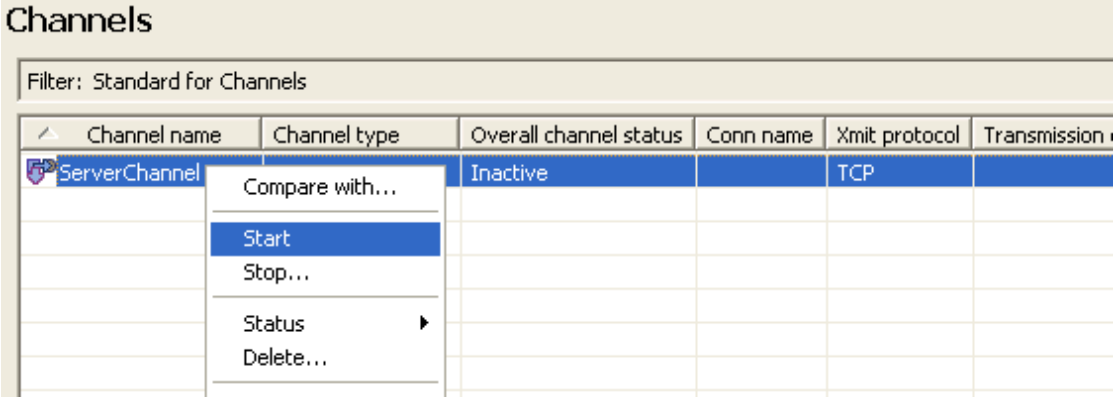


Figure 8: Starting Server-connection Channel

13. In the WebSphereMQ Explorer – Navigator, expand IBM WebSphere MQ > Queue Managers > SampleQM and right-click Queues node as shown in Figure 8.
14. From the pop-up menu, select **New** and click **Local queue** to add a local queue as shown in Figure 9.

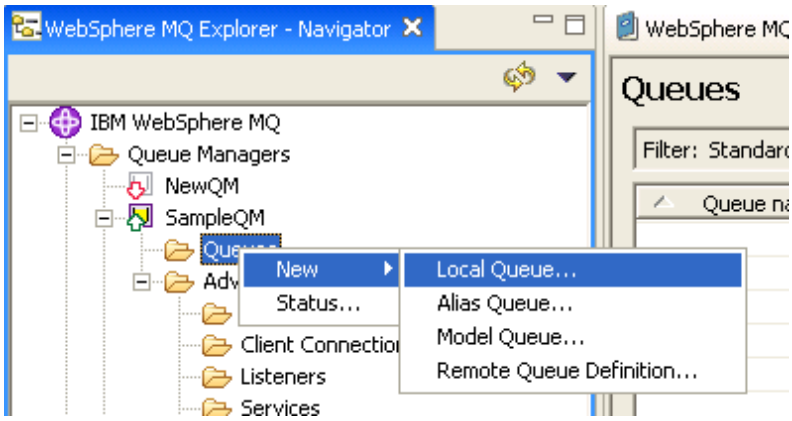


Figure 9: Adding a new local queue

15. Enter the **Name** in **Create a Local Queue** step, as shown in Figure 10, and click **Finish**.

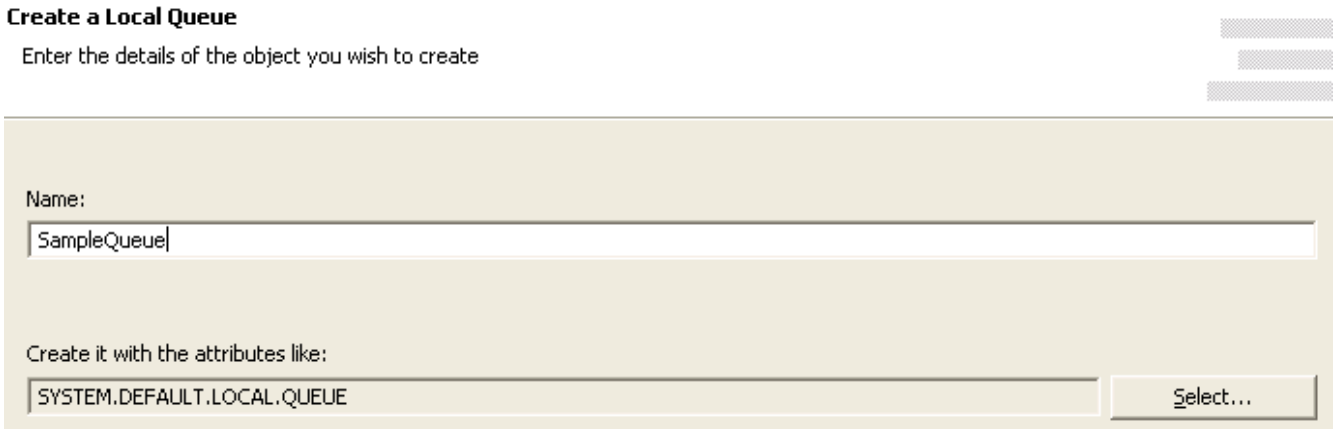


Figure 10: Name for local queue

On successful completion, newly added Local Queue is shown when **IBM WebSphere MQ** > **Queue Managers > SampleQM > Queues** node is expanded as shown in Figure 11.

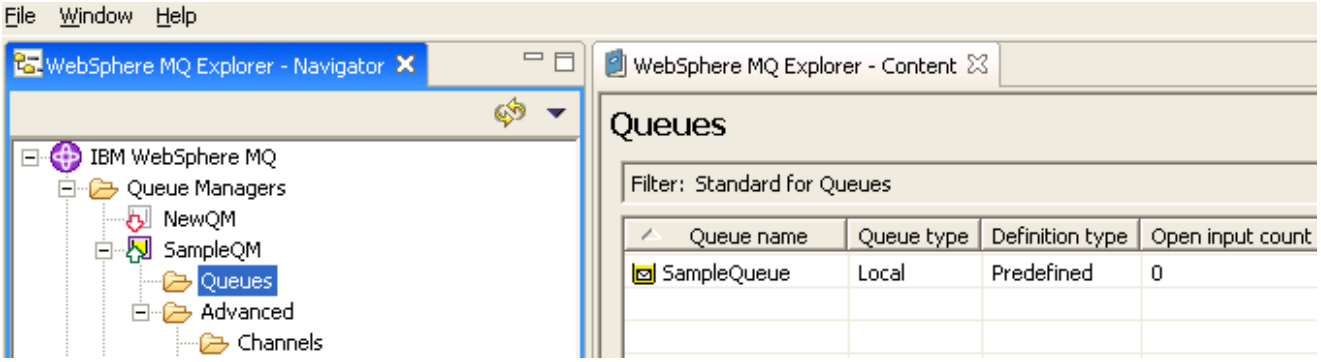


Figure 11: Newly added Local Queue

4.7.8.2 Managed Connection Factory

The connection details are configured in the first panel, **Managed Connection Factory** (**MCF**). Figure 12 illustrates the panel with expert properties view enabled.

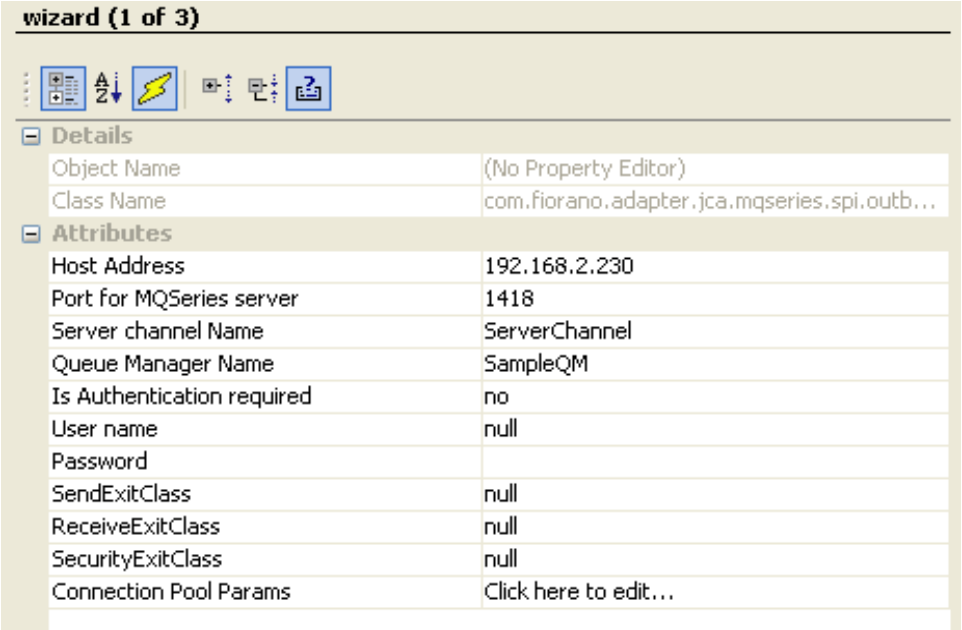


Figure 12: Connection configuration details in MCF panel

4.7.8.2.1 Attributes

4.7.8.2.1.1 Host Address

The hostname or IP address of the machine on which **IBM WebSphereMQ** Server is running. If connecting to **IBM WebSphereMQ** on the same machine on which the component is running, use localhost.

4.7.8.2.1.2 Port for MQSeries server

The port number on which IBM WebSphere MQ listens for connection requests to connect to configured Queue Manager. To view the port number for required Queue Manager (value for property **Queue Manager Name**), expand **IBM WebSphere MQ** > **Queue Managers > SampleQM > Advanced > Listeners** in **WebSphereMQ Explorer – Navigator** as shown in Figure 13.

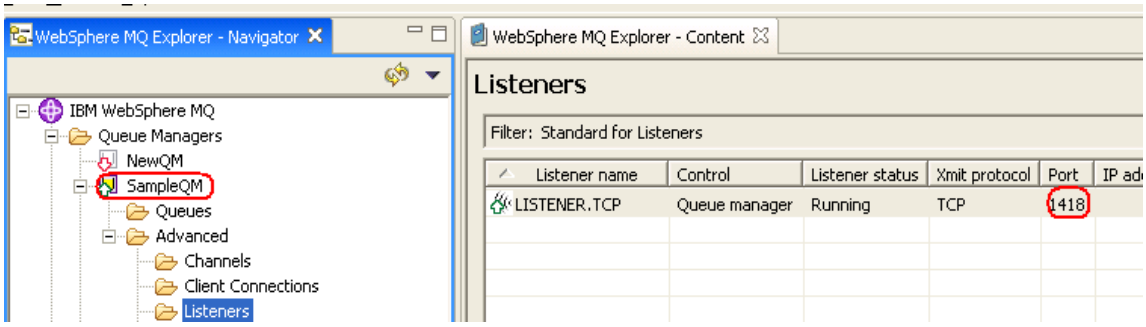


Figure 13: Port number of Queue Manager Sample QM

4.7.8.2.1.3 Server Channel Name

The name of the channel to be used for communicating with the Queue Manager. Note: The channel name is case-sensitive.

4.7.8.2.1.5 Queue Manager Name

The name of the Queue Manager in which destination queue is present.

4.7.8.2.1.6 Is Authentication required

- **yes -** Use authentication when connecting to the MQSeries Server. When this value is selected, values for properties Username and Password are used for authentication.
- **no -** Do not use authentication when connecting to the MQSeries Server. When this value is selected, values for properties Username and Password (if provided) are ignored while connecting to Queue Manager.

4.7.8.2.1.7 User name

The user name to connect to the MQSeries Queue Manager.

4.7.8.2.1.8 Password

The user password to connect to the MQSeries Queue Manager.

4.7.8.2.1.9 SendExitClass

The class that implements the MQsendExit interface. This class allows you to examine and possibly alter the data sent to the queue manager by the MQSeries client. At runtime new instance of this class is created and assigned to the variable MQEnvironment.sendExit (class in IBM MQSeries API).

4.7.8.2.1.10 ReceiveExitClass

The class that implements the MQReceiveExit interface. This class allows you to examine and possibly alter the data received from the queue manager by the MQSeries client. At runtime new instance of this class is created and assigned to the variable MQEnvironment.receiveExit.

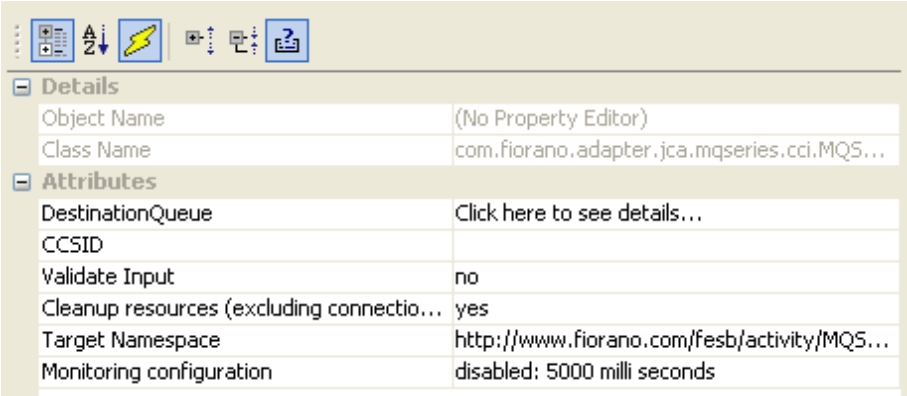
4.7.8.2.1.11 SecurityExitClass

The class that implements the MQSecurityExit interface. This class allows you to customize the security flows that occur when an attempt is made to connect to a queue manager. At runtime, new instance of this class is created and assigned to the variable MQEnvironment.securityExit.

A WebSphere MQ JMS application can use channel security, send, and receive exits on the MQI channel that starts when the application connects to a queue manager. An application connects to a queue manager by setting channel related fields or environment properties in the MQEnvironment class. Further information can be found at http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp?topic=/com.ibm.mq.csqzaw.doc/uj21370_.htm

4.7.8.3 Interaction Configuration

Business logic configuration details are configured in the second panel, **Interaction Configurations**. Figure 14 illustrates the panel with expert properties view enabled.



Details	
Object Name	(No Property Editor)
Class Name	com.fiorano.adapter.jca.mqseries.cci.MQS...
Attributes	
DestinationQueue	Click here to see details...
CCSID	
Validate Input	no
Cleanup resources (excluding connectio...	yes
Target Namespace	http://www.fiorano.com/fesb/activity/MQS...
Monitoring configuration	disabled: 5000 milli seconds

Figure 14: Business logic configuration in Interaction Configurations panel

4.7.8.3.1 Attributes

4.7.8.3.1.1 DestinationQueue

This property defines the queue on **IBM WebSphere MQ** to which messages are sent, format of message, and message sending options.

Click the ellipsis button to launch an editor for providing these configurations.

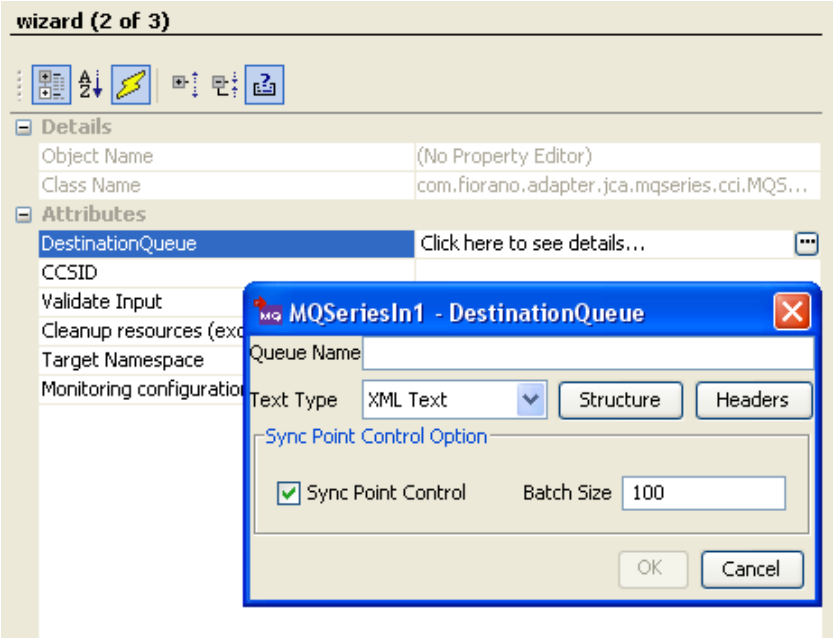


Figure 15: Launching editor for configuring queue and message sending options

- **Queue Name**

  Name of the queue on **IBM WebSphere MQ** to which messages are sent.

- **Text Type**

  A MQ message contains message descriptor (a set of headers that define the message) and data. The data is stored in binary format. Multiple fields are written in a sequence using a type specific API. The data in the message is read by the consuming application exactly in the same order as the data is written.

  The MQSeriesIn component takes in an input message in text format (either XML or raw), builds a MQ message internally from the message in text format based on message details configured in CPS and sends the message to the configured queue.

  There are two formats for defining the input message:

  - **XML Text**

    This option defines a XML structure with headers and fields of message body defined in CPS. The input XML contains values for headers and message body fields, which are parsed and respective values, are set on MQ message. The MQ message will then be sent to MQ queue.

    This option should be used in any of the following cases:

    - When the message body contains data for multiple fields. Example: Name and age of a person MQMD or RFH2 headers and/or their values vary with each input message.
    - When this option is selected, both **Structure** and **Headers** buttons are enabled.

  - **Raw Text**

This option does not define any structure for input message. The input message is set on the message body as a single string field. Headers (MQMD and RFH2) with values defined in CPS are sent on every message. Only one RFH2Header is added to the message.

This option should be used when the all of the followings conditions are met:

- When the message body contains only a single text field and complex transformation and XML parsing can be avoided. A typical case, when a message is pulled from one system and passed on without any modifications to IBM WebSphere MQ series. MQMD or RFH2 headers and their values are constant for all messages.
- When this option is selected, **Structure** button is disabled and **Headers** button is enabled.

Refer to section **Input and Output** for details about the effects of these configurations on input and output structures.

- **Structure**

  The structure button is enabled only when **Text Type** is XML Text. Click this button to launch editor to define the fields of the message. Refer Figure 16.

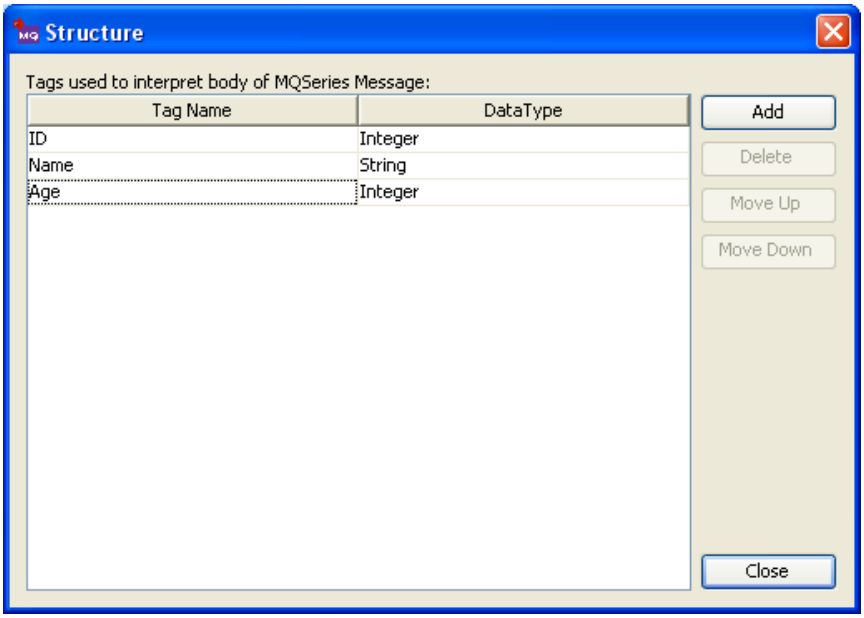


Figure 16: Editor to define fields in message

The editor contains a table with **Tag Name** and **Data Type** columns. Each row corresponds to a field in the MQ message. The value in **Tag Name** column is the element name for the element in XML structure which holds data for the field. The value in **Data Type** column is the type of the data that is held in the field. It can only contain following values (case-sensitive) - Char, Double, Float, Integer, Long Integer, Short Integer, UTF, String, Boolean, Byte Array. The Figure 16 shows the definition of structure to build a MQ message with data for a ID, name and age of a person.

The order of the fields is important and the order of the rows in this table defines the order of fields in the MQ message. The **Move Up** and **Move Down** buttons are used to reorder the fields. The **Add** and **Delete** buttons are used to add or remove the fields.

The schema for input XML is built based on the fields defined here. However, the types for elements in schema do not use the types defined here. All the elements are defined as string type and hence, the schema does not validate content of element with appropriate data type.

Note:

- Data in input XML for Boolean type fields should contain true or false (not case sensitive). Any other value is treated as false.
- Data in input XML for Byte Array type fields should contain Base64 encoded string. The **Base64-Encode** function under **Advanced Functions** in **Fiorano Mapper** can be used to convert byte array into Base64 encoded string.
- Data in input XML for Char type fields should contain only one character. If there is more than one character, only the first character is picked.
- If the order of elements in input XML differs from the order defined or if there are additional elements or if some of the elements are missing, one of the following happens:
    - error when building MQ message
    - mixed up values between fields
    - junk data is set on MQ message

  Refer to section Input and Output for details about the effects of these configurations on input and output structures.

- **Headers**

  Click **Headers** button to launch an editor, refer to Figure 17, that defines MQ message descriptor (MQMD) fields and RFH2 Headers to be used. This property depends on **Text Type.**

Defining headers when Text Type is XML Text

When **Text Type** is set to XML Text the editor defines the following:

- MQMD headers whose values have to be provided in input message.
- RFH2 headers, if required, and their default values.

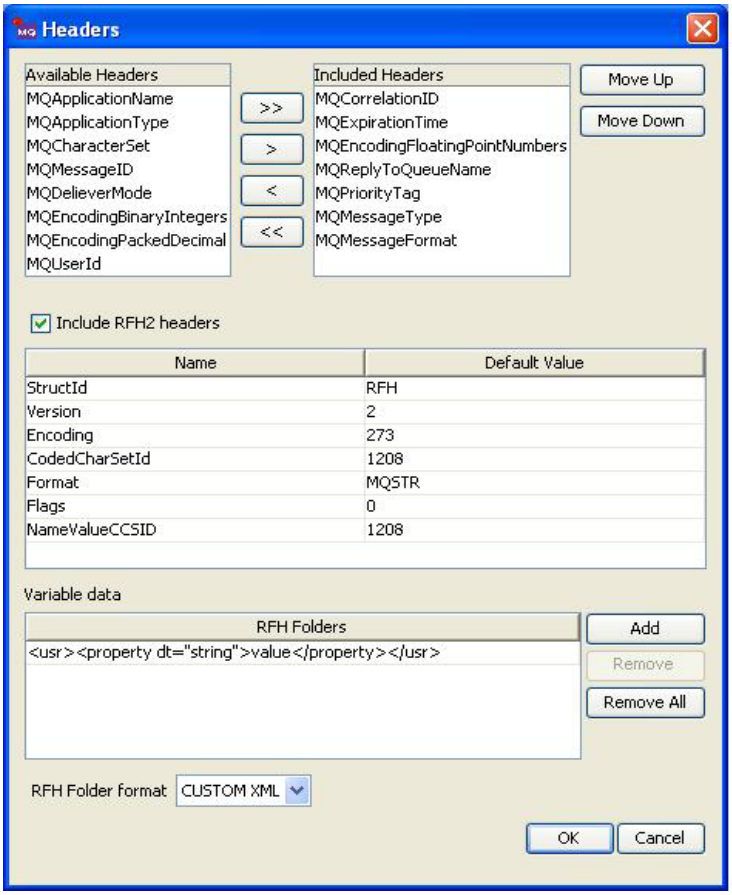


Figure 17: Headers editor when Text Type is XML Text

Defining MQMD headers

Available Headers: Contains MQMD headers that can be defined on MQ message and whose values are not taken from input message. Headers in this list are set with default values on MQ message.

Included Headers: Contains MQMD headers that can be defined on MQ message and whose values are taken from input message.

Headers in **Available Headers** and **Included Headers** together contain all MQMD headers.

MQMD headers can be selected or unselected as

- To populate values for MQMD headers from input message, select required headers from **Available Headers** list and click [>].
- To populate values for all MQMD headers from input message, click [>>].
- To use default values for MQMD headers, select required headers from **Included Headers** list and click [<].
- To use default values for all MQMD headers, click [<<].

The default values are used for MQMD headers which are in **Available Headers** list or which are in **Included Headers,** but corresponding elements are not present in input XML.

Refer to section **MQMD headers** for default values

Defining RFH2 headers

Select the **Include RFH2 headers** check box to set RFH2 headers on MQ Message, refer to Figure 17. If the check box **Include RFH2 headers** is unchecked the input schema created will not contain elements related to RFH2 headers.

Multiple RFH2 headers can be set in the input message. Each RFH2 header should contain all the fields shown in Figure 17. If any RFH2 header in input XML does not contain the particular field then corresponding value from the **Default Value** column is used. The **Variable data** is also a field of RFH2 header and is treated similar to other RFH2 fields in the table.

Refer to section **RFH2 header fields** for default values.

Defining headers when Text Type is Raw Text

When **Text Type** is set to Raw Text the editor defines the following:

- MQMD headers and their values that have to be set on MQ message.
- RFH2 headers, if required and their values that has to be set on MQ message.

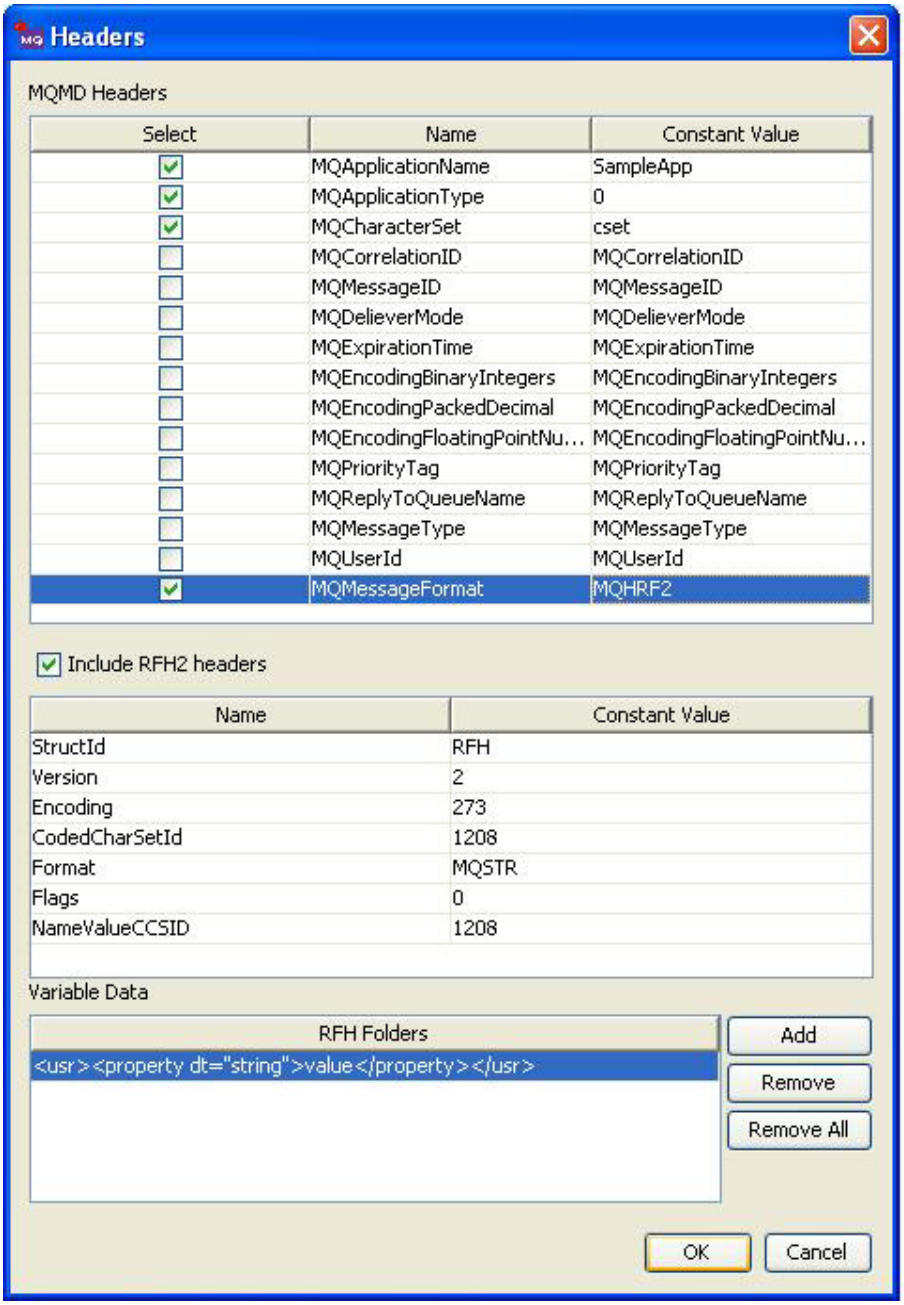


Figure 18: Selection of MQBD headers that has to be set on the input message

Defining MQMD headers

Select the check box in **Select** column if a particular MQMD header value in **Name** column has to be set on the input message with value specified in **Constant Value** column, refer to Figure 18. If the check box in **Select** column is not checked, then the default value for corresponding MQMD header is used. The values provided in Constant Value column are not validated against the data type of the corresponding MQMD header and hence, should be carefully defined.

Note: MQMD headers are set with same values on all messages.

Refer to section **MQMD headers** for default values and data types.

Defining RFH2 headers

Select the **Include RFH2 headers** check box to set RFH2 header on MQ Message, refer to Figure 18. If the check box **Include RFH2 headers** is unchecked RFH2 header is not set on the message.

Only one RFH2 header can be set on message. Values for each field of RFH2 header can be provided in **Constant Value** column against corresponding field as shown in Figure 18. The **Variable data** is also a field of RFH2 header and is treated similar to other RFH2 fields in the table. By default, when this editor is opened for the first time default values for all fields are loaded.

Refer to section **RFH2 header fields** for additional information.

MQMD Headers

Only some of most commonly used MQMD headers are provided in the CPS of the component. Refer to Table 1 for a short description and default value used for each MQMD header. Refer to link http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp?topic=/com.ibm.mq.csqzak.doc/mqmd.htm for detailed information on MQMD headers.

Note: Values in **Default value** column of Table 1 are the defaults defined in the component and may vary from the actual default values of **IBM WebSphere MQ**.

Table 1 Short descriptions, default values and data types of MQMD headers used in the component

MQMD header name	**Description**	**Default value**	**Data type**
MQApplicationName	Name of application that put the message.	Empty string (“”)	String
MQApplicationType	Type of application that put the message	0 (MQAT_NO_CONTEXT)	Integer
MQCharacterSet	Character set identifier of character data in the message	0 (MQCCSI_Q_MGR)	Integer
MQCorrelationID	A byte string that the application can use to relate one message to another, or to relate the message to other work that the application is performing	null (MQCI_NONE)	byte array as a hex string

MQMD header name	Description	Default value	Data type
MQMessageID	A byte string that is used to distinguish one message from another. The message identifier is a permanent property of the message, and persists across restarts of the queue manager.	null (MQMI_NONE)	byte array as a hex string
MQDeliveryMode	Delivery mode indicating whether the message survives system failures and restarts of the queue manager.	0 (MQPER_NOT_PERSISTENT)	Integer
MQExpirationTime	A period of time expressed in tenths of a second, set by the application that puts the message. The message becomes eligible to be discarded if it has not been removed from the destination queue before this period of time elapses.	-1 (MQEI_UNLIMITED)	Integer
MQEncodingBinaryIntegers	Subfield of encoding header that specifies encoding for binary integers	1 (MQENC_INTEGER_NORMAL)	Integer
MQEncodingPackedDecimal	Subfield of encoding header that specifies encoding for packed-decimal integers	16 (MQENC_DECIMAL_NORMAL)	Integer
MQEncodingFloatPointNumbers	Subfield of encoding header that specifies encoding for floating-point integers	256 (MQENC_FLOAT_IEEE_NORMAL)	Integer
MQPriorityTag	Priority of the Message	-1 (MQPRI_PRIORITY_AS_Q_DEF)	Integer
MQReplyToQueueName	Name of the message queue to which the application that issued the get request for the message sends MQMT_REPLY and MQMT_REPORT messages	Empty string (“”)	String
MQMessageType	Type of message	8 (MQMT_DATAGRAM)	Integer

MQMD header name	Description	Default value	Data type
MQUserId	User identifier of the application that originated the message. The queue manager treats this information as character data, but does not define the format of it.	Empty string (“”)	String
MQMessageFormat	A name that the sender of the message uses to indicate to the receiver the nature of the data in the message	null	String

RFH2 Header fields

Refer to Table 2 for a short description and default value used for each RFH2. Refer to link http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp?topic=/com.ibm.mq.csqzak.doc/csqzak10172.htm for detailed information on RFH2 header fields.

Note: Values in **Default value** column of Table 2 are the defaults defined in the component and may vary from the actual default values of **IBM WebSphere MQ.**

Table 2 Short descriptions, default values and data types of RFH2 header fields used in the component

RFH2 Header field	Description	Default value	Data type
StructId	Structure identifier. This field should strictly contain 4 characters. If there are more than 4 characters content after 8th character is pruned by the component. If there are less than 8 characters required additional blank spaces are padded at the end by the component.	RFH (MQRFH_STRUC_ID)	String of length 4 characters
Version	Structure version number	2 (MQRFH_VERSION_2)	Integer
Encoding	The numeric encoding of the data that follows the last NameValueData field; it does not apply to numeric data in the MQRFH2 structure itself	273 (MQENC_NATIVE)	Integer

RFH2 Header field	Description	Default value	Data type
CodedCharSetId	The character set identifier of the data that follows the last NameValueData field; it does not apply to character data in the MQRFH2 structure itself.	1208	Integer
Format	The format name of the data that follows the last NameValueData field. This field should strictly contain 8 characters. If there are more than 8 characters content after 8th character is pruned by the component. If there are less than 8 characters required additional blank spaces are padded at the end by the component.	MQSTR (MQFMT_STRING)	String of length 8 characters
Flags	This value should be set to MQRFH_NONE	0 (MQRFH_NO_FLAGS)	Integer
NameValueCCSID	The coded character set identifier of the data in the NameValueData field.	1208	Integer
Variable Data	A variable-length character string containing data encoded using an XML-like syntax. Refer to link http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp?topic=/com.ibm.mq.csqzak.doc/js07180.htm for additional details about this field	null	String

Variable Data

The variable portion contains a variable number of MQRFH2 folders. Each folder contains a variable number of elements or properties. Folders group together related properties. Each MQRFH2 folder can be added as an entry in the table as shown in the figure 18 using the add button. Refer to link http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp?topic=/com.ibm.mq.csqzak.doc/js07180.htm for additional details about this field

RFHFolder format

When the input type is XML, the way MQRFH2 folders can be specified in the input message depends on this property.

- TEXT

When this option is chosen the RFHFolders can be specified as CDATA. For example a sample jms Folder **<jms><Dst>queue:///Test</Dst><Dlv>4</Dlv><Tms>3543534</Tms></jms>** can be specified as <![CDATA[<jms><Dst>queue:///Test</Dst><Dlv>4</Dlv><Tms>3543534</Tms></jms>]]>

- CUSTOM XML

  When this opton is chosen the RFHFolders can be specified as parseable elements in the input schema. If the property value is empty then an attribute isNull="true" must be added to corresponding Property element. For example a sample jms Folder **<jms><Dst>queue:///Test</Dst><Dlv>4</Dlv><Tms>3543534</Tms></jms>** can be specified as

```
<ns1:Folder type="jms">
<ns1:Property name="Dst">queue:///Test</ns1:Property>
<ns1:Property name="Dlv">4</ns1:Property>
<ns1:Property name="Tms">3543534</ns1:Property>
</ns1:Folder>
```

Sync Point Control

The **IBM WebSphere MQ series** support transactions similar to JMS transaction. Select the check box if the messages sent to MQ queue have to be committed after sending a batch of messages. The messages are not visible or available for consumption on queue until a commit is performed. If the check box is unchecked, any messages sent are immediately available for consumption on the queue.

The **Batch Size** is taken into account only if this check box is checked.

- **Batch Size**

  The number of messages after which a commit should be performed, if the check box **Sync Point Control** is checked. The Batch size is counted based on the number of messages that are successfully sent to MQ Queue. If any message could not be sent to MQ Queue due to an error that message is not counted, but the count continues.

4.7.8.3.1.2 CCSID

The Coded Character Set Identification should be an integer or null, in case of null, 819 (ISO 8859-1 ASCII) is used as CCSID. This character set is set on the MQMessage. This overrides the MQCharacterSet header property set in MQMD headers.

4.7.8.3.2 Using Named Configurations

Changes are done in eStudio w.r.t to Named Configurations. With the implementation of Named Configurations MCF panel is changed as shown in Figure below.

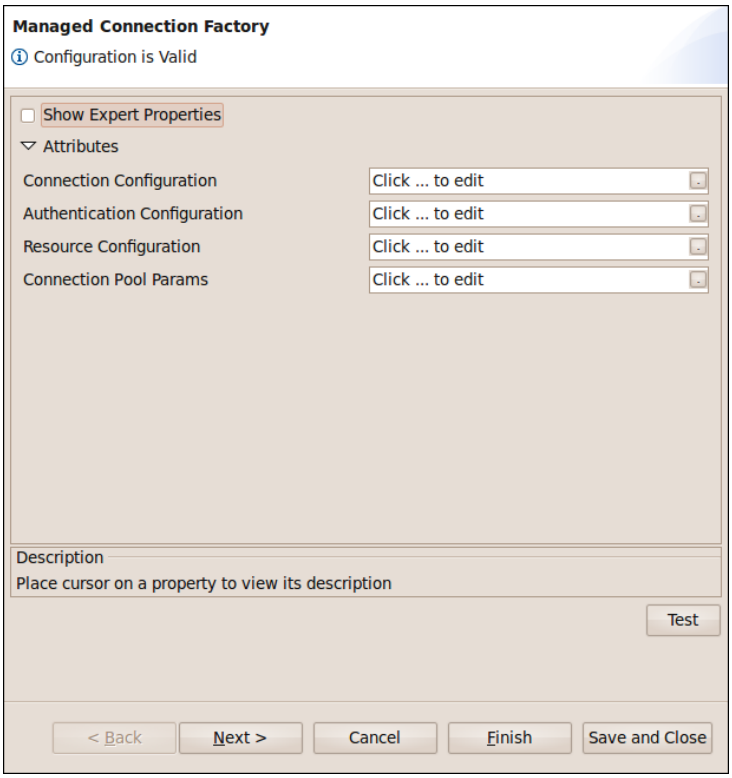


Figure: Modified Managed Connection Factory

Some properties are grouped as one property in the modified configuration panel. Host Address, Port for MQSeries server, Server channel Name and Queue Manager Name which were present previously in the MCF panel are grouped under the name Connection Configuration.

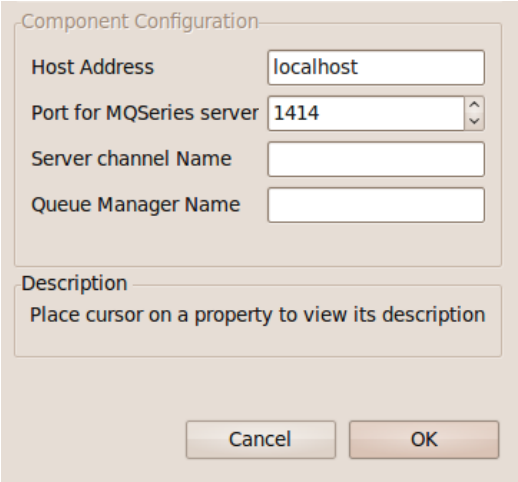


Figure: Connection Configuration

UserName and Password required for Authentication which were present previously in the MCF panel are grouped under the name Authentication Configuration.

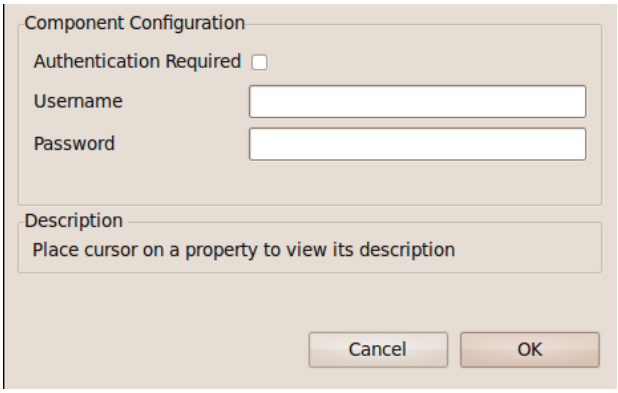


Figure: Authentication Configuration

SendExit Class, ReserveExit Class and SecurityExit Class which were present previously in the MCF panel are grouped under the name Resource Configuration.

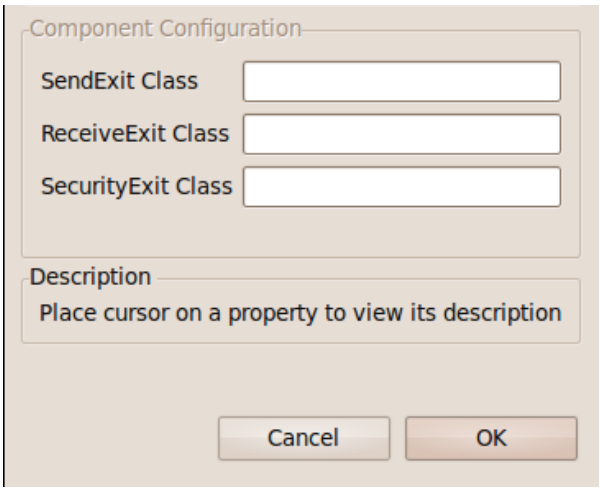


Figure: Resource Configuration

Similarly Destination Queue Panel is also modified to use Named Configurations.

4.7.8.4 Input and Output

4.7.8.4.1 Input

The input schema for the component is defined based on the configuration selected.

- When the **Text Type** is set to Raw Text there is no schema defined for the input. The input message text is written MQ message body as a single String field. Selected MQMD headers with values defined in CPS and unselected MQMD headers with default values mentioned in section **MQMD Headers** are set on MQ Message. If RFH2 header is checked, one RFH2 header with field values defined in CPS is added. The MQMD headers and RFH2 header added is same for all messages.

- When the **Text Type** is set to XML Text the input schema varies based on the configuration of structure and headers.
- When only fields of message body are defined in **Structure** dialog as shown in Figure 19.

Tags used to interpret body of MQSeries Message:

Tag Name	DataType
Name1	String
Name2	String

Figure 19: Fields defined for message body

The input schema is defined as shown in Figure 20 and a sample input is shown in Figure 21.

<ns1:MQMessage>	
<ns1:MessageBody>?	
[0,1]	
<ns1:Name1>	#string
<ns1:Name2>	#string

Figure 20: Schema when only fields of message body are defined

```
<ns1:MQMessage xmlns:ns1="http://www.fiorano.com/fesb/activity/MQSeriesInlIn/In">
    <ns1:MessageBody>
        <ns1:Name1>Name1</ns1:Name1>
        <ns1:Name2>Name2</ns1:Name2>
    </ns1:MessageBody>
</ns1:MQMessage>
```

Figure 21: Sample input XML for schema in Figure 20

All the fields defined are added as child elements under **MessageBody** in the same order in which these fields are defined in CPS. The Input XML should have all the fields and should strictly follow the order. MQMD headers are not explicitly set by the component.

- When fields of message body are defined in **Structure** dialog as shown in Figure 19 and MQMD headers are defined in **Headers** dialog as shown in Figure 22.

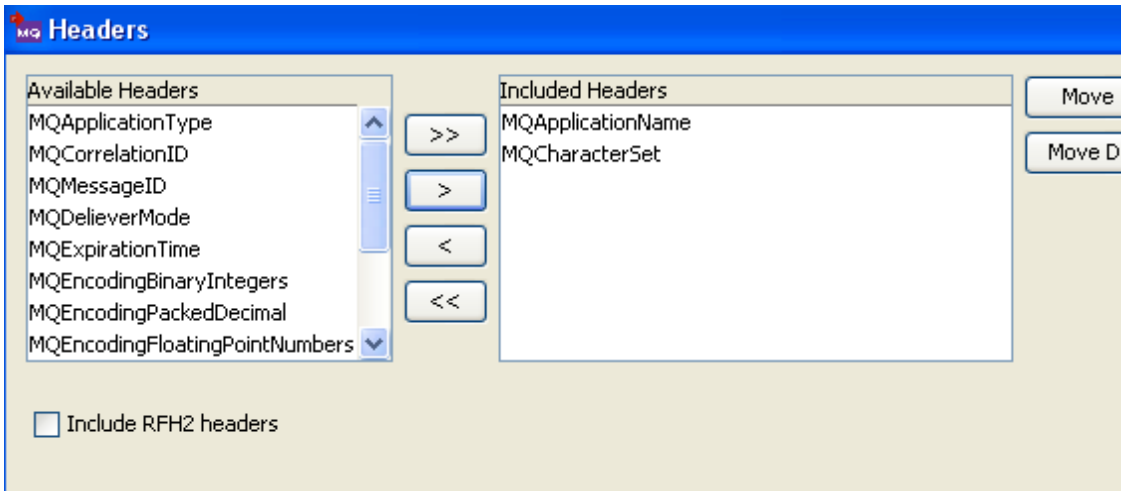


Figure 22: Selecting MQMD headers whose values have to be taken from input XML

The input schema is defined as shown in Figure 23 and a sample input is shown in Figure 24.

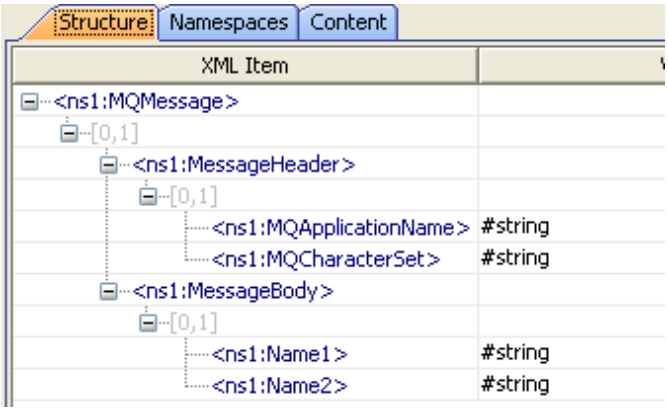


Figure 23: Input schema when fields of message body and some MQMD headers are defined

```
<nsl:MQMessage xmlns:nsl="http://www.fiorano.com/fesb/activity/MQSeriesInlIn/In">
   <nsl:MessageHeader>
      <nsl:MQApplicationName>MQApplicationName</nsl:MQApplicationName>
      <nsl:MQCharacterSet>MQCharacterSet</nsl:MQCharacterSet>
   </nsl:MessageHeader>
   <nsl:MessageBody>
      <nsl:Name1>Name1</nsl:Name1>
      <nsl:Name2>Name2</nsl:Name2>
   </nsl:MessageBody>
</nsl:MQMessage>
```

Figure 24: Sample input XML for schema defined in Figure 23

All the fields defined are added as child elements under **MessageBody** in the same order in which these fields are defined in CPS and all the MQMD headers selected are added as child elements under **Message Header** in the same order in which these headers are defined in CPS.

If **Message Header** element is not present MQMD headers are not explicitly set by the component and it behaves exactly like the case where only the fields of message body are defined.

If **Message Header** element is present, then for the MQMD fields that are present under the **Message Header** element in input XML, values are taken from input XML. For the MQMD fields that are not present under the **Message Header** element, default values as described in section **MQMD Headers** are used irrespective of whether the header is selected or not in CPS as shown in Figure 22.

- When fields of message body are defined in **Structure** dialog as shown in figure 19 and both MQMD headers and RFH2 header are defined in **Headers** dialog as shown in Figure 25.

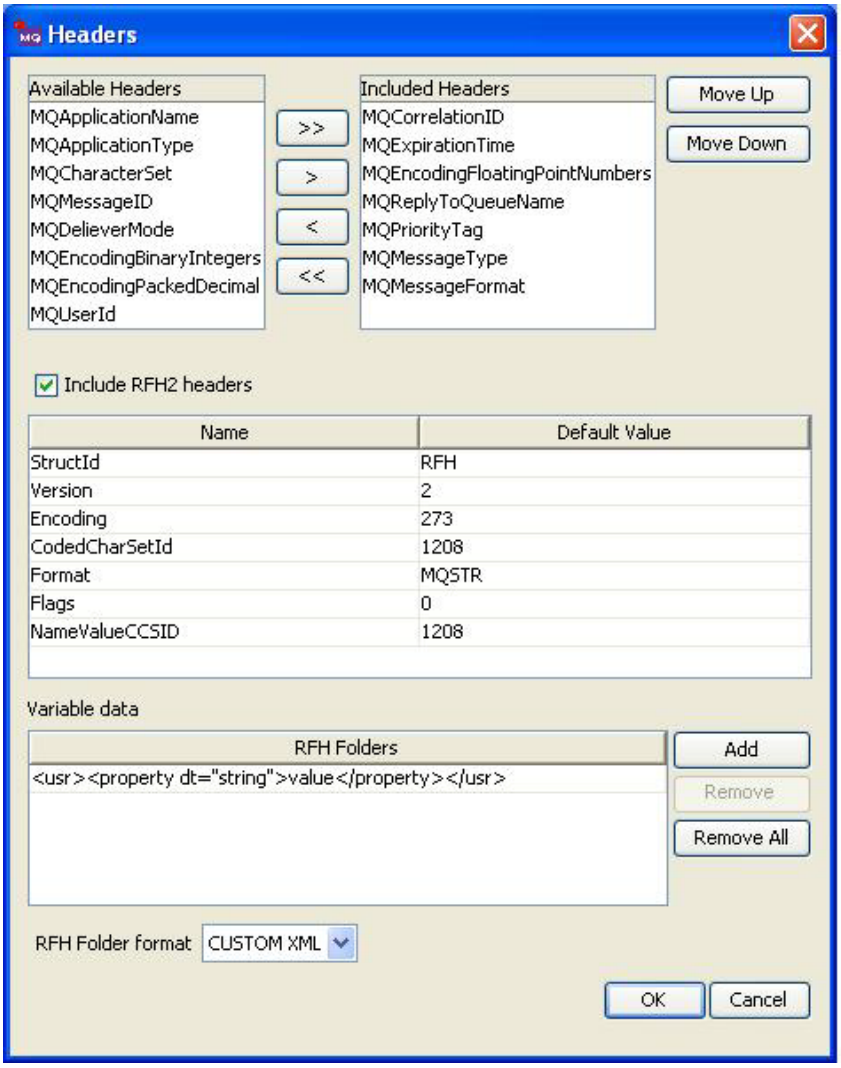


Figure 25: Selecting some MQMD headers and enable RFH2 header

The input schema is defined as shown in Figure 26 and a sample input is shown in Figure 27.

Structure | Namespaces | Content

XML Item	Value Type
<ns1:MQMessage>	
[0,1]	
<ns1:MessageHeader>	
[0,1]	
<ns1:MQCorrelationID>	#string
<ns1:MQExpirationTime>	#string
<ns1:MQEncodingFloatingPointNumbers>	#string
<ns1:MQReplyToQueueName>	#string
<ns1:MQPriorityTag>	#string
<ns1:MQMessageType>	#string
<ns1:MQMessageFormat>	#string
<ns1:RFH2Headers>?	
<ns1:RFH2Header>+	
[0,1]	
<ns1:StructId>	#string
<ns1:Version>	#string
<ns1:Encoding>	#string
<ns1:CodedCharSetId>	#string
<ns1:Format>	#string
<ns1:Flags>	#string
<ns1:NameValueCCSID>	#string
<ns1:RFH2Folders>?	
<ns1:Folder>*	
<ns1:Property>*	
@name	#string
@dt	#string
@isNull	#string
#string	
@type	#string
<ns1:MessageBody>	
[0,1]	
<ns1:Name1>	#string
<ns1:Name2>	#string

Figure 26: Schema when RFH2 headers are selected as well

```
<ns1:MQMessage xmlns:ns1="http://www.fiorano.com/fesb/activity/MQSeriesInlIn/In">
   <ns1:MessageHeader>
      <ns1:MQCorrelationID>MQCorrelationID</ns1:MQCorrelationID>
      <ns1:MQExpirationTime>MQExpirationTime</ns1:MQExpirationTime>
      <ns1:MQEncodingFloatingPointNumbers>MQEncodingFloatingPointNumbers</ns1:MQEncodingFloatingPointNumbers>
      <ns1:MQReplyToQueueName>MQReplyToQueueName</ns1:MQReplyToQueueName>
      <ns1:MQPriorityTag>MQPriorityTag</ns1:MQPriorityTag>
      <ns1:MQMessageType>MQMessageType</ns1:MQMessageType>
      <ns1:MQMessageFormat>MQMessageFormat</ns1:MQMessageFormat>
      <ns1:RFH2Headers>
         <ns1:RFH2Header>
            <ns1:StructId>StructId</ns1:StructId>
            <ns1:Version>Version</ns1:Version>
            <ns1:Encoding>Encoding</ns1:Encoding>
            <ns1:CodedCharSetId>CodedCharSetId</ns1:CodedCharSetId>
            <ns1:Format>Format</ns1:Format>
            <ns1:Flags>Flags</ns1:Flags>
            <ns1:NameValueCCSID>NameValueCCSID</ns1:NameValueCCSID>
            <ns1:RFH2Folders>
               <ns1:Folder type="type">
                  <ns1:Property name="name" dt="dt" isNull="isNull">string</ns1:Property>
               </ns1:Folder>
            </ns1:RFH2Folders>
         </ns1:RFH2Header>
      </ns1:RFH2Headers>
   </ns1:MessageHeader>
   <ns1:MessageBody>
      <ns1:Name1>Name1</ns1:Name1>
      <ns1:Name2>Name2</ns1:Name2>
   </ns1:MessageBody>
</ns1:MQMessage>
```

Figure 27: Sample input XML for schema defined in Figure 26

All the fields defined are added as child elements under **MessageBody** in the same order in which these fields are defined in CPS, and all the MQMD headers selected are added as child elements under **Message Header** in the same order in which these headers are defined in CPS. The RFH2 headers are added as well under **Message Header** element as **RFH2Headers** element, and the **RFH2Headers** element can contain multiple **RFH2Header** elements.

If **Message Header** element is not present, MQMD headers and RFH2 header are not explicitly set by the component and it behaves exactly like the case where only the fields of message body are defined.

If **Message Header** element is present, then for the MQMD fields that are present under the **Message Header** element in input XML, values are taken from input XML. For the MQMD fields that are not present under the **Message Header** element, default values as described in section **MQMD Headers** are used irrespective of whether the header is selected or not in CPS as shown in Figure 25. If **RFH2Headers** element is not present or if **RFH2Headers** element is present but does not have any child **RFH2Header** elements then RFH2 headers are not set. If there is a **RFH2Headers** element which contains one or more **RFH2Header** elements as children, then one RFH2 header is added for each of the **RFH2Header** elements present in the same order. A RFH2 header is populated with field values from the values present at elements with corresponding field names. If a particular field element is not present, as a child to RFH2 header then the default value defined in the CPS for that field is taken.

If the input message contains a property with name TargetQueueName and the value of property is not null message, the message is sent to the queue with name specified by the property. If the property is not defined or a value is not set for the property the message is sent to queue with name configured in CPS.

4.7.8.4.2 Output

The output schema for the component is fixed. The schema is shown in Figure 28.

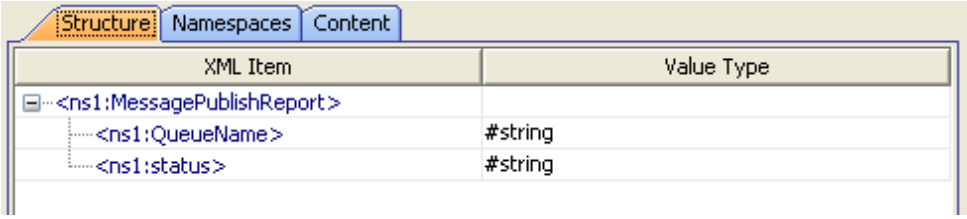


Figure 28: Schema for output message

The output message contains only two elements:

- **QueueName** – The name of queue to which the message is sent.
- **status** – success

The queue name to which the message is sent is also set as message property with name TargetQueueName.

4.7.8.5 Functional Demonstration

4.7.8.5.1 Scenario 1

Configure the component for Text type mode as Raw Text and configure **Headers** as shown in Figure 29.

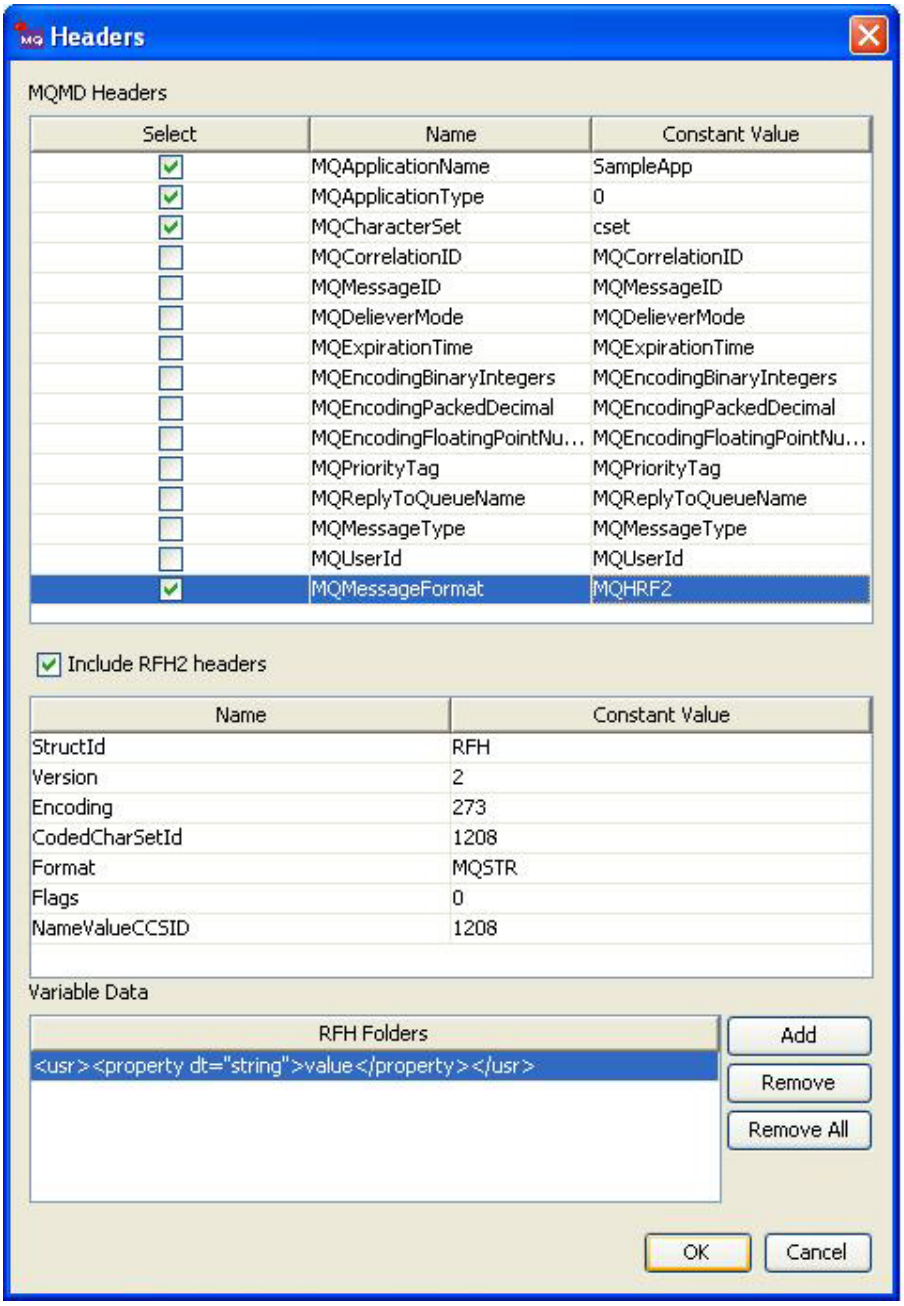


Figure 29: Configuration of headers for scenario 1

Input Message

```
testMessage
```

Output Message

```
<?xml version="1.0" encoding="UTF-8"?>
<MessagePublishReport xmlns="http://www.fiorano.com/fesb/activity/MQSeriesIn1In/Out">
   <QueueName>TestQueue</QueueName>
   <status>success</status>
</MessagePublishReport>
```

Note: MQMD headers that are selected in Figure 19 are set on MQMessage with values provided in **Constant Value** column. For the MQMD headers which are not selected in CPS, default values are set on MQ Message.

4.7.8.5.2 Scenario 2

Configure the component for XML type message.

Add two fields Name1 and Name2 with type String in **Structure** editor as shown in Figure 30.

Tags used to interpret body of MQSeries Message:

Tag Name	DataType
Name1	String
Name2	String

Figure 30: Fields in message body for scenario 2

Select MQMD headers MQCharacterSet and MQApplicationName and check RFH2 Headers as shown in Figure 31

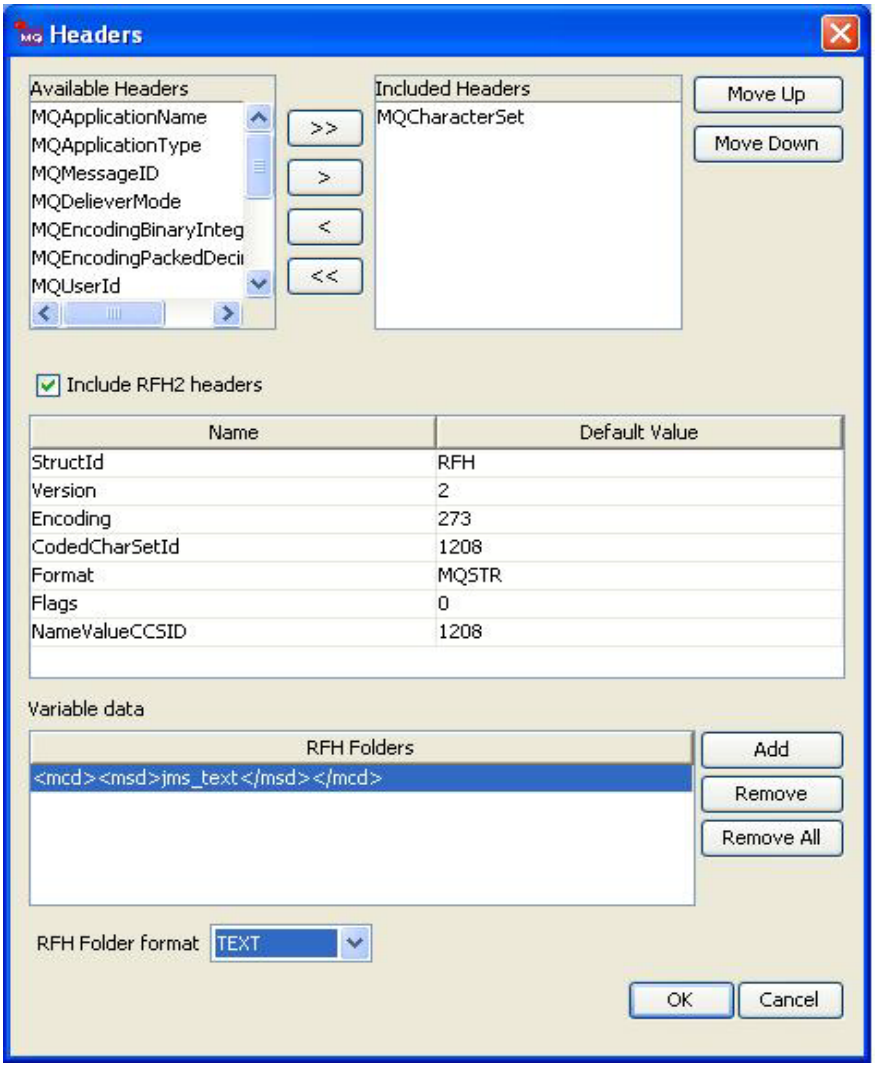


Figure 31: Headers defined for scenario 2

InputMessage:

```
<ns1:MQMessage xmlns:ns1="http://www.fiorano.com/fesb/activity/MQSeriesIn1In/In">
   <ns1:MessageHeader>
      <ns1:MQCharacterSet>1208</ns1:MQCharacterSet>
      <ns1:RFH2Headers>
         <ns1:RFH2Header>
            <ns1:StructId>RFH</ns1:StructId>
            <ns1:Version>2</ns1:Version>
            <ns1:Encoding>273</ns1:Encoding>
            <ns1:CodedCharSetId>1208</ns1:CodedCharSetId>
            <ns1:Format>MQSTR</ns1:Format>
            <ns1:Flags>0</ns1:Flags>
            <ns1:NameValueCCSID>1208</ns1:NameValueCCSID>
            <ns1:VariableData><![CDATA[<mcd>jms_text</mcd>]]></ns1:VariableData>
         </ns1:RFH2Header>
      </ns1:RFH2Headers>
   </ns1:MessageHeader>
```

```
   <ns1:MessageBody>
      <ns1:Name1>Name1value</ns1:Name1>
      <ns1:Name2>Name2value</ns1:Name2>
   </ns1:MessageBody>
</ns1:MQMessage>
OutputMessage:
<?xml version="1.0" encoding="UTF-8"?>
<MessagePublishReport xmlns="http://www.fiorano.com/fesb/activity/MQSeriesIn1In/Out">
   <QueueName>TestQueue</QueueName>
   <status>success</status>
</MessagePublishReport>
```

Note: In the above sample, input message only not all MQMD headers which are selected in CPS are provided. For those MQMD headers which are not provided in input message, default values are set. And for RFH2 headers, default values are taken from the CPS if they are not provided in input message.

4.7.8.5.3 Scenario 3

Configure the component for XML type message.

The RFH Format is chosen as "CUSTOM XML" instead of "TEXT" in figure 31. The input and output will be as below.

InputMessage:

```
<ns1:MQMessage xmlns:ns1="http://www.fiorano.com/fesb/activity/MQSeriesIn1In/In">
   <ns1:MessageHeader>
      <ns1:MQCharacterSet>1208</ns1:MQCharacterSet>
      <ns1:RFH2Headers>
         <ns1:RFH2Header>
            <ns1:StructId>RFH</ns1:StructId>
            <ns1:Version>2</ns1:Version>
            <ns1:Encoding>273</ns1:Encoding>
            <ns1:CodedCharSetId>1208</ns1:CodedCharSetId>
            <ns1:Format>MQSTR</ns1:Format>
            <ns1:Flags>0</ns1:Flags>
            <ns1:NameValueCCSID>1208</ns1:NameValueCCSID>
            <ns1:RFH2Folders>
              <ns1:Folder type="mcd">
                   <ns1:Property name="msd">jms_text</ns1:Property>
            </ns1:Folder>
            </ns1:RFH2Folders>
         </ns1:RFH2Header>
      </ns1:RFH2Headers>
   </ns1:MessageHeader>
   <ns1:MessageBody>
      <ns1:Name1>Name1value</ns1:Name1>
      <ns1:Name2>Name2value</ns1:Name2>
   </ns1:MessageBody>
</ns1:MQMessage>
OutputMessage:
<?xml version="1.0" encoding="UTF-8"?>
```

```
<MessagePublishReport xmlns="http://www.fiorano.com/fesb/activity/MQSeriesIn1In/Out">
   <QueueName>TestQueue</QueueName>
   <status>success</status>
</MessagePublishReport>
```

Note: In the above sample, input message does not have all MQMD headers which are selected in CPS. For such MQMD headers default values are set. For RFH2 headers, default values are taken from the CPS if they are not provided in input message.

4.7.8.6 Useful Tips

The correct CCSID should be set for message encoding when transferring messages from AS 400 systems to other platforms and vice versa.

4.7.9 MQSeriesOut

The MQSeriesOut component provides an interface to queues on IBM WebSphere MQ 5.3 and above, using MQSeries client for Java. The MQSeriesOut component receives messages from queues on MQSeries Queue Manager.

4.7.9.1 Configuration and Testing

4.7.9.1.1 Creating Queues on IBM WebSphere MQ using WebSphere MQ Explorer

Required queue should be created on **IBM WebSphere MQ** prior to configuring the component. This section can be ignored if the queue from which messages are received is already created.

Steps for creating a queue on **IBM Websphere MQ**

- Start **WebSphereMQ Explorer**.
- In the **WebSphereMQ Explorer – Navigator**, expand **IBM WebSphere MQ,** right-click **Queue Managers** node.
- From the pop-up menu point to **New** and click **Queue Manager...** (shown in Figure 1)

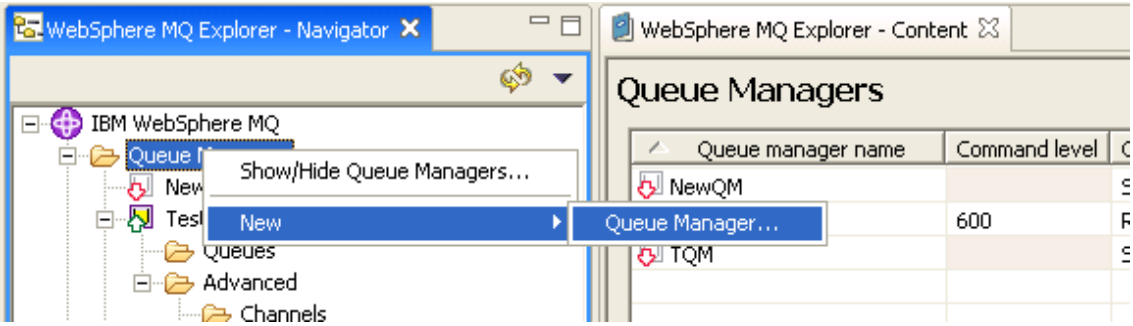


Figure 1: Adding new queue manager

- Enter the **Queue manager name** with required name in **Enter basic values (step 1)**.

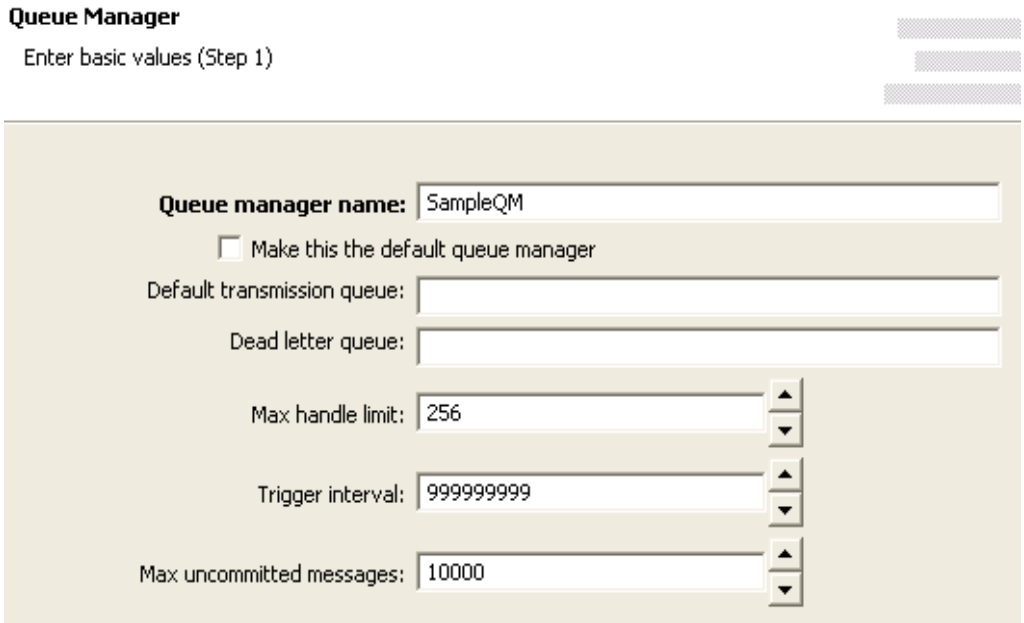


Figure 2: Providing a name for queue manager

- Proceed to **Enter listener options (Step 4)** of the wizard.
- Provide a port number which is not used by any other application or any other Queue Manager in **IBM WebSphere MQ** as shown in Figure 3.

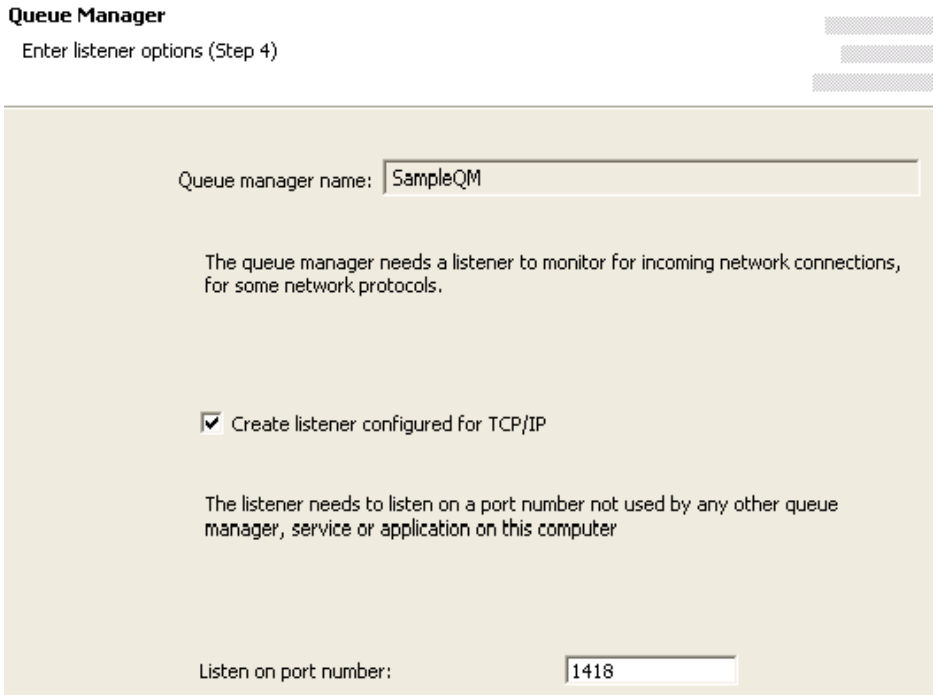


Figure 3: Providing port number

- Click the **Finish** button.

- A new Queue Manager with given name is created and shown in **WebSphereMQ Explorer – Navigator**.

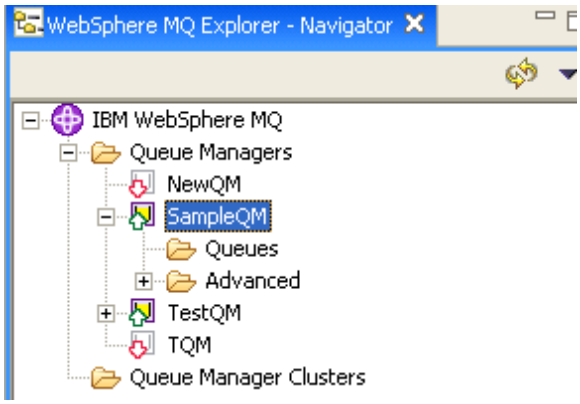


Figure 4: New Queue Manager Sample QM

- In the **WebSphereMQ Explorer – Navigator**, expand **IBM WebSphere MQ** > **Queue Managers > SampleQM > Advanced** and right-click **Channels** node as shown in Figure 5.

- From the pop-up menu, point to **New** and click **Server-connection Channel** to add a server-connection channel as shown in Figure 5

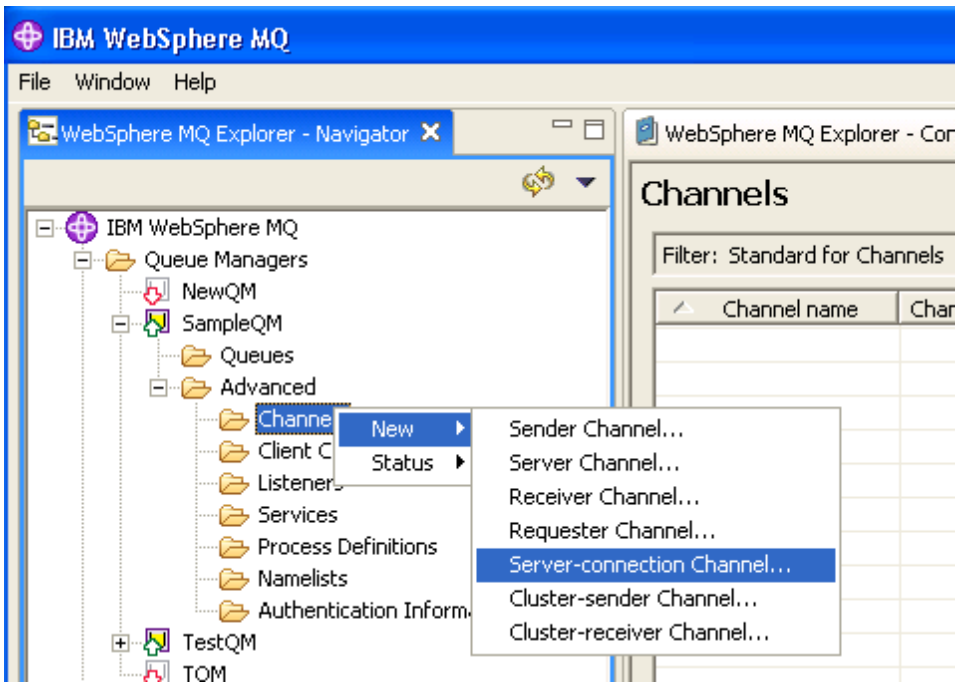


Figure 5: Adding a server-connection channel

- Enter the **Name** in **Create a Server-connection Channel** step, as shown in Figure 6, and click **Finish**.

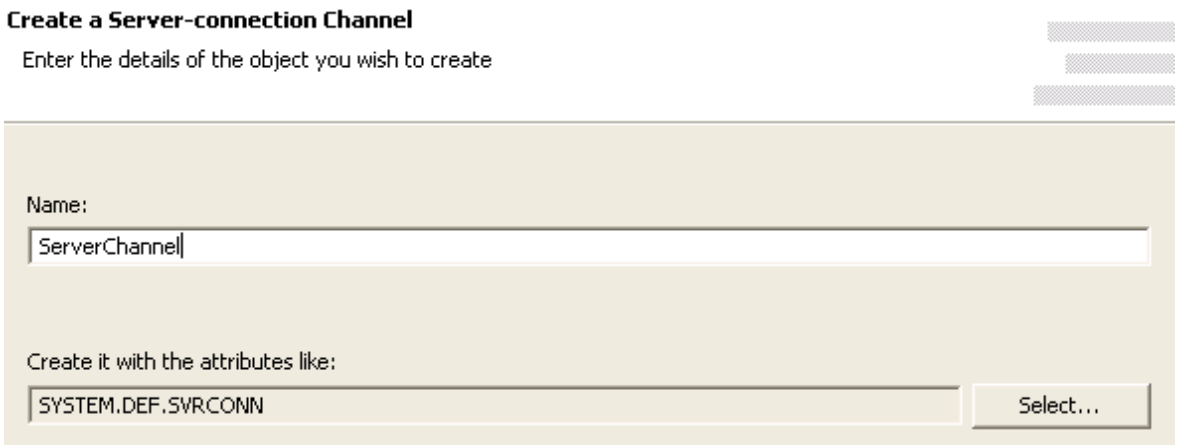


Figure 6: Configuring Server-connection Channel with required name

- On successful completion, newly added server-connection channel is shown when **IBM WebSphere MQ** > **Queue Managers > SampleQM > Advanced > Channels** node is expanded as shown in Figure 7.

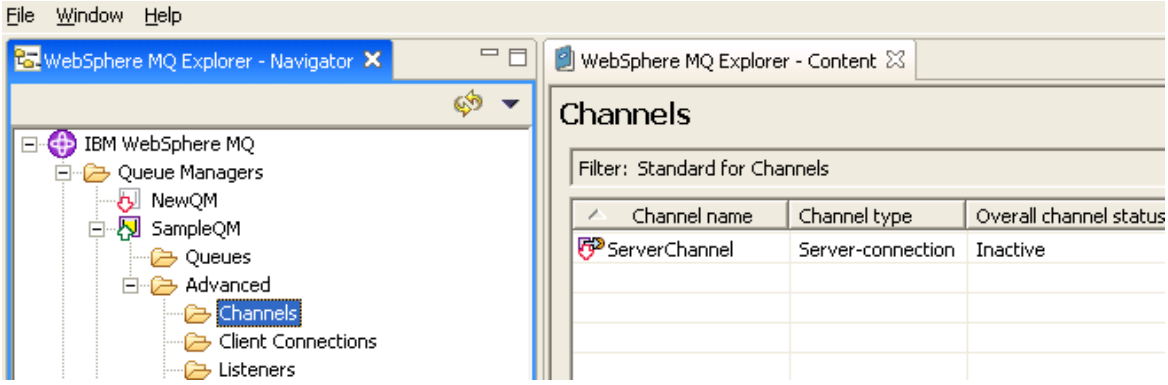


Figure 7: Newly added server connection channel

- Right-click the newly added **Server-connection Channel** and click Start option from the pop-up menu as shown in Figure 8.

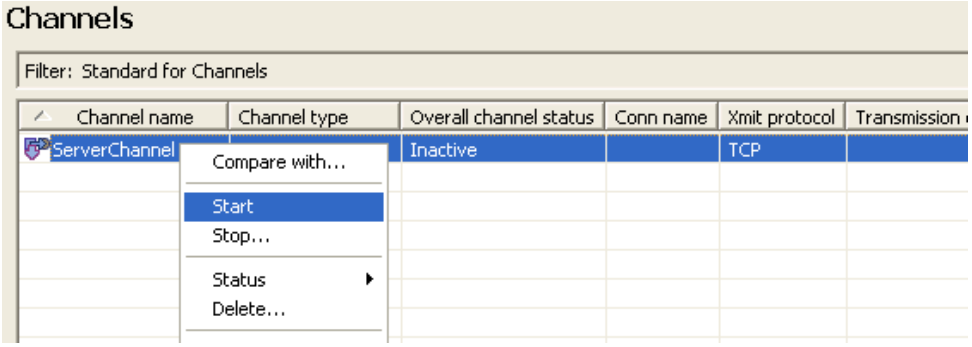


Figure 8: Starting Server-connection Channel

- In the **WebSphereMQ Explorer – Navigator**, expand **IBM WebSphere MQ** > **Queue Managers > SampleQM** and right-click **Queues** node as shown in Figure 8.
- From the pop-up menu, point to **New** and click **Local queue** to add a local queue as shown in Figure 9

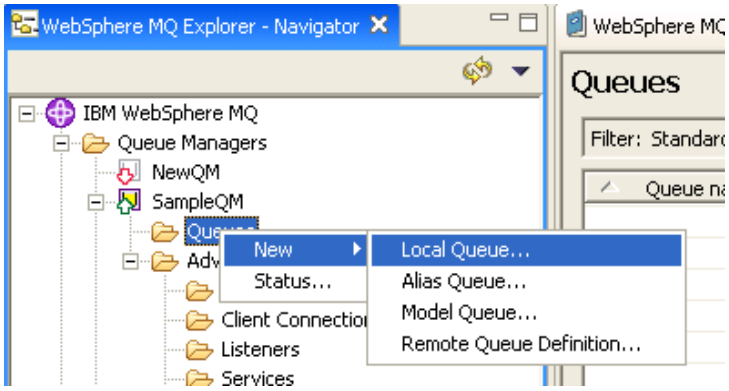


Figure 9: Adding a new local queue

- Enter the **Name** in **Create a Local Queue** step, as shown in Figure 10, and click **Finish**.

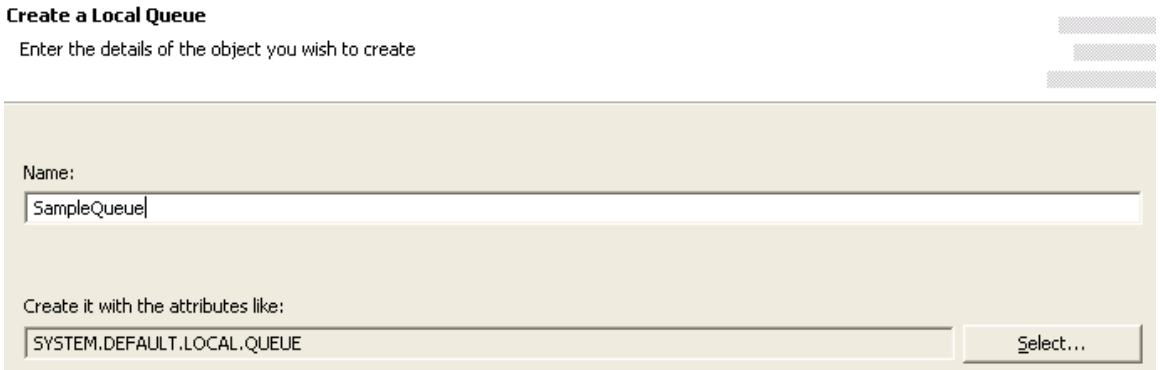


Figure 10: Name for local queue

- On successful completion, newly added Local Queue is shown when **IBM WebSphere MQ** > **Queue Managers > SampleQM > Queues** node is expanded as shown in Figure 11.

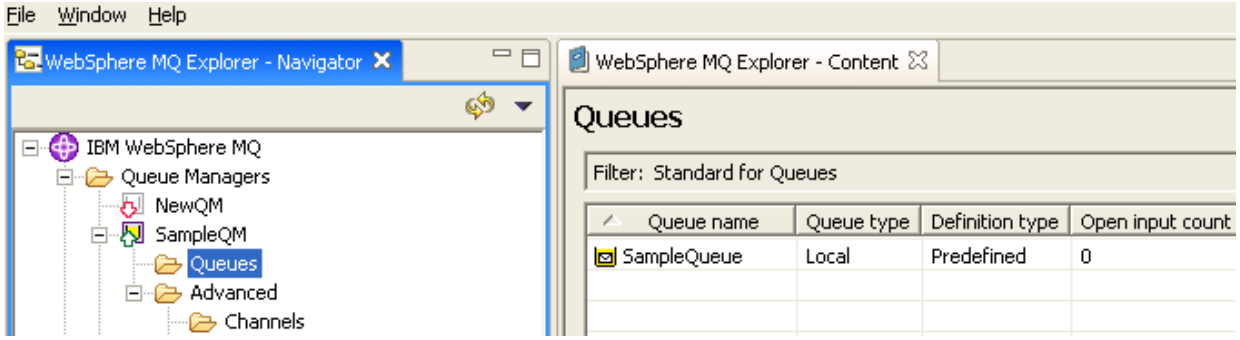


Figure 11: Newly added Local Queue

Note: To put a test message in the Queue from the wizard, expand SampleQM -> Queues, right-click on SampleQueue and select **Put Test Message** and then enter some message in the Queue.

4.7.9.2 Managed Connection Factory

The Connection details are configured in the first panel, **Managed Connection Factory** (**MCF**). The Figure 12 illustrates the panel with expert properties view enabled.

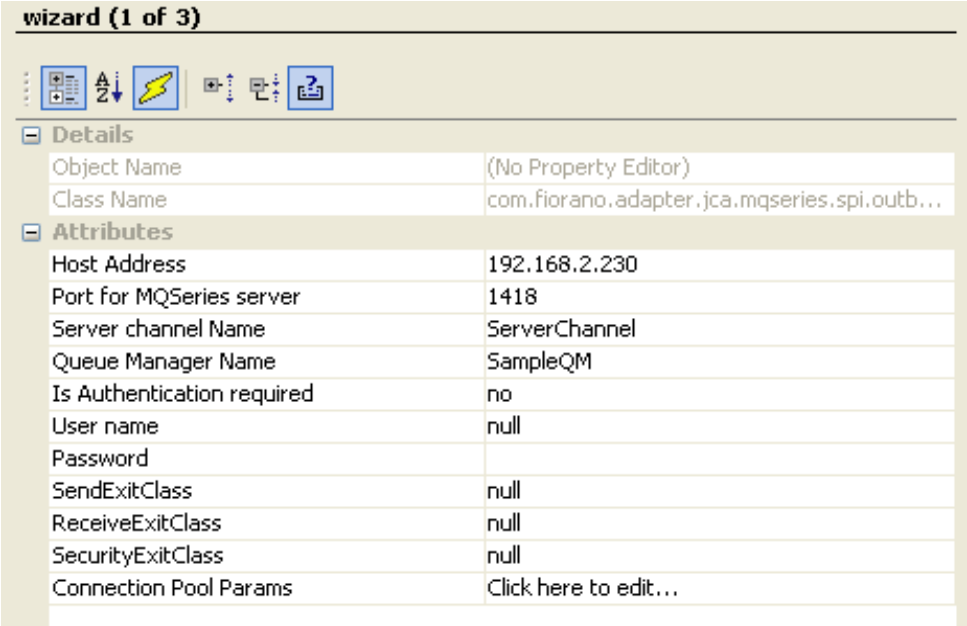


Figure 12: Connection configuration details in MCF panel

4.7.9.2.1 Attributes

4.7.9.2.1.1 Host Address

The hostname or IP address of the machine on which **IBM WebSphereMQ** Server is running. If connecting to **IBM WebSphereMQ** on the same machine on which the component is running, use localhost.

4.7.9.2.1.2 Port for MQSeries server

The port number on which IBM WebSphere MQ listens for connection requests to connect to configured Queue Manager. To view the port number for required Queue Manager (value for property **Queue Manager Name**, expand the node **IBM WebSphere MQ** > **Queue Managers > SampleQM > Advanced > Listeners** as shown in Figure 13. The value in **Port** column is the required port number.

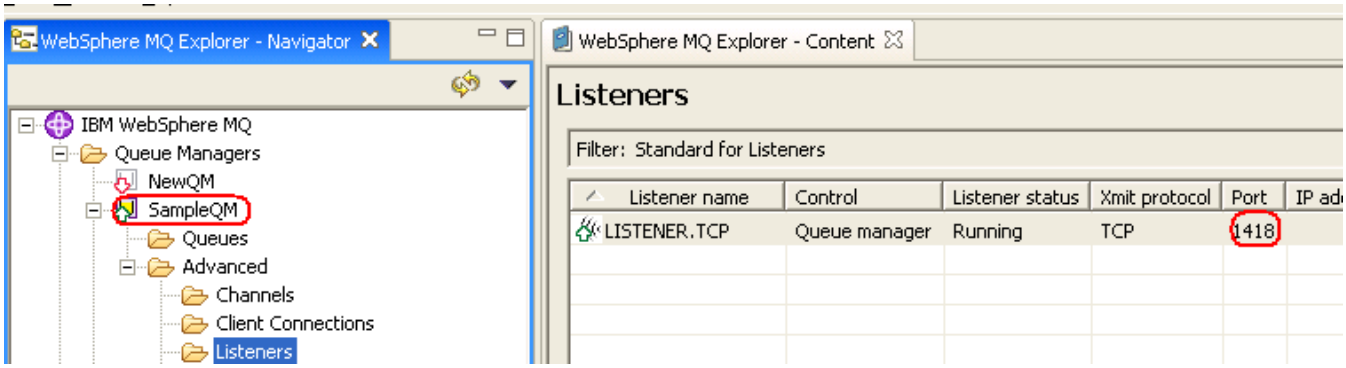


Figure 13: Port number of Queue Manager Sample QM

4.7.9.2.1.3 Server channel Name

The case-sensitive name of the channel to be used for communicating with the Queue Manager.

4.7.9.2.1.4 Queue Manager Name

The name of the Queue Manager in which destination queue is present.

4.7.9.2.1.5 Is Authentication required

- **yes**

  Use authentication when connecting to the MQSeries Server. When this value is selected, values for properties **Username** and **Password** are used for authentication.

- **no**

  Do not use authentication when connecting to the MQSeries Server. When this value is selected, values for properties **Username** and **Password** are be used for authentication.

4.7.9.2.1.6 Username

The user name to be used to connect to the MQSeries Queue Manager. The ID is used to identify the WebSphere® MQ client. It overrides the value of WebSphere MQ environment variable MQ_USER_ID.

If no security exit is defined for this client, the value of userID is transmitted to the server and is available for use by the server security exit.

The default value is "" (empty string).

4.7.9.2.1.7 Password

The password to be used to connect to the MQSeries Queue Manager. The password is used to verify the identity of the WebSphere® MQ Client. It overrides the value of MQEnvironment variable MQ_PASSWORD .

If a security exit is not defined for this client, the value of password is transmitted to the server and is available to the server security exit when it is invoked.

The default value is "" (empty string).

4.7.9.2.1.8 Send Exit Class

The class that implements the MQsendExit interface. This class allows you to examine and possibly alter the data sent to the queue manager by the MQSeries client. At runtime new instance of this class is created and assigned to the variable MQEnvironment.sendExit (class in IBM MQSeries API).

4.7.9.2.1.9 Receive Exit class

The class that implements the MQReceiveExit interface. This class allows you to examine and possibly alter the data received from the queue manager by the MQSeries client. At runtime new instance of this class is created and assigned to the variable MQEnvironment.receiveExit.

4.7.9.2.1.10 SecurityExit Class

The class that implements the MQSecurityExit interface. This class allows you to customize the security flows that occur when an attempt is made to connect to a queue manager. At runtime, new instance of this class is created and assigned to the variable MQEnvironment.securityExit.

A WebSphere MQ JMS application can use channel security, send, and receive exits on the MQI channel that starts when the application connects to a queue manager. An application connects to a queue manager by setting channel related fields or environment properties in the MQEnvironment class. Further information can be found at http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp?topic=/com.ibm.mq.csqzaw.doc/uj21370_.htm

4.7.9.3 Interaction Configuration

Business logic configuration details are configured in the second panel, **Interaction Configurations**. The Figure 14 illustrates the panel with expert properties view enabled.

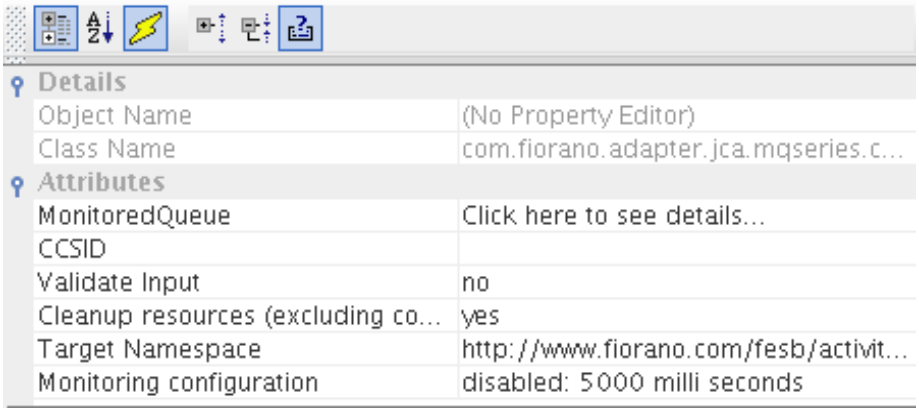


Figure 14: Business logic configuration in Interaction Configurations panel

4.7.9.3.1 Attributes

4.7.9.3.1.1 MonitoredQueue

This property defines the queue on **IBM WebSphere MQ from** which messages are received, format of message, and message selection options.

Click the ellipsis button to launch an editor for providing these configurations.

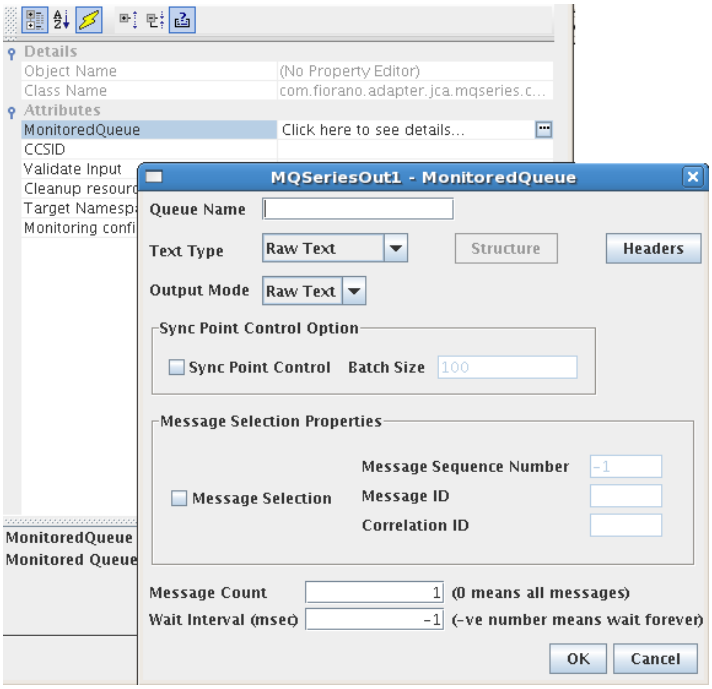


Figure 15: Launching editor for configuring queue and message selection options

- **Queue Name**

  Name of the queue on **IBM WebSphere MQ** from which messages are received.

- **Text Type**

  An MQ message contains message descriptor (a set of headers that define the message) and data. The data is stored in binary format. Multiple fields are written in a sequence using a type specific API. The data in the message is read by the consuming application exactly in the same order as the data is written.

  The MQSeriesOut component parses messages read from a MQ queue based on **Structure** and **Headers** configurations in the CPS. It builds output message based on **Text Type**, **Output Mode**, **Structure** and **Headers** configurations in the CPS.

  - **Parse Message**

    This option is used when the MQ message contains multiple fields in the message body. When this option is selected, both **Structure** button and **Headers** button are enabled. Fields in the MQ message are defined in **Structure** editor.

    The MQ message is parsed based on the fields defined and a XML message containing fields and headers with their respective values is generated. The output message is always a XML message.

  - **Raw Text**

    This option is used if MQ message contains only a single string field. MQ message is read from the queue and the data from the message is set in text format on output message. Any headers that have to be read from the MQ message are defined in **Headers** editor. The output message can either be in raw text or in XML format based on the value for property **Output Mode**.

    When this option is selected, **Structure** button is disabled and **Headers** button is enabled.

    Refer to section Input and Output for details about the effects of these configurations on input and output structures.

- **Output Mode**

  When **Text Type** is Raw Text, format of the message to be sent on output port can be set either as XML or Raw Text.

  There are two formats for defining the output message:

  - **Raw Text**

    This option is used when the user wants the output message to be set as raw text. Headers defined in CPS are set as message properties on output message.

  - **XML**

    This option is used when the user wants the output message in XML format. Output XML contains values for header fields defined in CPS and message body which contains the text read from MQ message.

    Note: When the **Include RFH2 Headers** is selected in **Headers** option, the Output Mode gets disabled and output message is always set to **xml** format.

- **Structure**

  This button is enabled only when **Text Type** is Parse Message. Click Structure button to launch editor to define the fields of the message, as shown in Figure 16.

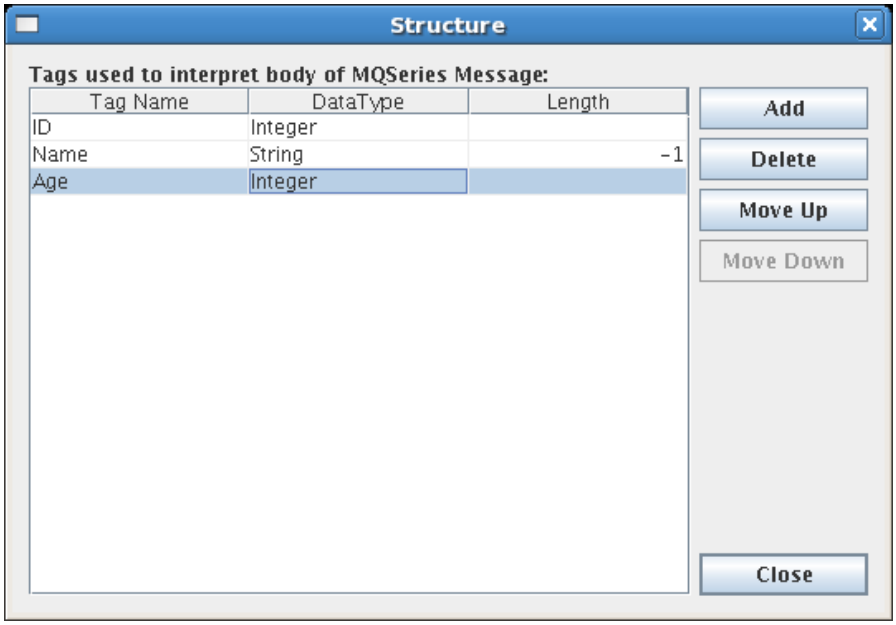


Figure 16: Editor to define fields in message

The editor contains a table with **Tag Name**, **Data Type** and **Length** columns. Each row corresponds to a field in the MQ message. The value in **Tag Name** column is the element name for the element in XML structure which holds data for the field. The value in **Data Type** column is the type of the data that is held in the field. It can only contain following values (case-sensitive) - Char, Double, Float, Integer, Long Integer, Short Integer, UTF, String, Boolean, Byte Array. The value in **Length** column is the length of the field that has to be read from MQ message. This value is applicable for only String and Byte Array data types and the value is ignored for other data types. For String data type, default value is -1 which means whole message is read from MQ message and is set on the output message. Figure 16 shows the definition of structure to build the output message from MQ message with data for an ID, name and age of a person.

Order of the fields is important and the order of the rows in this table defines the order of fields in the MQ message. **Move Up** and **Move Down** buttons are used to reorder the fields. **Add** and **Delete** buttons are used to add or remove the fields.

Schema for output XML is built based on the fields defined here. However, the types for elements in schema do not use the types defined here. All the elements are defined as string type and hence, the schema does not validate content of element with appropriate data type.

Note:

- Data in output XML for Boolean type fields contains true or false.
- Data in output XML for Byte Array type fields contains Base64 encoded string. The **Base64-Decode** function under **Advanced Functions** in **Fiorano Mapper** can be used to convert Base64 encoded string into byte array.
- If the order of elements defined in **Structure** editor differs from order of fields in MQ message or if there are additional fields defined in **Structure** editor or if some of the fields present in the MQ message are not defined in the **Structure** editor, one of the following happens:

- error when building output message from MQ message.
- mixed up values between fields.
- junk data is set on output message.

Refer to section Input and Output for details about the effects of these configurations on input and output structures.

- **Headers**

  Click **Headers** button to launch an editor, Figure 17 defines MQ message descriptor(MQMD) fields that have to be read from the MQ message and set on output message.

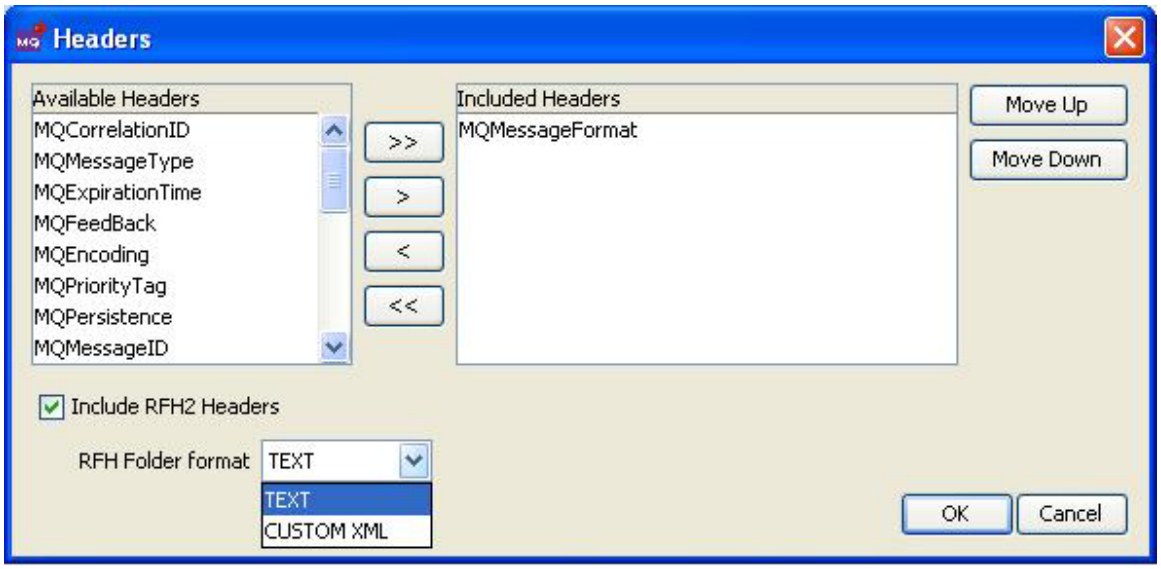


Figure **17: Headers editor**

Defining MQMD headers

Available Headers - Contains MQMD headers that are defined on MQ message.

Included Headers - Contains MQMD headers that can be defined on output message and whose values are taken from MQ message received from the queue.

Selected headers are set in Output XML when

- Text type is parse message or
- Text type is Raw Text and Output mode is XML

When output message is raw text, selected headers are set as message properties on the output message.

Headers in **Available Headers** and **Included Headers** together contain all MQMD headers.

MQMD headers can be selected or unselected as

- To read values of MQMD headers from MQ message and set them on output message, select required headers from **Available Headers** list and click > .
- To read values of all MQMD headers from MQ message and set them on output message, click >> .
- To remove any of selected headers, select required headers from **Included Headers** and click < .
- To remove all selected headers, click << .

Refer to section **MQMD headers** for default values.

Include RFH2 Headers- Select this option to parse MQRFH2 headers present in MQ message and set them on output message.If this option is not selected,then MQRFH2 headers are discarded and the output message contains only message data present in MQ message.

RFHFolder format - This property is used to specify the way in which the variable portion of the RFHHeaders has to be sent in output XML. The variable portion of a RFH Header contains variable number of MQHRF2 folders. Refer to link http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp?topic=/com.ibm.mq.csqzak.doc/js07180.htm for additional details. It can be chosen as explained below

- TEXT

  When this option is chosen the RFHFolders in the header is output as CDATA. For example a sample jms Folder
  <jms><Dst>queue:///Test</Dst><Dlv>4</Dlv><Tms>3543534</Tms></jms>
  present in the retrieved message is output as
  <![CDATA[<jms><Dst>queue:///Test</Dst><Dlv>4</Dlv><Tms>3543534</Tms></jms>]]>
- CUSTOM XML

  When this opton is chosen the RFHFolders in the header is output as elements confirming to a schema. For example a sample jms Folder
  <jms><Dst>queue:///Test</Dst><Dlv>4</Dlv><Tms>3543534</Tms></jms>
  present in the retrieved message is output as

```
<ns1:Folder type="jms">
<ns1:Property name="Dst">queue:///Test</ns1:Property>
<ns1:Property name="Dlv">4</ns1:Property>
<ns1:Property name="Tms">3543534</ns1:Property>
</ns1:Folder>
```

Note:

- When the **Include RFH2 Headers** is selected, the output message is always in XML format and **Output Mode** is disabled.
- The component is able to parse **MQRFH2** headers present in MQ message. Any other headers except **MQMD** and **MQRFH2** headers (if present) are not parsed and are set in message body of the output message.

Refer to section Input and Output for details about the effects of these configurations on input and output structures

MQMD Headers

Only some of most commonly used MQMD headers are provided in the CPS of the component. Refer to Table 1 for a short description and default value used for each MQMD header. Refer to link http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp?topic=/com.ibm.mq.csqzak.doc/mqmd.htm for detailed information on MQMD headers.

Table 1 Short descriptions, default values and data types of MQMD headers used in the component

MQMD header name	Description	Default value	Data type
MQApplicationName	Name of application that put the message.	Empty string (“”)	String
MQApplicationType	Type of application that put the message	0 (MQAT_NO_CONTEXT)	Integer
MQCharacterSet	Character set identifier of character data in the message	0 (MQCCSI_Q_MGR)	Integer
MQCorrelationID	A byte string that the application can use to relate one message to another, or to relate the message to other work that the application is performing	null (MQCI_NONE)	byte array as a hex string
MQMessageID	A byte string that is used to distinguish one message from another. The message identifier is a permanent property of the message, and persists across restarts of the queue manager.	null (MQMI_NONE)	byte array as a hex string
MQDeliveryMode	Delivery mode indicating whether the message survives system failures and restarts of the queue manager.	0 (MQPER_NOT_PERSISTENT)	Integer
MQExpirationTime	A period of time expressed in tenths of a second, set by the application that puts the message. The message becomes eligible to be discarded if it has not been removed from the destination queue before this period of time elapses.	-1 (MQEI_UNLIMITED)	Integer

MQMD header name	Description	Default value	Data type
MQEncodingBinaryIntegers	Subfield of encoding header that specifies encoding for binary integers	1 (MQENC_INTEGER_NORMAL)	Integer
MQEncodingPackedDecimal	Subfield of encoding header that specifies encoding for packed-decimal integers	16 (MQENC_DECIMAL_NORMAL)	Integer
MQEncodingFloatPointNumbers	Subfield of encoding header that specifies encoding for floating-point integers	256 (MQENC_FLOAT_IEEE_NORMAL)	Integer
MQPriorityTag	Priority of the Message	-1 (MQPRI_PRIORITY_AS_Q_DEF)	Integer
MQReplyToQueueName	Name of the message queue to which the application that issued the get request for the message sends MQMT_REPLY and MQMT_REPORT messages	Empty string (“”)	String
MQMessageType	Type of message	8 (MQMT_DATAGRAM)	Integer
MQUserId	User identifier of the application that originated the message. The queue manager treats this information as character data, but does not define the format of it.	Empty string (“”)	String
MQMessageFormat	A name that the sender of the message uses to indicate to the receiver the nature of the data in the message	null	String

Sync Point Control Option

- **Sync Point Control**

  The **IBM WebSphere MQ series** support transactions similar to JMS transaction. Select the check box if the transaction has to be committed after receiving a batch of messages. On selecting this option, the component assigns the message with sync point control. The message is not visible outside the unit of work (in this case, the sync point batch size) until the unit of work is committed. If the unit of work is rolled back from the server, the message is deleted.

  The **Batch Size** is taken into account only if this check box is checked.

- **Batch Size**

The number of messages after which a commit should be performed if the check box **Sync Point Control** is checked. The Batch size is counted based on the number of messages that are successfully received from MQ Queue. If any message could not be sent to MQ Queue due to an error that message is not counted but the count continues.

Note: If the batch size is 'n', then all the n messages are sent out as a single aggregated message.

Message Selection Properties

- **Message Selection**

  This option is used to receive specific messages from the MQ queue. Selection is based on the MQMD headers **Message ID**, **Correlation ID** and **Message Sequence Number**.

  - **Message Sequence Numbe**r

    This is the sequence number of a logical message within a group.

  - **Message ID**

    This is a byte string that is used to distinguish one message from another.

  - **Correlation ID**

    This is a byte string that the application can use to relate one message to another, or to relate the message to other work that the application is performing

- **Message Count**

  The number of messages to be received from the Queue. If the message count is specified as 'n', then the component aggregates and sends all the messages at a time after receiving 'n' messages. If Sync Point Control is checked this property is disabled and batch size is used to aggregate. Default value is 1. If this value is 0, all messages are fetched till the time out occurs as specified by Wait Interval and the aggregated message is sent to output port.

- **Wait Interval**

  Maximum time in milliseconds the component should wait for a message on the MQ queue. When a request is sent to get message from queue, a message is received if it is present on the queue. If there are no messages on the queue, it waits for the specified interval of time for the message. Default value is -1 which specifies infinite wait time that is, waits until a message is received. Any value which is less than 0 (zero) specifies infinite time.

  This property works in conjunction with **Message Count** property.

  Example: If the **Message Count** is set to 10000 and **Wait Interval** is set to 10 seconds. The component tries to fetch 10 messages from MQ queue. For each request to fetch message, if an unconsumed message is present on the MQ queue it is immediately fetched. If there are no messages on the MQ queue, then the component waits for utmost 10 seconds. If a message is not available on the queue during this wait time, the component builds output XML based on messages received so far for this request and sends it on the output port. If there are no messages received an empty output is sent on the output port.

  If there are no messages on the queue till the time out occurs, then an empty message is sent to output port. Please refer to **Scenario 4** in Functional demonstration section for the affects of this property on output message.

Note: When Message Count is 0 (zero) and **Wait Interval** is -1, the component receives messages from the queue and processes them for aggregation but no message is sent to output port. This combination of **Message Count** and **Wait Interval** should not be used.

4.7.9.3.1.2 CCSID

Coded Character Set Identification (should be an integer or null), in case of null, 819 (ISO 8859-1 ASCII) is used as CCSID. Data in MQ message is always present in the form of bytes. This field is used while converting this data bytes (decodes bytes into string using the specified Charset) and this conversion depends on the type of output message.

- When text type is Raw Text and Output mode is Raw Text, the data bytes in MQ message is converted to string data using the CCSID value and set as output message.
- When text type is Raw Text and Output mode is XML, the data bytes in MQ message is converted to string data using the CCSID value and set as message body in output message.
- When text type is parse message, the data bytes in MQ message are read based on the structure fields. Whenever there is a String data field, CCSID value is used for conversion of the data bytes.
- The behavior is unspecified if the data bytes present in message are not valid in specified Charset. There can be error while building the output message or they can be junk data in output message.

4.7.9.3.1.2 Action when message is not available?

This property specifies the action to be taken when there are no messages on the queue within the timeout specified.This property is disabled when the wait interval(timeout) is -1 (infinite).

- **Send Empty Message**: This value is set by default.An empty message is sent to output port when 'Text type' is set to 'Raw Text'. When Texttype is set to 'Parse message', a default xml message with empty 'MQMessages' element is sent to output port.
- **Treat as Exception**: An exception message is sent to error port and no message is sent to output port.
- **No Action**: No message is sent to either output port or error port.

4.7.9.3.2 Using Named Configurations

Changes are done in eStudio w.r.t to Named Configurations. With the implementation of Named Configurations MCF panel is changed as shown in Figure below.

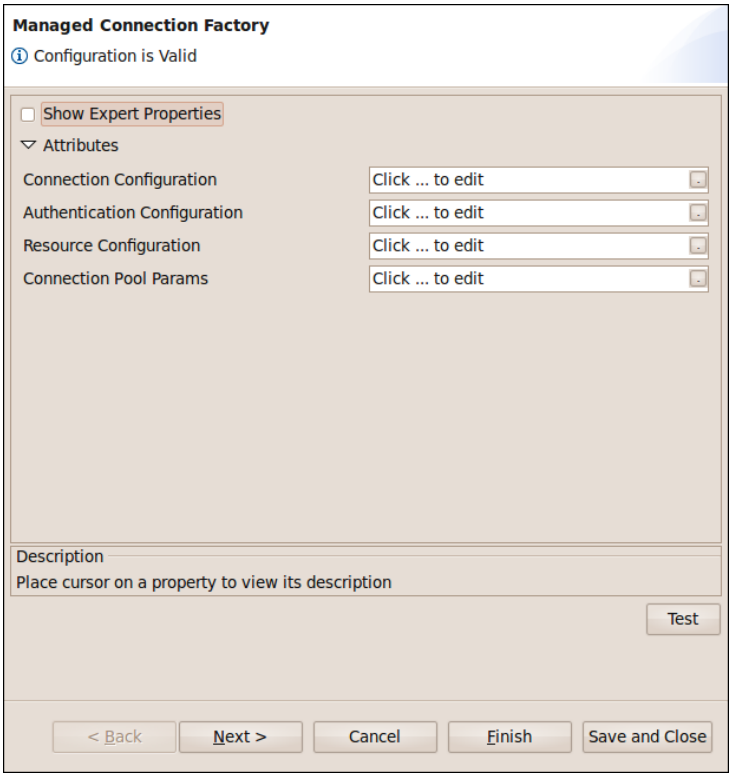


Figure: Modified Managed Connection Factory

Some properties are grouped as one property in the modified configuration panel. Host Address, Port for MQSeries server, Server channel Name and Queue Manager Name which were present previously in the MCF panel are grouped under the name Connection Configuration.

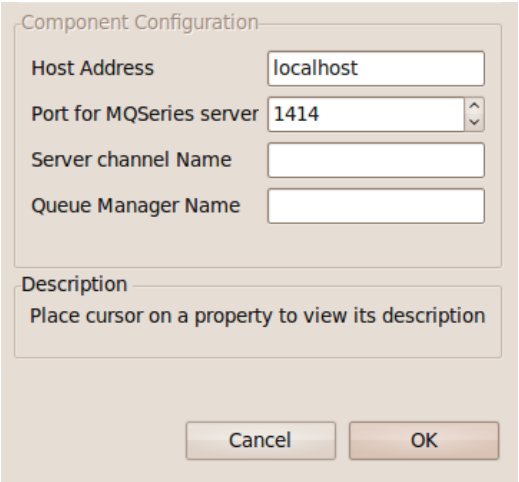


Figure: Connection Configuration

UserName and Password required for Authentication which were present previously in the MCF panel are grouped under the name Authentication Configuration.

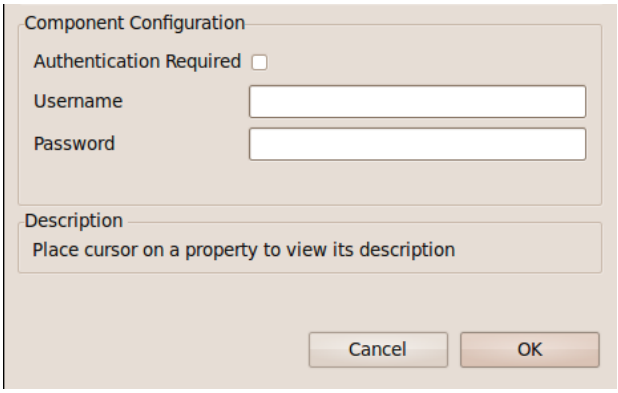


Figure: Authentication Configuration

SendExit Class, ReserveExit Class and SecurityExit Class which were present previously in the MCF panel are grouped under the name Resource Configuration.

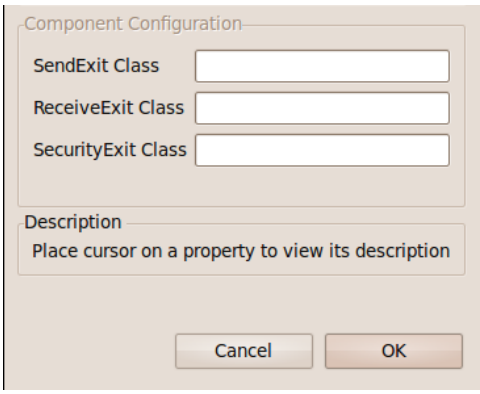


Figure: Resource Configuration

Similarly Monitored Queue Panel is also modified to use Named Configurations.

4.7.9.4 Input and output

4.7.9.4.1 Input

The input schema for the component is fixed. The schema is shown in Figure 18.

Structure | Namespaces | Content

XML Item	Value Type
<ns1:QueueProperties>	
<ns1:QueueName>?	#string
<ns1:EBCDIC_TO_ASCII_Conversion>	#string
<ns1:IsSelectionRequired>?	#string
<ns1:MessageID>?	#string
<ns1:CorrelationID>?	#string
<ns1:MessageSequenceNumber>	#int
<ns1:MessageCount>	#int
<ns1:WaitInterval>	#int
<ns1:IsSyncPointControl>?	#string
<ns1:SyncSize>?	#string

Figure 18: Schema for input message

All the elements present in the Input message are optional. If a particular element is present, its value is used while receiving the message; else default value specified in CPS is used.

Test BC

Input Message | Output Message

```
<ns1:QueueProperties xmlns:ns1="http://www.fiorano.com/fesb/activity/MQSeriesOutlOut/In
</ns1:QueueProperties>
```

Figure 19: Input message with no elements specified

Note: The component also accepts null message as an input. When null/empty message is sent as an input, then the value specified in CPS is used.

Data in the fields EBCDIC_TO_ASCII_CONVERSION, IsSelectionRequired, IsSyncPointControl should be of boolean type and for data in SyncSize should be of Integer type, if data is not present in proper data types, error occurs while creating the output message.

4.7.9.4.2 Output

The output schema for the component is based on the configuration selected.

When the **Text Type** is set to Raw Text and **Output Mode** is set to XML Text, then there is no schema defined for the output. The output message text contains a single String field. Selected MQMD headers in CPS with values present of MQ message are set on output message.

When the **Text Type** is set to Parse Message the output schema varies based on the configuration of structure and headers.

- When only fields of message body are defined in **Structure** dialog as shown in Figure 20.

Tags used to interpret body of MQSeries Message:

Tag Name	DataType
Name1	String
Name2	String

Figure 20: Fields defined for message body

The output schema is defined as shown in Figure 21 and a sample output message is shown in Figure 22.

```
<ns1:MQMessages>
    <ns1:MQMessage>*
        <ns1:MessageBody>
            <ns1:Name1>          #string
            <ns1:Name2>          #string
```

Figure 21: Schema when only fields of message body are defined

All the fields defined are added as child elements under **MessageBody** in the same order in which these fields are defined in CPS. MQMD headers are not explicitly set by the component.

- When fields of message body are defined in **Structure** dialog as shown in Figure 20 and MQMD headers are defined in **Headers** dialog as shown in Figure 23

```
<ns1:MQMessages xmlns:ns1="http://www.fiorano.com/fesb/activity/MQSeriesOut2Out/C
   <ns1:MQMessage>
      <ns1:MessageBody>
         <ns1:Name1>Name1</ns1:Name1>
         <ns1:Name2>Name2</ns1:Name2>
      </ns1:MessageBody>
   </ns1:MQMessage>
</ns1:MQMessages>
```

Figure 22: MQMD headers are defined in Headers

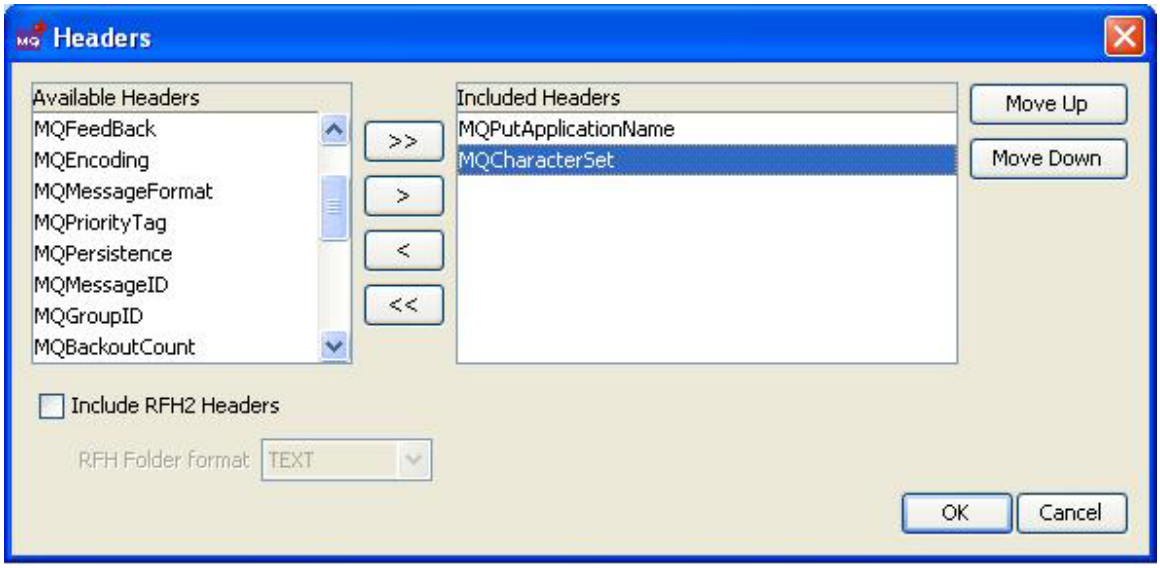


Figure 23: Selecting MQMD headers whose values have to be taken from input XML

The output schema is defined as shown in Figure 24 and a sample output is shown in Figure 25.

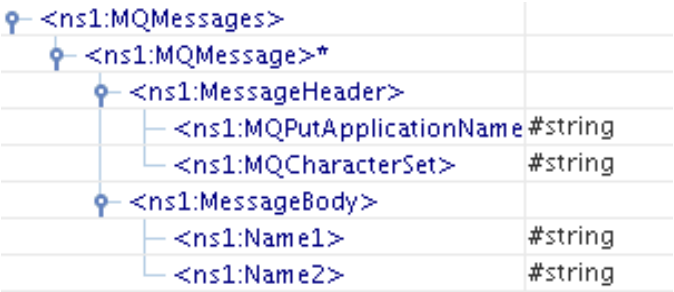


Figure 24: output schema

```
<ns1:MQMessages xmlns:ns1="http://www.fiorano.com/fesb/activity/MQSeriesOut2Out/C
   <ns1:MQMessage>
      <ns1:MessageHeader>
         <ns1:MQPutApplicationName>MQPutApplicationName</ns1:MQPutApplicationName
         <ns1:MQCharacterSet>MQCharacterSet</ns1:MQCharacterSet>
      </ns1:MessageHeader>
      <ns1:MessageBody>
         <ns1:Name1>Name1</ns1:Name1>
         <ns1:Name2>Name2</ns1:Name2>
      </ns1:MessageBody>
   </ns1:MQMessage>
</ns1:MQMessages>
```

Figure 25: Sample input XML for schema defined in Figure 23

All the fields defined are added as child elements under **MessageBody** in the same order in which these fields are defined in CPS and all the MQMD headers selected are added as child elements under **Message Header** in the same order in which these headers are defined in CPS.

When the **Text Type** is set to Raw Text and **Output Mode** is set to XML Text, the output schema varies based on the configuration headers.

- When there are no selected headers in CPS, then the output Schema is set as shown in Figure 26 and a sample output is shown in Figure 27.

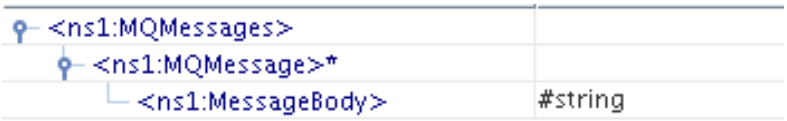


Figure 26: Schema when there are no selected headers

```
<ns1:MQMessages xmlns:ns1="http://www.fiorano.com/fesb/activity/MQSeriesOu
   <ns1:MQMessage>
      <ns1:MessageBody>MessageBody</ns1:MessageBody>
   </ns1:MQMessage>
</ns1:MQMessages>
```

Figure 27: Ouput XML when there are no selected headers

- When MQMD headers are defined in Headers dialog as shown in Figure 23, then the output Schema is set as shown in Figure 28 and a sample output is shown in Figure 29.

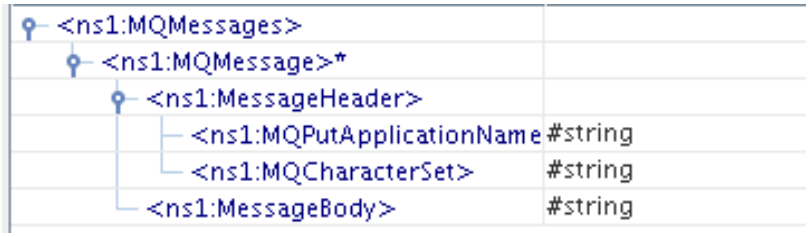


Figure 28: Schema when there are selected headers

```
<ns1:MQMessages xmlns:ns1="http://www.fiorano.com/fesb/activity/MQSeriesOut2Ou
   <ns1:MQMessage>
      <ns1:MessageHeader>
         <ns1:MQPutApplicationName>MQPutApplicationName</ns1:MQPutApplicationN
         <ns1:MQCharacterSet>MQCharacterSet</ns1:MQCharacterSet>
      </ns1:MessageHeader>
      <ns1:MessageBody>MessageBody</ns1:MessageBody>
   </ns1:MQMessage>
</ns1:MQMessages>
```

Figure 29: Ouput XML when there are selected headers

When **Include RFH2 Headers** option is selected in **Headers** as shown in Figure 30 and **Text Type** is set to **Raw Text**, then output schema is set as shown in Figure 31 and a sample output is shown in the Figure 32.

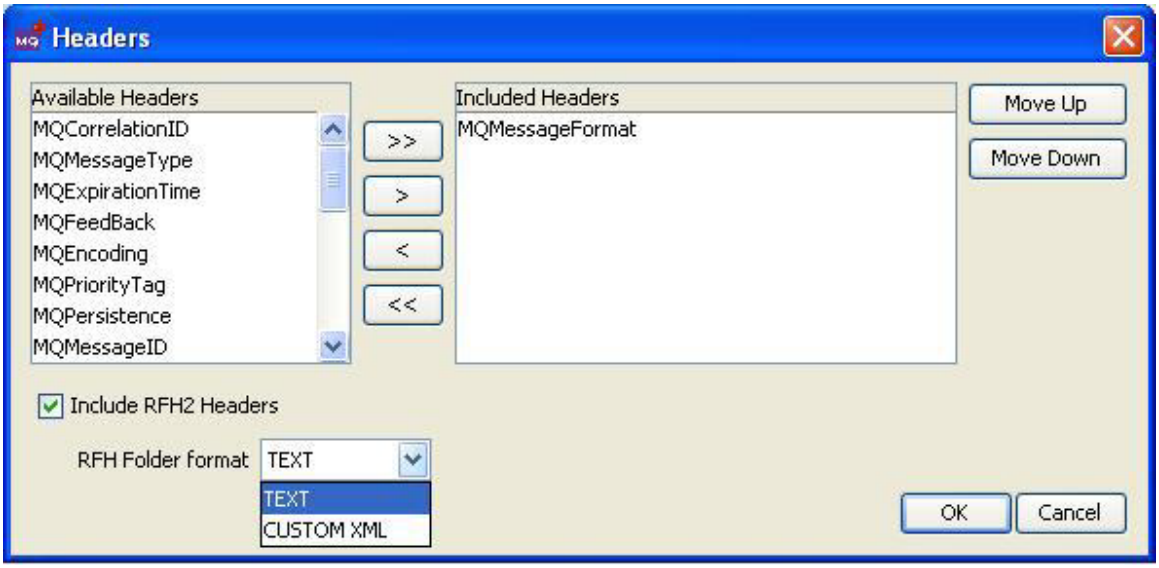


Figure 30: Option to include RFH2 headers

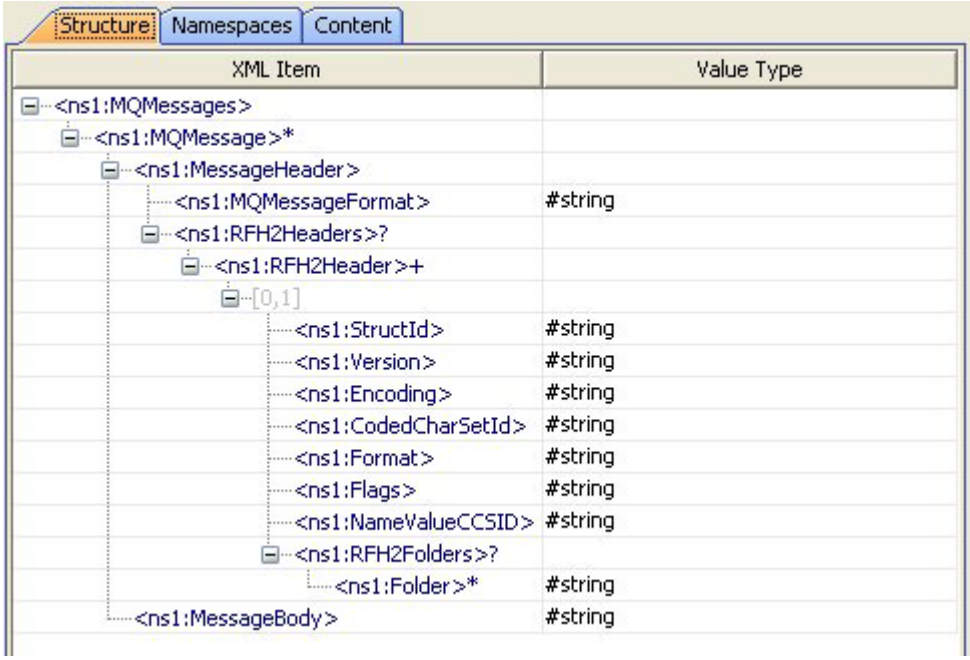


Figure 31: Output Schema when Include RFH2 Headers is selected

```
<nsl:MQMessages xmlns:nsl="http://www.fiorano.com/fesb/activity/MQSeriesOutlOut/Out">
   <nsl:MQMessage>
      <nsl:MessageHeader>
         <nsl:MQMessageFormat>MQMessageFormat</nsl:MQMessageFormat>
         <nsl:RFH2Headers>
            <nsl:RFH2Header>
               <nsl:StructId>StructId</nsl:StructId>
               <nsl:Version>Version</nsl:Version>
               <nsl:Encoding>Encoding</nsl:Encoding>
               <nsl:CodedCharSetId>CodedCharSetId</nsl:CodedCharSetId>
               <nsl:Format>Format</nsl:Format>
               <nsl:Flags>Flags</nsl:Flags>
               <nsl:NameValueCCSID>NameValueCCSID</nsl:NameValueCCSID>
               <nsl:RFH2Folders>
                  <nsl:Folder>Folder</nsl:Folder>
               </nsl:RFH2Folders>
            </nsl:RFH2Header>
         </nsl:RFH2Headers>
      </nsl:MessageHeader>
      <nsl:MessageBody>MessageBody</nsl:MessageBody>
   </nsl:MQMessage>
</nsl:MQMessages>
```

Figure 32: Sample output XML when Include RFH2 Headers is selected.

Note: When the **Include RFH2 headers** option is selected, the headers are set in output message in xml format.

For all the messages that are sent to output port, a message property QueueName is set with the value of queue name from which message has been received.

4.7.9.5 Functional Demonstration

4.7.9.5.1 Scenario 1

Configure the component for Text type mode as Raw Text, Output Mode as Raw Text and configure **Headers** as shown in Figure 33.

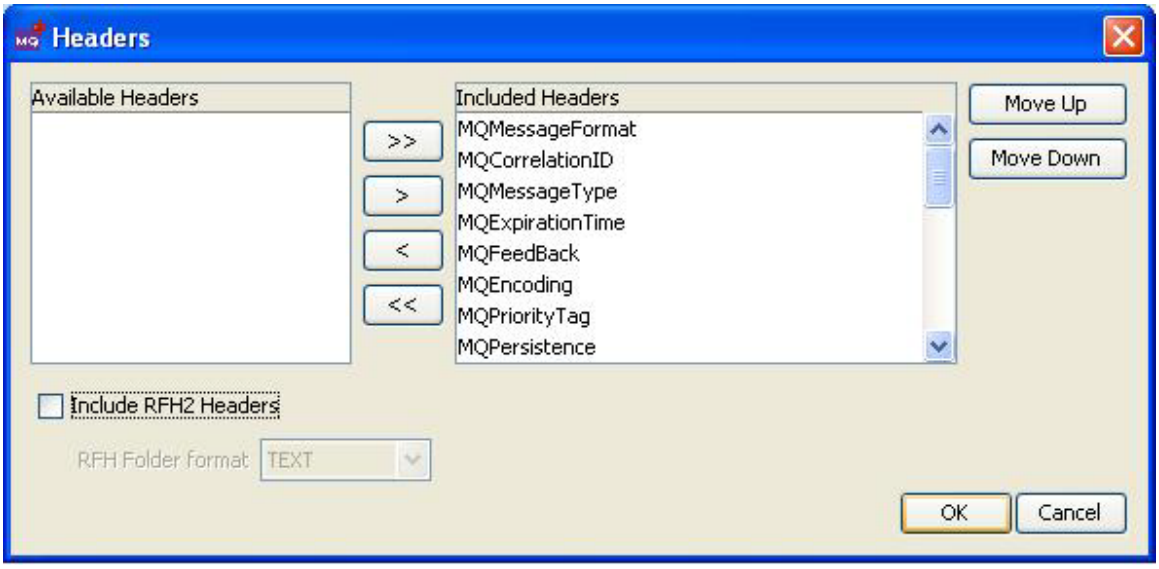


Figure 33: Configuration of headers for Scenario 1

Input Message

```
<ns1:QueueProperties
xmlns:ns1="http://www.fiorano.com/fesb/activity/MQSeriesOut1Out/In">
</ns1:QueueProperties>
```

Output Message

```
TestMessage
```

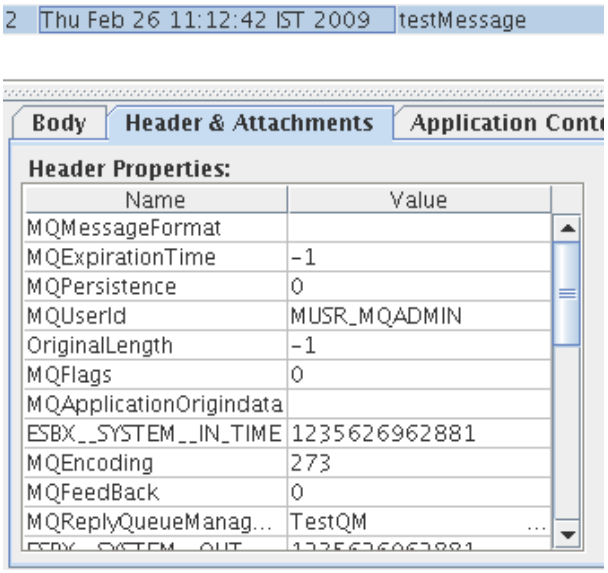


Figure 34: Message properties of output message

4.7.9.5.2 Scenario 2

Configure the component for Text type mode as Raw Text, Output Mode as XML and configure **Headers** as shown in Figure 33.

InputMessage

```
<ns1:QueueProperties
xmlns:ns1="http://www.fiorano.com/fesb/activity/MQSeriesOut1Out/In">
</ns1:QueueProperties>
```

OutputMessage

```
<?xml version="1.0" encoding="UTF-8"?>
<MQMessages xmlns="http://www.fiorano.com/fesb/activity/MQSeriesOut1Out/Out">
   <MQMessage>
      <MessageHeader>

<MQCorrelationID>000000000000000000000000000000000000000000000000</MQCorrelationID>
         <MQMessageType>8</MQMessageType>
         <MQExpirationTime>-1</MQExpirationTime>
         <MQFeedBack>0</MQFeedBack>
         <MQEncoding>273</MQEncoding>
         <MQCharacterSet>819</MQCharacterSet>
         <MQMessageFormat>        </MQMessageFormat>
         <MQPriorityTag>0</MQPriorityTag>
         <MQPersistence>0</MQPersistence>
         <MQMessageID>414D512054657374514D202020202020F41CA64920004004</MQMessageID>
         <MQGroupID>000000000000000000000000000000000000000000000000</MQGroupID>
         <MQBackoutCount>0</MQBackoutCount>
         <MQReplyToQueueName>
</MQReplyToQueueName>
```

```
        <MQReplyQueueManagerName>TestQM
</MQReplyQueueManagerName>
        <MQUserId>MUSR_MQADMIN</MQUserId>
        <MQPutApplicationType>28</MQPutApplicationType>
        <MQPutApplicationName>Websphere MQ Client for Java</MQPutApplicationName>

<MQPutDateTime>java.util.GregorianCalendar[time=?,areFieldsSet=false,areAllFieldsSet=f
alse,lenient=true,zone=sun.util.calendar.ZoneInfo[id="GMT",offset=0,dstSavings=0,useDa
ylight=false,transitions=0,lastRule=null],firstDayOfWeek=1,minimalDaysInFirstWeek=1,ER
A=?,YEAR=2009,MONTH=1,WEEK_OF_YEAR=?,WEEK_OF_MONTH=?,DAY_OF_MONTH=26,DAY_OF_YEAR=?,DAY
_OF_WEEK=?,DAY_OF_WEEK_IN_MONTH=?,AM_PM=0,HOUR=4,HOUR_OF_DAY=4,MINUTE=59,SECOND=46,MIL
LISECOND=50,ZONE_OFFSET=?,DST_OFFSET=?]</MQPutDateTime>
        <MQApplicationOrigindata>    </MQApplicationOrigindata>
        <MQMsgSequenceNumber>1</MQMsgSequenceNumber>
        <MQOffset>0</MQOffset>
        <MQFlags>0</MQFlags>
        <OriginalLength>-1</OriginalLength>
      </MessageHeader>
      <MessageBody>testMessage</MessageBody>
   </MQMessage>
</MQMessages>
```


4.7.9.5.3 Scenario 3

Configure the component for Text type mode as Parse Message and configure **Headers** as shown in Figure 33.

InputMessage


```
<ns1:QueueProperties
xmlns:ns1="http://www.fiorano.com/fesb/activity/MQSeriesOut1Out/In">
</ns1:QueueProperties>
```


OutputMessage


```
<?xml version="1.0" encoding="UTF-8"?>
<MQMessages xmlns="http://www.fiorano.com/fesb/activity/MQSeriesOut1Out/Out">
   <MQMessage>
      <MessageHeader>

<MQCorrelationID>000000000000000000000000000000000000000000000000</MQCorrelationID>
        <MQMessageType>8</MQMessageType>
        <MQExpirationTime>-1</MQExpirationTime>
        <MQFeedBack>0</MQFeedBack>
        <MQEncoding>273</MQEncoding>
        <MQCharacterSet>819</MQCharacterSet>
        <MQMessageFormat>        </MQMessageFormat>
        <MQPriorityTag>0</MQPriorityTag>
        <MQPersistence>0</MQPersistence>
        <MQMessageID>414D512054657374514D202020202020F41CA64920004002</MQMessageID>
        <MQGroupID>000000000000000000000000000000000000000000000000</MQGroupID>
        <MQBackoutCount>0</MQBackoutCount>
```

```
         <MQReplyToQueueName>
</MQReplyToQueueName>
         <MQReplyQueueManagerName>TestQM
</MQReplyQueueManagerName>
         <MQUserId>MUSR_MQADMIN</MQUserId>
         <MQPutApplicationType>28</MQPutApplicationType>
         <MQPutApplicationName>Websphere MQ Client for Java</MQPutApplicationName>

<MQPutDateTime>java.util.GregorianCalendar[time=?,areFieldsSet=false,areAllFieldsSet=f
alse,lenient=true,zone=sun.util.calendar.ZoneInfo[id="GMT",offset=0,dstSavings=0,useDa
ylight=false,transitions=0,lastRule=null],firstDayOfWeek=1,minimalDaysInFirstWeek=1,ER
A=?,YEAR=2009,MONTH=1,WEEK_OF_YEAR=?,WEEK_OF_MONTH=?,DAY_OF_MONTH=26,DAY_OF_YEAR=?,DAY
_OF_WEEK=?,DAY_OF_WEEK_IN_MONTH=?,AM_PM=0,HOUR=4,HOUR_OF_DAY=4,MINUTE=59,SECOND=40,MIL
LISECOND=760,ZONE_OFFSET=?,DST_OFFSET=?]</MQPutDateTime>
         <MQApplicationOrigindata>    </MQApplicationOrigindata>
         <MQMsgSequenceNumber>1</MQMsgSequenceNumber>
         <MQOffset>0</MQOffset>
         <MQFlags>0</MQFlags>
         <OriginalLength>-1</OriginalLength>
      </MessageHeader>
      <MessageBody>
         <Name2>Name2</Name2>
      </MessageBody>
   </MQMessage>
</MQMessages>
```

4.7.9.5.4 Scenario 4

This scenario describes the aggregation of messages using the property Message Count and Wait Interval. Configure the component as described in any of the above 3 scenarios.

InputMessage

```
<ns1:QueueProperties
xmlns:ns1="http://www.fiorano.com/fesb/activity/MQSeriesOut1Out/In">
   <ns1:MessageCount>2</ns1:MessageCount>
 <ns1:WaitInterval>5000</ns1:WaitInterval>
</ns1:QueueProperties>
```

When there are two messages available on the MQ queue, then both of them are fetched as shown in the **OutputMessage**.

OutputMessage

```
<?xml version="1.0" encoding="UTF-8"?>
<MQMessages xmlns="http://www.fiorano.com/fesb/activity/MQSeriesOut1Out/Out">
   <MQMessage>
      <MessageBody>
         <Name1>Name1</Name1>
      </MessageBody>
   </MQMessage>
   <MQMessage>
      <MessageBody>
```

```
        <Name1>Name1</Name1>
      </MessageBody>
   </MQMessage>
</MQMessages>
```

When time out occurs and no message is available on the queue, sample message sent to output port is shown below.

OutputMessage

```
<?xml version="1.0" encoding="UTF-8"?>
<MQMessages xmlns="http://www.fiorano.com/fesb/activity/MQSeriesOut1Out/Out"/>
```

4.7.9.5.5 Scenario 5

This scenario describes to include RFH2 headers in output message .Configure the component for **Text Type** to **Raw Text** and headers as shown in Figure 30.

InputMessage

```
<ns1:QueueProperties
xmlns:ns1="http://www.fiorano.com/fesb/activity/MQSeriesOut1Out/In">
</ns1:QueueProperties>
```

OutputMessage

```
<?xml version="1.0" encoding="UTF-8"?>
<MQMessages xmlns="http://www.fiorano.com/fesb/activity/MQSeriesOut1Out/Out">
   <MQMessage>
      <MessageHeader>
         <MQMessageFormat>MQHRF2  </MQMessageFormat>
         <RFHHeaders>
            <RFHHeader>
               <StructId>RFH </StructId>
               <Version>2</Version>
               <Encoding>273</Encoding>
               <CodedCharSetId>1208</CodedCharSetId>
               <Format>MQSTR   </Format>
               <Flags>0</Flags>
               <NameValueCCSID>1208</NameValueCCSID>
               <VariableData>vardata </VariableData>
            </RFHHeader>
         </RFHHeaders>
      </MessageHeader>
      <MessageBody>test message</MessageBody>
   </MQMessage>
</MQMessages>
```

4.7.9.5.6 Scenario 6

This scenario describes the property 'Action when message is not available' property. Set 'Wait Interval' to a finite value(say 1000 msec) and TextType to 'Parse Message'.

Set 'Action when message is not available' to 'Send Empty Message'.

InputMessage

```
<ns1:QueueProperties
xmlns:ns1="http://www.fiorano.com/fesb/activity/MQSeriesOut1Out/In">
</ns1:QueueProperties>
```

OutputMessage

```
<?xml version="1.0" encoding="UTF-8"?>
<MQMessages xmlns="http://www.fiorano.com/fesb/activity/MQSeriesOut1Out/Out"/>
```

Note: For the same scenario,but when TextType is set to 'RawText', a message with empty content is sent to output port

4.7.9.6 Useful Tips

The correct CCSID should be set for message encoding when transferring messages from AS 400 systems to other platforms and vice versa.

4.7.10 MSMQ Receiver

The MSMQReceiver component is used to receive messages from MSMQ. The name of the queue from which a message needs to be retrieved can be specified using the CPS.

4.7.10.1 Configuration and Testing

The MSMQ server and queue can be configured in the connection properties panel of CPS.

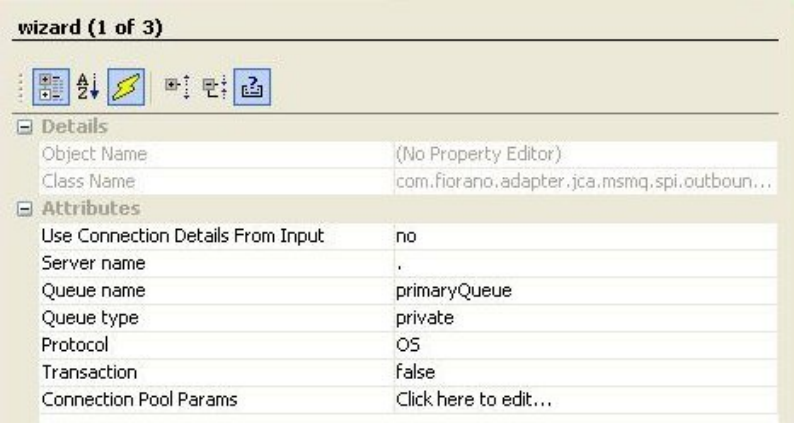


Figure 4.353: Sample MSMQ server configuration

Use Connection Details From Input: Parameters to create the connection can be specified in the input message when this property is set to true. If this property is selected the validation errors in the managed connection factory panel of the CPS are treated as warnings. So user can bypass this step without giving valid configuration and complete the configuration of the component. If valid properties are not provided even in the input message exception will be thrown at runtime.

4.7.10.1.1 Using Named Configurations

Changes are done in eStudio w.r.t to Named Configurations. With the implementation of Named Configurations MCF panel is changed as shown in Figure below.

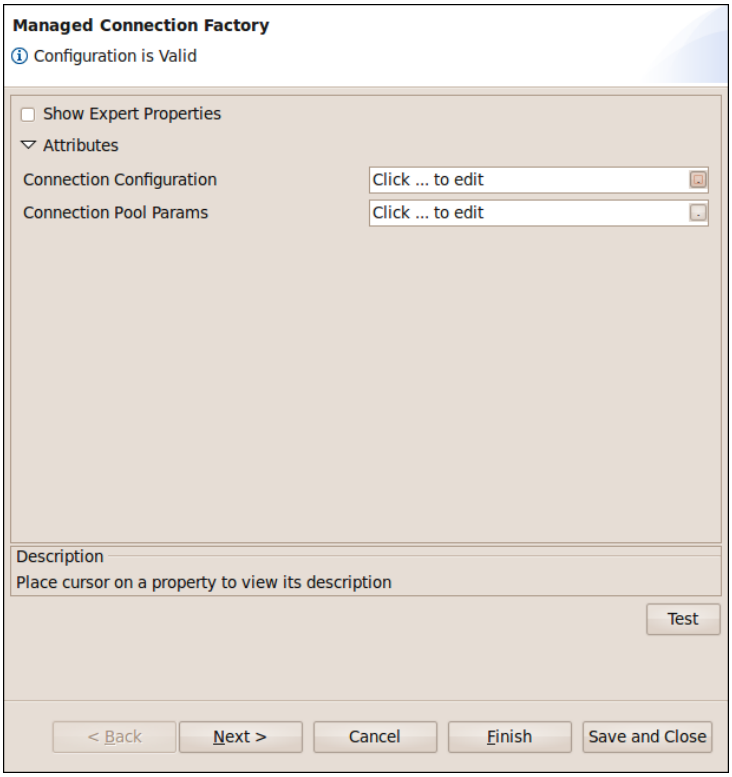


Figure: Modified Managed Connection Factory

Some properties are grouped as one property in the modified configuration panel. Server name, Queue Name, Queue type, Protocol and Transaction which were present previously in the MCF panel are grouped under the name Connection Configuration.

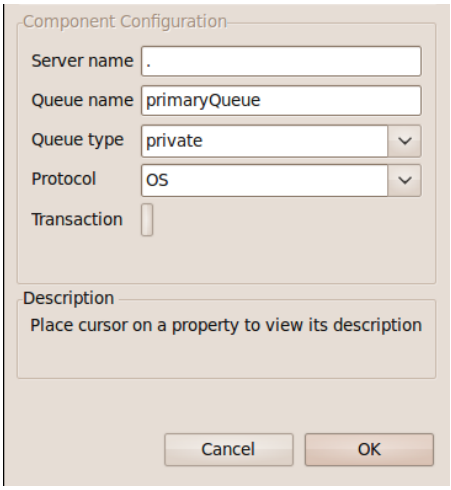


Figure: Connection Configuration

Server connection can be tested from within the CPS by clicking on **test** in the connection properties panel.

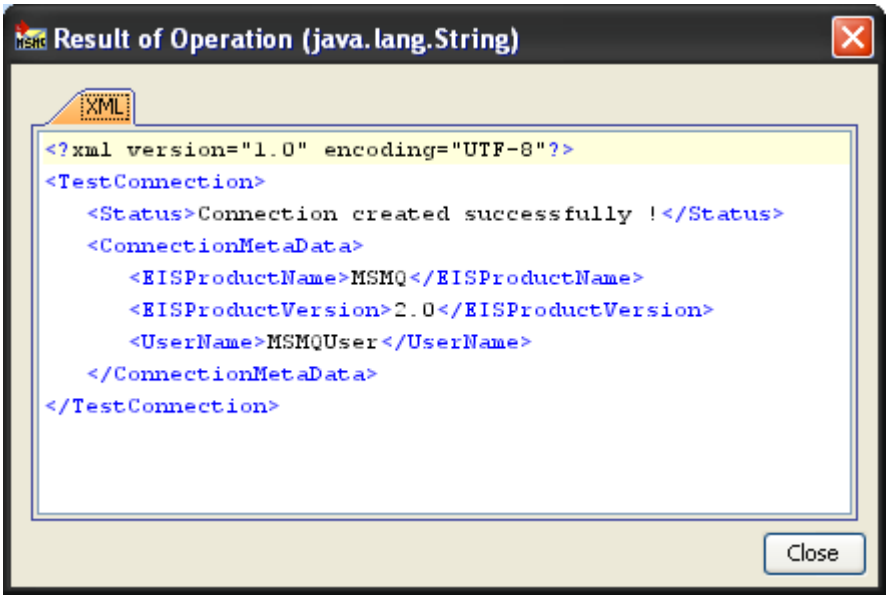


Figure 4.354: Sample connection test result indicating success

No specific information is required to be captured in Interaction properties panel.

The configuration can be tested by sending a test message when you click on the **Test** option in the interaction properties panel.

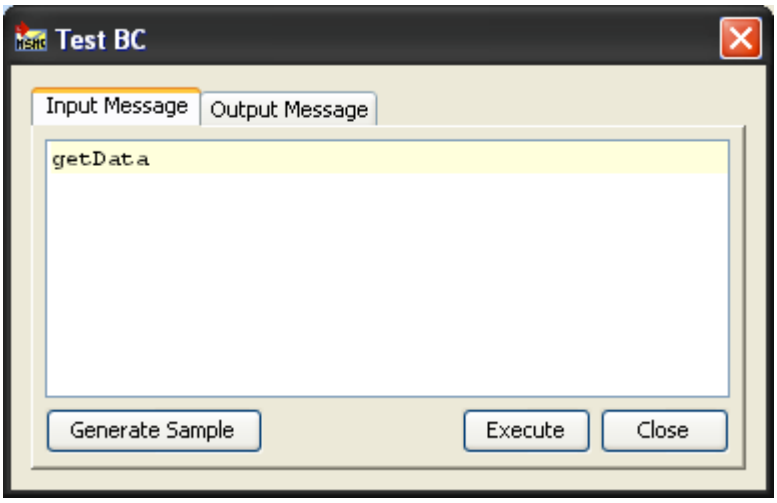


Figure 4.355: Sample input

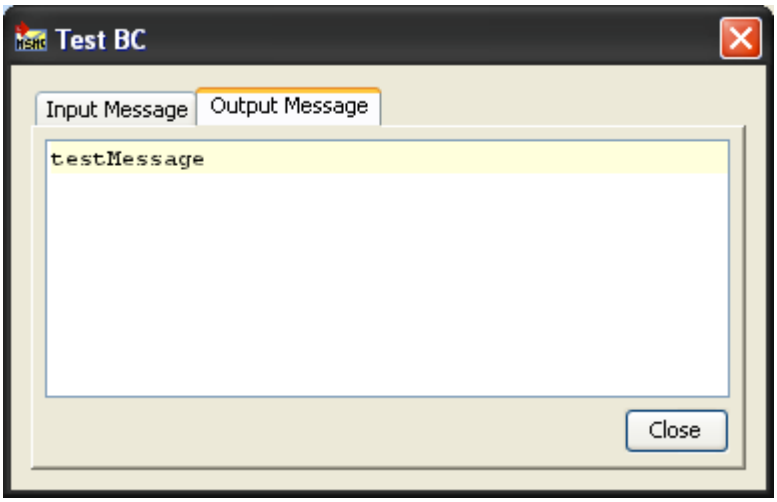


Figure 4.356: Sample response

4.7.10.2 Input and Output

4.7.10.2.1 Input Schema

There is no input schema for this adapter.

When the property Use Connection details from input is chosen, the following input schema with the element ConnectionFactorySettings, is generated. Properties that are used to create the connection are present under this element.

XML Item	Value Type
<ns1:MSMQRequest>	
<ConnectionFactorySettings>?	
<ConnectionProperties>?	
<ServerName>?	#string
<QueueName>?	#string
<QueueType>?	public | private
<Protocol>?	OS | HTTP | HTTPS | TCP | SPX
<Transaction>?	true | false

Figure: Input schema with ConnectionFactorySettings

4.7.10.2.1 Output Schema

There is no output schema for this adapter.

4.7.10.3 Functional Demonstration

4.7.10.3.1 Scenario 1

Receive messages from a local MSMQ Server.

Configure the MSMQ Receiver as described in *Configuration and Testing section* and use feeder and display component to send sample input and check the response respectively.

Figure 4.357: Configuration the MSMQ Receiver

Sample Input

```
getdata
```

Figure 4.358: Demonstrating Scenario 1 with sample input

Sample Output

```
This is a sample message
```

Figure 4.359: Demonstrating Scenario 1 with sample output

4.7.10.4 Use case scenario

In the revenue control packet example the transaction file details are received from an MSMQ server from where they are picked up by other applications for processing.

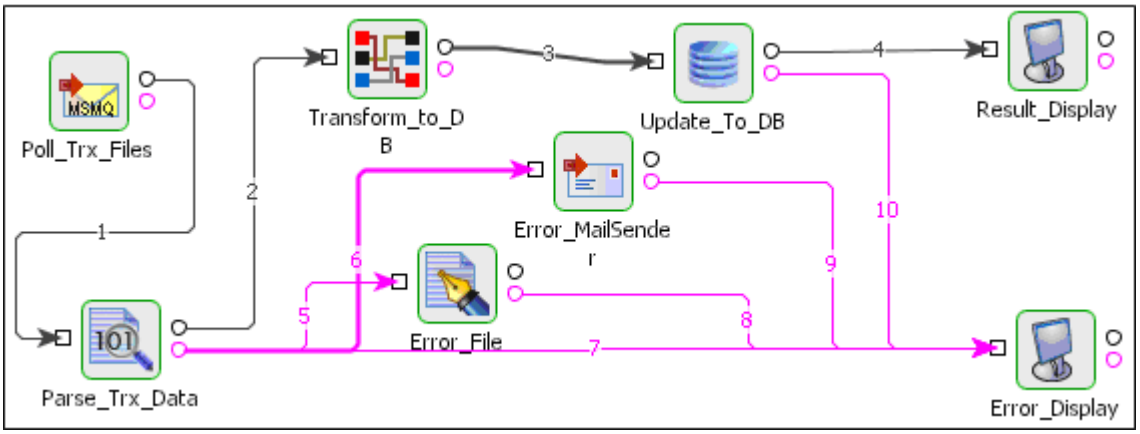


Figure 4.360: Rrevenue control packet

The event process demonstrating this scenario is bundled with the installer. The bundled process shows it as a File Reader component instead of a MSMQ Receiver component.

Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Studio.

4.7.10.5 Useful Tips

- If a queue specified in the CPS does not exist in a local or remote MSMQ server, it is not automatically created.
- This component runs only on Windows Platform.

4.7.11 MSMQ Sender

The MSMQ Sender component is used to send messages to MSMQ. The name of the queue to which a message needs to be sent can be specified using the CPS.

4.7.11.1 Configuration and Testing

The MSMQ server and queue can be configured in the connection properties panel of CPS.

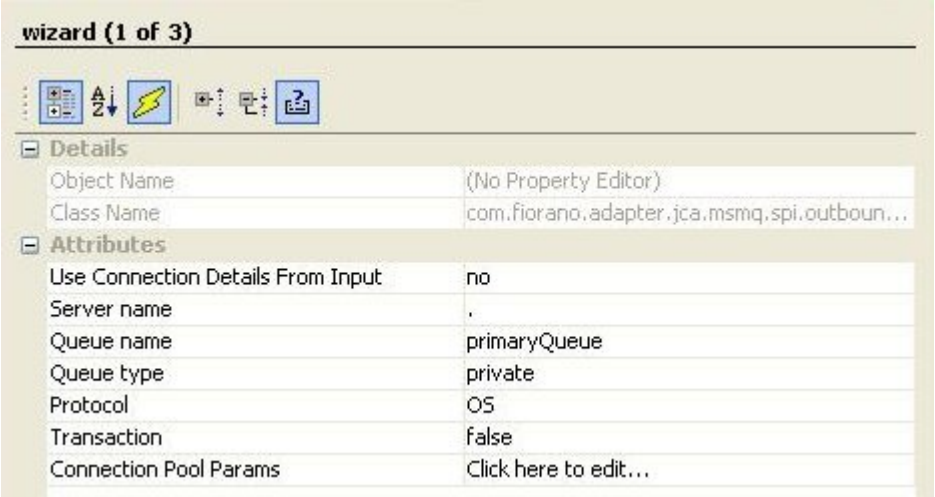


Figure 4.361: Sample MSMQ server configuration

Use Connection Details From Input: Parameters to create the connection can be specified in the input message when this property is set to true. If this property is selected the validation errors in the managed connection factory panel of the CPS are treated as warnings. So user can bypass this step without giving valid configuration and complete the configuration of the component. If valid properties are not provided even in the input message exception will be thrown at runtime.

4.7.11.1.1 Using Named Configurations

Changes are done in eStudio w.r.t to Named Configurations. With the implementation of Named Configurations MCF panel is changed as shown in Figure below.

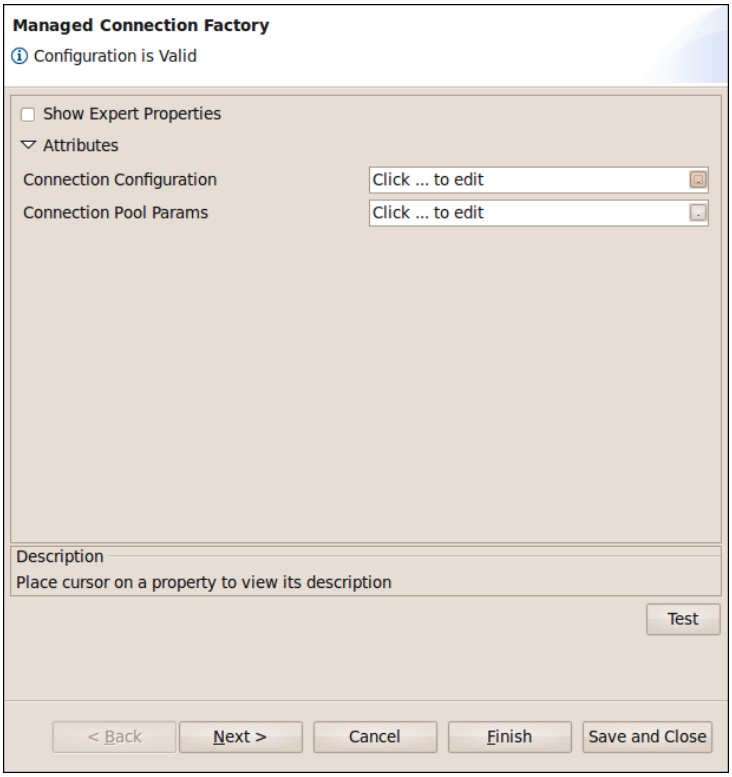


Figure: Modified Managed Connection Factory

Some properties are grouped as one property in the modified configuration panel. Server name, Queue Name, Queue type, Protocol and Transaction which were present previously in the MCF panel are grouped under the name Connection Configuration.

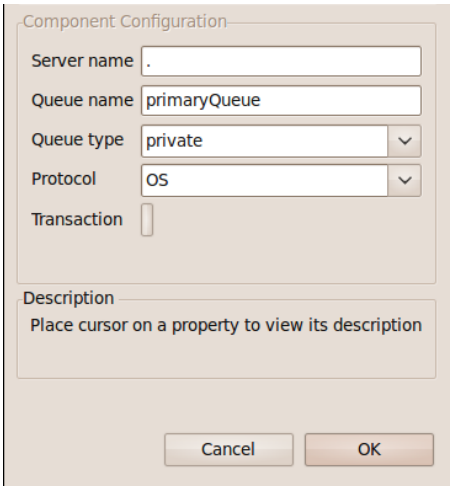


Figure: Connection Configuration

Server connection can be tested from within the CPS by clicking on **test** in the connection properties panel.

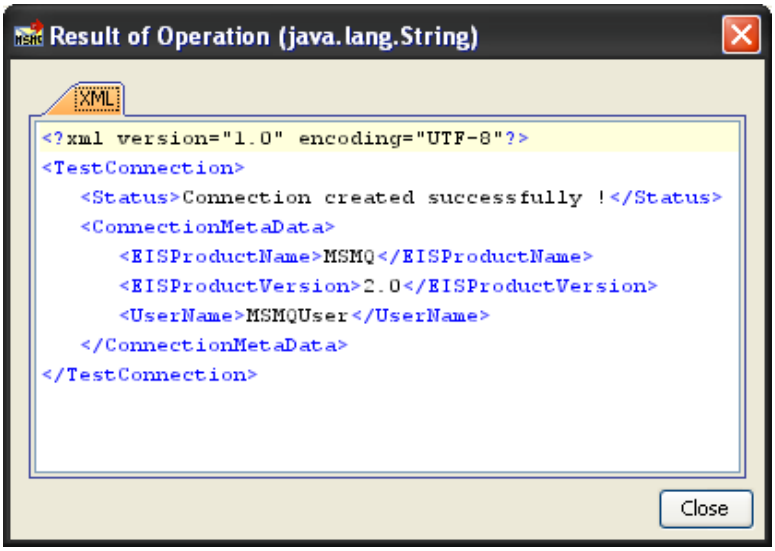


Figure 4.362: Sample connection test result indicating success

No specific information is required to be captured in Interaction properties panel.

The configuration can be tested by sending a test message when you click on the **Test** option in the interaction properties panel.

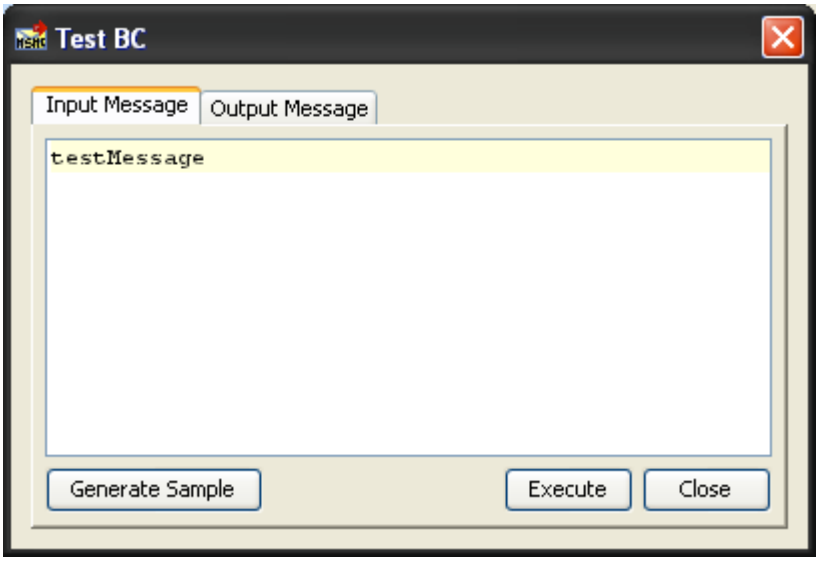


Figure 4.363: Sample input

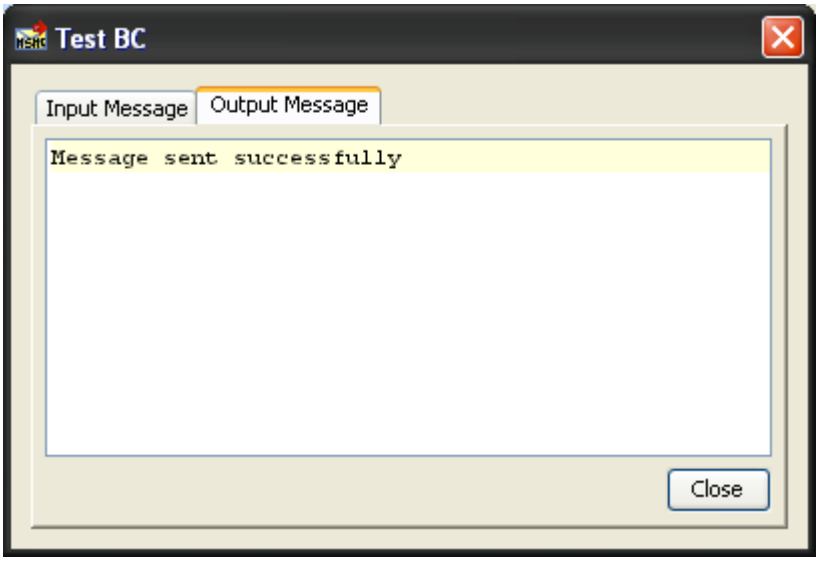


Figure 4.364: Sample response

4.7.11.2 Input and Output

4.7.11.2.1 Input Schema

There is no input schema for this adapter.

When the property Use Connection details from input is chosen, the following input schema with the element ConnectionFactorySettings, is generated. Properties that are used to create the connection are present under this element.

XML Item	Value Type
<ns1:MSMQRequest>	
<ConnectionFactorySettings>?	
<ConnectionProperties>?	
<ServerName>?	#string
<QueueName>?	#string
<QueueType>?	public | private
<Protocol>?	OS | HTTP | HTTPS | TCP | SPX
<Transaction>?	true | false
<Data>?	#string

Figure: Input schema with ConnectionFactorySettings

4.7.11.2.2 Output Schema

There is no output schema for this adapter.

4.7.11.3 Functional Demonstration

4.7.11.3.1 Scenario 1

Sending messages to a local MSMQ Server.

Configure the MSMQ Sender as described in *Configuration and Testing section* and use feeder and display component to send sample input and check the response respectively.

Figure 4.365: Feeder and Display Component Sample Input

Sample Input

```
test message
```

Figure 4.366: Sample input

Sample Output

```
Message sent successfully
```

Figure 4.367: Sample Output

4.7.11.4 Use case scenario

In the purchasing system example the record purchase details are sent to an MSMQ server from where they are picked up by other applications for processing.

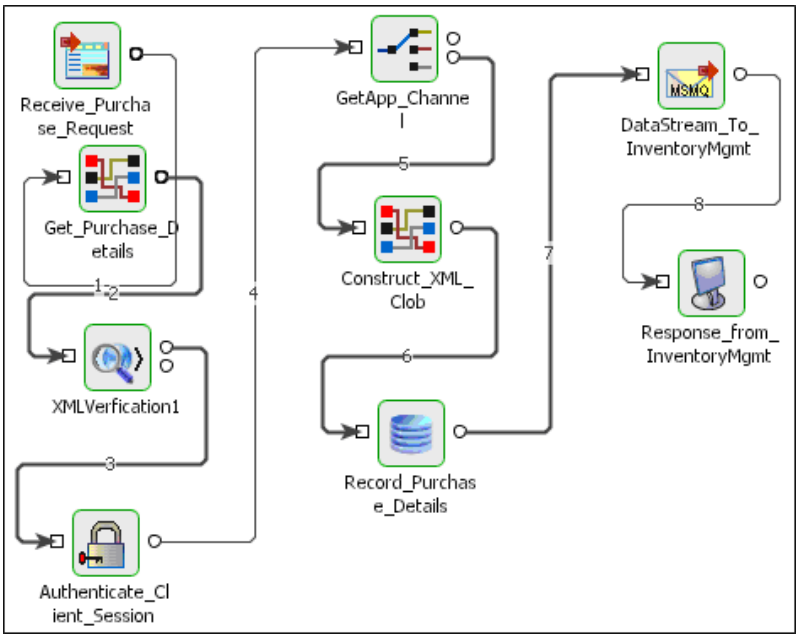


Figure 4.368: Purchasing System

The event process demonstrating this scenario is bundled with the installer. The bundled process shows it as a HTTP component instead of a MSMQ Sender component.

Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Studio.

4.7.11.5 Useful Tips

- If a queue specified in the CPS does not exist in a local or remote MSMQ server, it is not automatically created.
- This component runs only on Windows Platform.

4.7.12 TibcoRVIn

The TibcoRVIn component is used to send messages to the messaging queue server of TIBCO Rendezvous. The name of the queue to which a message needs to be sent can be specified using the CPS.

Points to note

The following required files need to be copied from the TIBCO installation directory for the component to run on windows platform. All these should be added as resources Tibrv system library.

- tibrvj.jar

- libeay32.dll
- ssleay32.dll
- TIBCO.Rendezvous.dll
- tibrv.dll
- tibrvcm.dll
- tibrvcmq.dll
- tibrvcom.dll
- tibrvft.dll
- tibrvjsd.dll
- tibrvj.dll
- tibrvsd.dll
- tibrvsdcom.dll

4.7.13 TibcoRVOut

The TibcoRVOut component is used to receive messages from the messaging queue server of TIBCO Rendezvous. The name of the queue from which a message needs to be retrieved can be specified using the CPS.

Points to note

The following required files need to be copied from the TIBCO installation directory for the component to run on windows platform. All these should be added as resources Tibrv system library.

- tibrvj.jar
- libeay32.dll
- ssleay32.dll
- TIBCO.Rendezvous.dll
- tibrv.dll
- tibrvcm.dll
- tibrvcmq.dll
- tibrvcom.dll
- tibrvft.dll
- tibrvjsd.dll
- tibrvj.dll
- tibrvsd.dll
- tibrvsdcom.dll

4.8 Performance

The Performance category consists of components like Receiver and Sender. The following section describes each component.

4.8.1 Receiver

Receiver component is used to consume JMS messages (on its input port) to measure the performance. The rate at which the messages are consumed depends upon the message size, number of connections, sessions, producers, so on. Please check the logs to see the performance. User can configure the component parameters as Runtime Arguments.

The component stops after receiving all the messages.

Points to note

- NumConsumers should be >= NumSessions and NumSessions should be >= NumConnections, otherwise it is waste of resources. Consumers are uniformly distributed over Sessions and Sessions over Connections.
- The component automatically stops once all the messages are received.

4.8.1.1 Configuration and Testing

The Receiver component doesn't have a Custom Property Sheet. It accepts the following parameters as runtime arguments.

- **Total message count** - Number of messages to be published on the output port.
- **Is Transacted** - Specify whether the session is transacted.
- **Transaction size** - Specify the number of messages to be transacted at a time.
- **Selectors** – Message Selector if any (This selector is used in creating the consumer object).
- **Number of connections** - Number of connections to be created.
- **Number of sessions** - Number of sessions to be created.
- **Number of Consumers** - Number of producers to be created.
- **Sleep time** – Sleep time till all messages are received.

These runtime arguments can be configured from Receiver Properties.

From the Studio, click on the component and you can see the properties window at the right side. Check the below screenshot for runtime arguments.

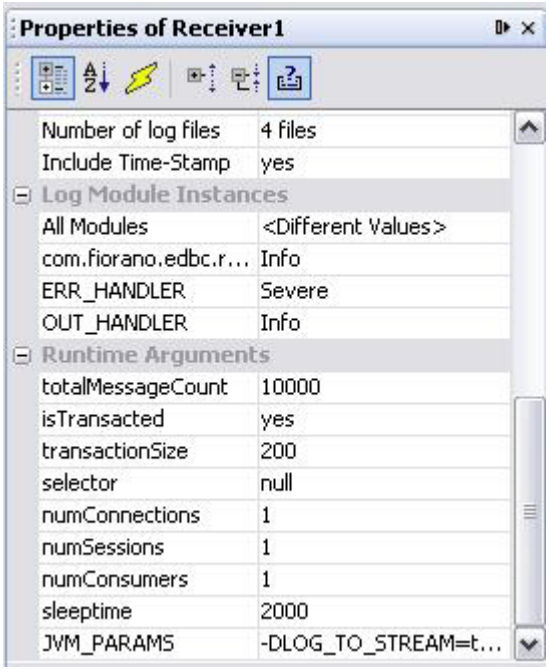


Figure 4.369: Screenshot showing the Receiver properties

4.8.1.2 Functional Demonstration

4.8.1.2.1 Scenario 1

Scenario demonstration of Receiver which is configured to receive 1000 messages.

Configure the Receiver as described in section Configuration and Testing and use a Sender component to send the input messages to the Receiver. Both the sender and receiver are configured for 1000 messages.

Figure 4.370: Scenario demonstration showing the performance numbers

4.8.1.3 Useful Tips

NumConsumers should be >= NumSessions and NumSessions should be >= NumConnections, otherwise it is waste of resources. Consumers are uniformly distributed over Sessions and Sessions over Connections.

The component automatically stops once all the messages are received.

4.8.2 Sender

Sender component is used to publish JMS messages (on its output port) to measure the performance. The rate at which the messages are published depends upon the message size, number of connections, sessions, producers, and so on. Please check the logs to see the performance. User can configure the component parameters as Runtime Arguments.

The component stops after sending all the messages.

Points to note

- NumProducers should be >= NumSessions and NumSessions should be >= NumConnections, otherwise it is waste of resources. Producers are uniformly distributed over Sessions and Sessions over Connections.
- The component automatically stops once all the messages are sent.

4.8.2.1 Configuration and Testing

The Sender component doesn't have a Custom Property Sheet. It accepts the following parameters as runtime arguments.

- **Total message count** - Number of messages to be published on the output port.
- **Is Transacted** - Specify whether the session is transacted.
- **Transaction size** - Specify the number of messages to be transacted at a time.
- **Message size** - Size of the message (in Bytes) to be published. Default message is sent in case xml file path is not provided.
- **Xml file path** - Location of the xml file to be sent as the message content. If the path is not provided, then the component sends the default XML message to its output port.
- **Number of connections** - Number of connections to be created.
- **Number of sessions** - Number of sessions to be created.
- **Number of Producers** - Number of producers to be created.

These runtime arguments can be configured from **Sender Properties**.

From the Studio, click on the component and you can see the properties window at the right side. Check the below screenshot for runtime arguments.

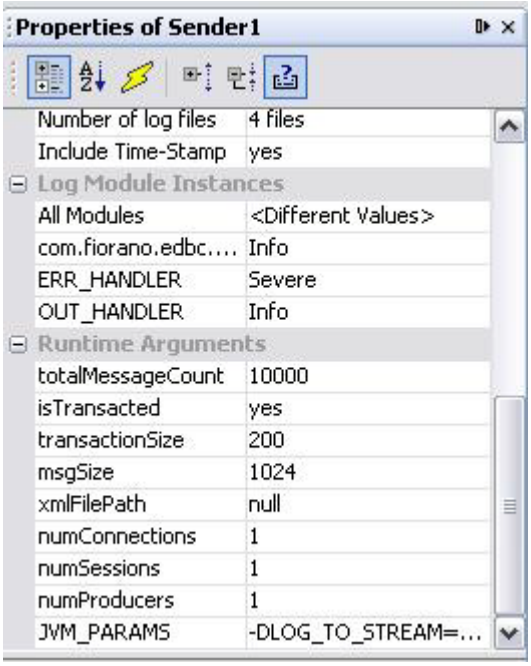


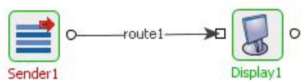
Figure 4.371: Screenshot showing the Sender properties

4.8.2.2 Functional Demonstration

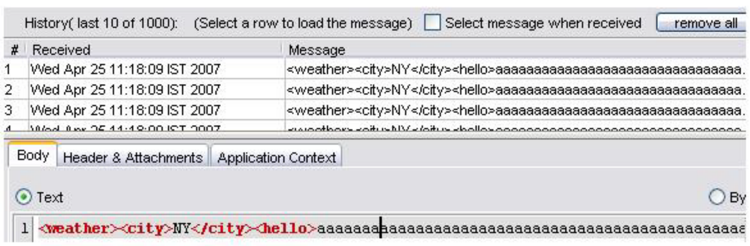
4.8.2.2.1 Scenario 1

Scenario demonstration of Sender which is configured to send 1000 messages. Sender by default sends an XML message if we don't provide the xmlFilePath runtime argument.

Configure the Sender as described in *Configuration and Testing* section to send 1000 messages and use a display component to check the response respectively.



Sample Output



Sender Logs Showing performance numbers

```
25-04-2007, 11:18:01 : INFO : Configuration Parameters.
25-04-2007, 11:18:01 : INFO : 1. Total Messages = 1,000
25-04-2007, 11:18:01 : INFO : 2. Transaction Size = 200
25-04-2007, 11:18:01 : INFO : 3. Message size = 1,024 Bytes
25-04-2007, 11:18:01 : INFO : Initialized Sender
25-04-2007, 11:18:01 : INFO : Sent 200 messages.
25-04-2007, 11:18:01 : INFO : Sent 400 messages.
25-04-2007, 11:18:01 : INFO : Sent 600 messages.
25-04-2007, 11:18:01 : INFO : Sent 800 messages.
25-04-2007, 11:18:01 : INFO : Sent 1,000 messages.
25-04-2007, 11:18:01 : INFO : Sent all 1,000 messages.
25-04-2007, 11:18:01 : INFO : publish rate for producer 0 = 2,288.33 messages per sec.
25-04-2007, 11:18:01 : INFO : Average publish rate = 2,288.33 messages per second
```

Figure 4.372: Scenario demonstration showing sample output and performance numbers

4.8.2.3 Useful Tips

NumProducers should be >= NumSessions and NumSessions should be >= NumConnections, otherwise it is waste of resources. Producers are uniformly distributed over Sessions and Sessions over Connections.

The component automatically stops once all the messages are sent.

4.9 Samples

The Samples category consists of components like BinaryFileReader, CRM, CompositeBC, LDAPLookup, LDAPAuthenticator, MarketPricesGui, Prices, RfqManager, TradeBus, and ERP. The following section describes each component.

4.9.1 Binary File Reader

BinaryFileReader reads the binary content which is provided as input and converts it into xml message. The message contains CRC info and other details.

Note: The source code for this component is available with the installer.

4.9.2 CRM

CRM is a simulator for Clarify Management Adapter. Its output is a purchase order in xml format. Username and password can be set using CPS. The purchase order details can be modified using the runtime UI.

Points to note

- The source code for this component is available with the installer.
- This component cannot be launched in-memory of the peer server.

4.9.3 Composite BC

The CompositeBC component is an EDBC component which enables you to execute more than one BC component programmatically. The sample CompositeBC is configured for the HTTP and SMTP (both are BC) components to execute in a linear sequence. The message request on the input port of CompositeBC component is fed as input to the HTTP component. The output of HTTP component is provided as input to the SMTP component (with some modifications) and the output of the SMTP component is put on the response port of the CompositeBC. Similarly, one can modify the CompositeBC component to execute 'N' number of BC components.

Note: The source code for this component is available with the installer.

4.9.4 LDAP Lookup

The LDAPLookup component enables the lookup of information organized in a directory-like fashion on a Lightweight Directory Access Protocol (LDAP) server. This information could be encryption certificates, pointers to printers and other services on a network, and provide a single logon facility where one password for a user is shared between many services.

Points to note

- In case of Authentication/Lookup/Binding failure, messages are sent to the output port with the appropriate messages like Authentication failed/Lookup failed and so on... No message comes out onto the Error port.
- In the Lookup operation, when the user enters the Root node (in CPS), the substring starting with 'dc' is checked against the substring starting with 'dc' of the string 'SECURITY_PRINCIPAL' specified in Managed Connection Factory panel. In case of mismatch, an appropriate error message is shown. If it matches, the Base node and Filter is cleared.
- In the Bind operation, adding new attributes/ adding multiple values to an existing attribute can be achieved with the help of the attribute 'AdditionalAttribute'. Always make sure that 'cn' (at least one, if you are giving multiple 'cn's) holds the value of 'cn' given in 'dn'. Also make sure that 'sn' is provided if the value of 'objectClass' is 'person'. One can add multiple users at a time also.
- The source code for this component is available with the installer.

4.9.5 LDAP Authenticator

The LdapAuthenticator is used to authenticate against an LDAP server. It's a light weight component which only does authentication. It does not do lookup or bind.

Note: The source code for this component is available with the installer.

4.9.6 Market Prices GUI

This component is used for the Bond Trading Demo Sample where the user can view the list of stock quotes which changing prices. The user can raise a RFQ (request for quote) for any listed stock for a specified number. The runtime UI helps the user to track the price changes and the raise requests.

Points to note

- The source code for this component is available with the installer.
- This component cannot be launched in-memory of the peer server.

4.9.7 Prices

The Prices adapter is used to generate the price for a bond.

Note: The source code for this component is available with the installer.

4.9.7.1 Configuration and Testing

Component is not configurable.

4.9.7.2 Input and Output

4.9.7.2.1Input Schema

Schema Element	Description
<ISIN>	International Securities Identification Numbering. This is the unique identifier of the bond.
<ISSUER>	The issuer of the bond.
<CURRENCY>	Currency of the bond.
<MATURITY_RANGE_FROM>	Range for the maturity value.
<MATURITY_RANGE_TO>	
COUPON_RANGE_MIN	Range for the coupon value.
COUPON_RANGE_MAX	
CREDITRATING_RANGE_FROM	Range for credit rating value.
CREDITRATING_RANGE_TO	
CREDITRATING_AGENCY	Agency for credit rating.

4.9.7.2.2 Output Schema

Schema Element	Description
<ISIN>	International Securities Identification Numbering. This is the unique identifier of the bond.
<BID_PRICE>	Maximum price a buyer is willing to pay for a bond.
<ASK_PRICE>	Minimum price the seller is willing to accept for a bond.
<TIME_STAMP>	The time at which the prices are generated.

4.9.7.3 Use Case scenario

In a bond trading scenario, the price is generated using Prices adapter.

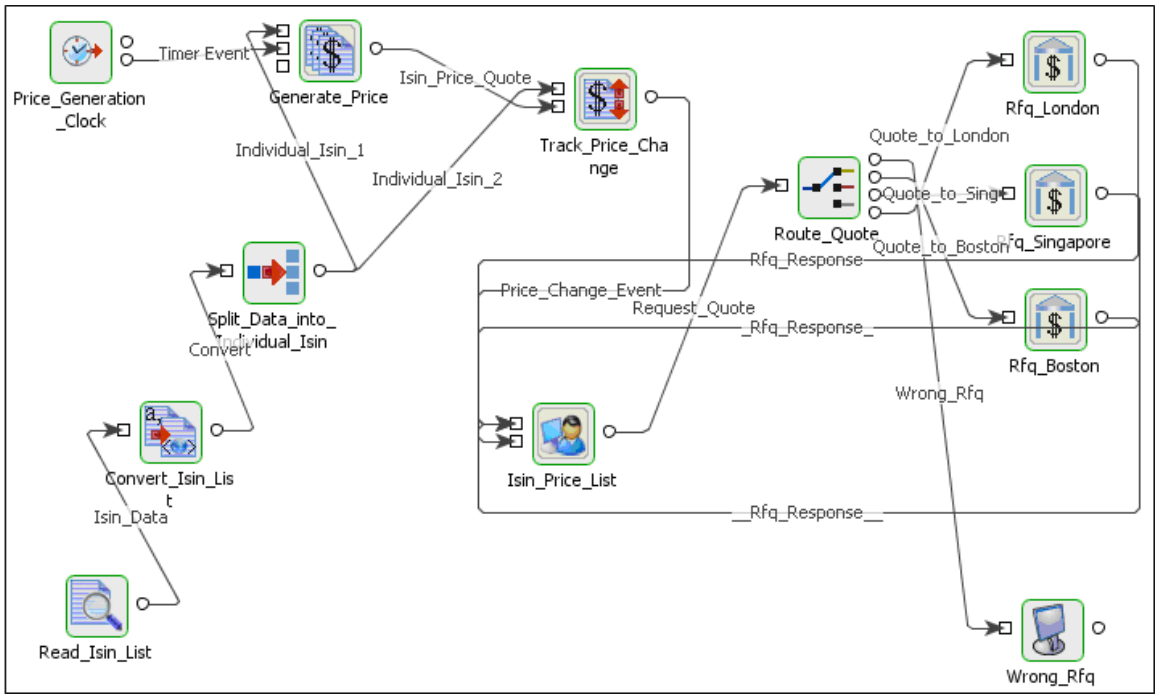


Figure 4.373: Bond Trading Scenario

The event process demonstrating this scenario is bundled with the installer. Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Studio.

4.9.8 RFQ Manager

This component is used for the Bond Trading Demo Sample to respond the quoted price for the request for quotes received by this component. It provides a runtime UI for the stock exchange to send the quotation with price back to the requested user.

Points to note

- The source code for this component is available with the installer.
- This component cannot be launched in-memory of the peer server.

4.9.9 Trade Bus

This component is used for the Bond Trading Demo Sample co-relates the changes in price for every stock and shows up a maintenance screen for all changes occurring for all stocks listed. The runtime UI is used to show the changes as they occur.

Points to note

- The source code for this component is available with the installer.
- This component cannot be launched in-memory of the peer server.

4.9.10 ERP

The ERP component is a simulator for an ERP System. Input to the system is a Purchase Order in XML format. Output to the system is a Rejection or Acceptance of Purchase Order on respective ports. Username and password can be set using CPS. The purchase order can be accepted or rejected using the runtime UI.

Points to note

- The source code for this component is available with the installer.
- This component cannot be launched in-memory of the peer server.

4.10 Script

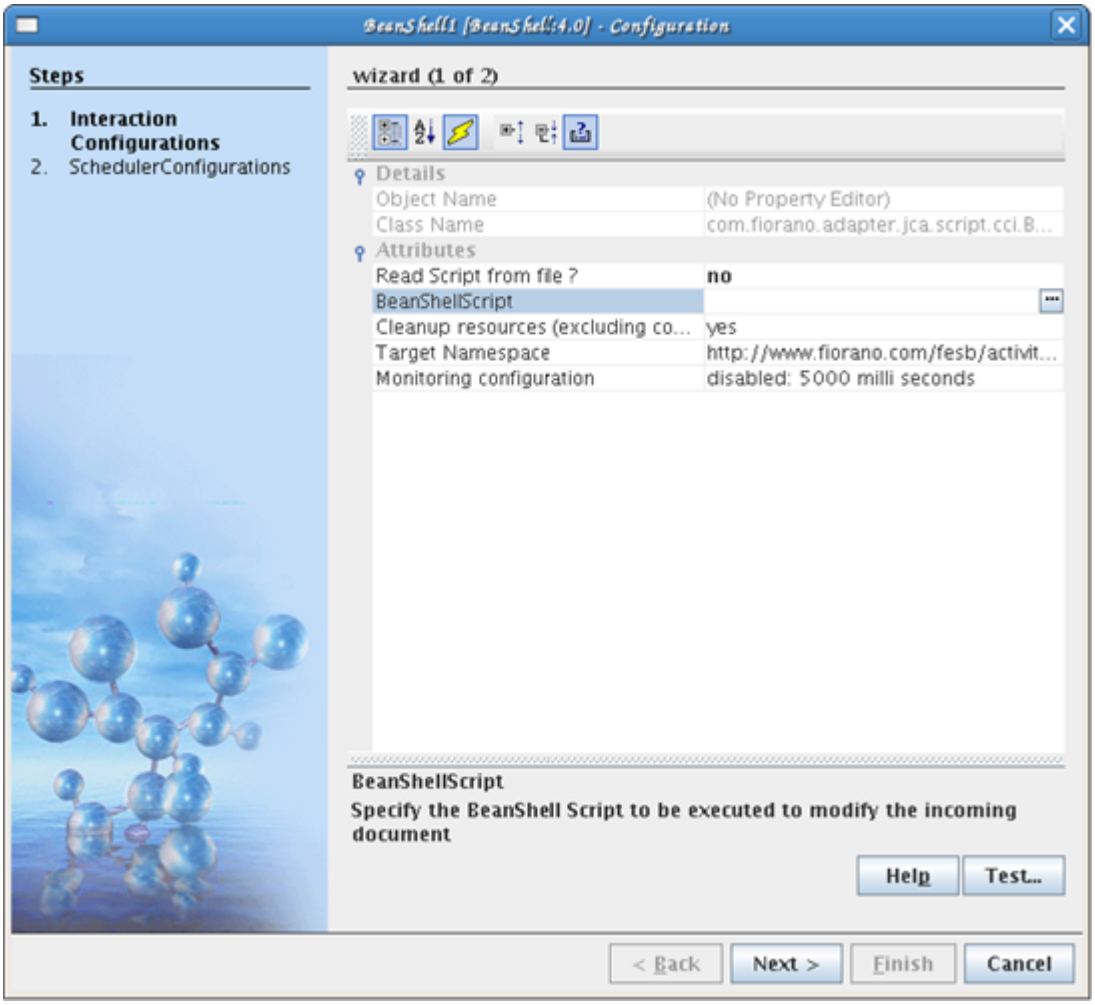
This component is used for executing BeanShell Scripts. The BeanShell Script to be executed is specified in the Custom Property Sheet (CPS). This component executes the script on documents it receives as input and returns the result.

4.10.1 BeanShell Script

This component is used for executing BeanShell scripts. The BeanShell script to be executed is specified using the CPS This component executes the script on documents it receives as input and returns the result.

Note: The component uses the document object to get the content and properties of the message. The result (return value) after executing the script is set as content of the output message.

4.10.1.1 Interaction Configuration



The following properties can be configured in the Interaction Configuration panel.

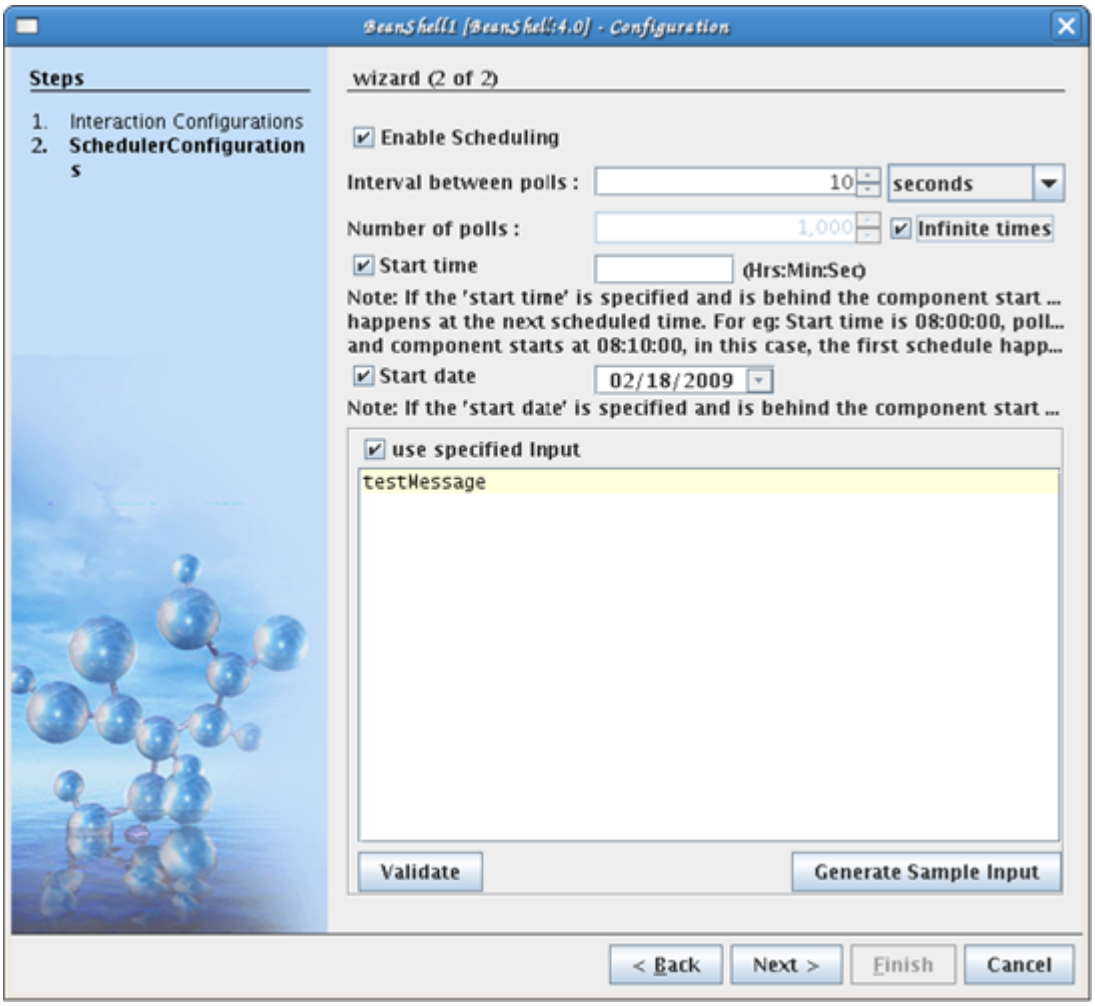
1. **Read Script from a File?** - If this attribute is set to 'Yes' then you can provide the complete path of the BeanShell Script file (file should have ".bsh" extension) which you want to execute. If it is set to 'No', then you have to specify the BeanShell Script in the CPS.
2. **BeanShell Script** - Specify the BeanShell Script to be executed to modify the incoming document.
3. **Script File Path** - Path of the BeanShell Script File which has to be executed.
4. **Cleanup resources (excluding connection) after each document** (yes / no)

Points to Note

- Input message content and properties can be accessed using document object. For example, 'document.getText()' returns input message content. To access a message property named 'TestProperty, 'document.getRecord().getProperty('TestProperty')' returns the value of the property.

- Message properties can be set using 'setProperty' method.
- Message properties which are set on input message also appear on output message unless they are changed in the script. For example, if a property named 'TestProperty' is present on input message then output message also contains the property unless it is changed in script provided.
- If there is no return value in the script, input record is sent as output message with empty message content.

SchedulerConfigurations Panel

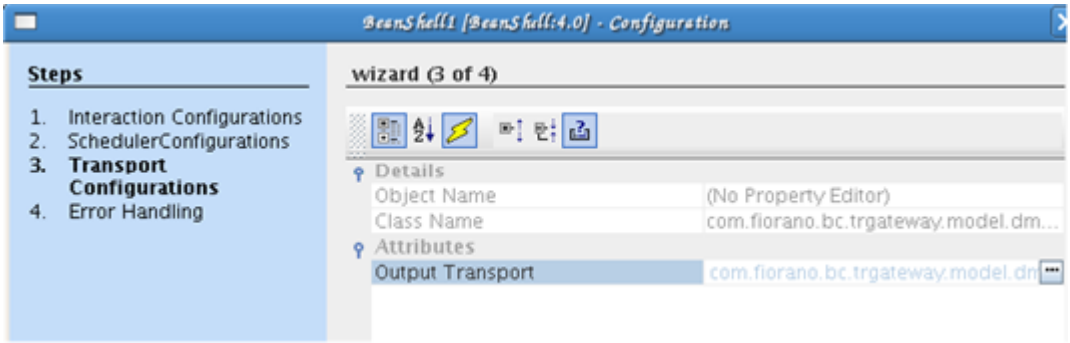


- **Intervals between polls** - Time interval between the polls
- **Number of polls** - The number of times that the input request will be sent to the input port of the adapter
- **Start time** - The polling start time. If the start time is specified and is below the component start time, the first schedule will happen at the next schedule time. For example, start time is 08:00:00, poll interval is 30 minutes, and component starts at 8:10:00, in this case the first schedule will happen at 08:30:00.
- **Start date** - The polling start date. If the start time is specified and is below the component start time then it will be ignored.

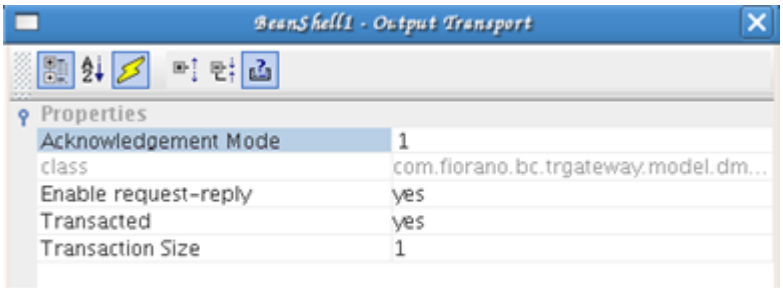
- **Use specified Input** - By enabling this property, you can specify the inputs which will be sent to the input port of the adapter.
- **Validate** - Validates the specified input with the input port xsd

Transport Configurations

Transport configurations are specified only when Scheduling is enabled.



Output Transport



Properties

- **Acknowledgement Mode** - The acknowledgement mode of the JMS session. The acknowledgement mode is set at the time the session is created. If the session is transacted, the acknowledgement mode is ignored.
- **Enable request-reply** - If "Enable request-reply" is set to "yes" then responses will be sent to "JMSReplyTo" if it is present, else response is sent on the output port.
- **Transacted** - Set "yes" if the JMS session created should be transacted, "no" otherwise.
- **Transaction Size** - Transaction size is the number of messages after which commit will be called on corresponding JMS session.

4.10.1.2 Sample Input and Output

The configuration can be tested by clicking the Test button in the interaction Configuration panel.

The below screenshots show the sample input and output for the following BeanShell script.

```
map = new HashMap();
map.put("foo", "bar");
importObject( map );
return ( get("foo") );
```

Figure 1: sample BeanShell Script

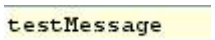


Figure 2: Sample Input Message

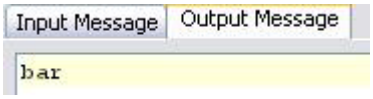


Figure 3: Response Generated

4.10.1.3 Functional Demonstration

4.10.1.3.1 Scenario 1

Execution of BeanShell Script provided.

Configure the BeanShell Script adapter as described in section Configuration and Testing and use feeder and display component to send sample input and to check the response respectively. In the example given below, the script provided is same as that of in Fig 1:

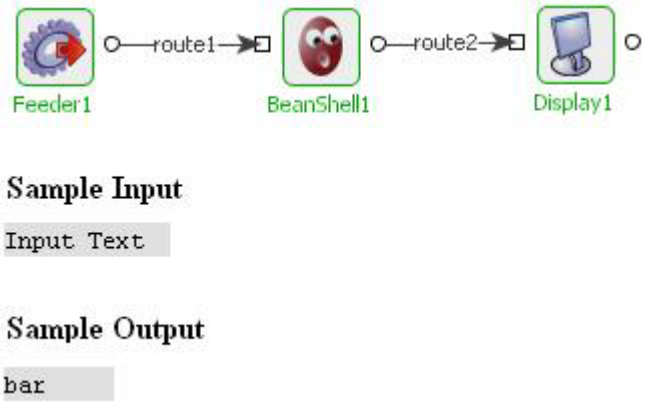


Figure 4: Scenario demonstration with sample input and output

4.10.1.3.2 Scenario 2

This sample shows accessing an input message. Configure the component with the following script.

```
return (document.getText());
```

Figure 5: sample BeanShell Script

```
testMessage
```

Figure 6: Sample Input Message

Figure 7: Response Generated

4.10.1.4 Useful Tips

The component uses the document object to get the content and properties of the message. The result (return value) after executing the script is set as output message.

4.10.2 Groovy Scrip

This component is used for executing Groovy Scripts. The Groovy Script to be executed is specified in the Custom Property Sheet (CPS). This component executes the script on documents it receives as input and returns the result.

Note: The component uses the document object to get the content and properties of the message. The result after executing of the script should be set back onto the document object.

4.10.2.1 Configuration and Testing

4.10.2.1.1 Interaction Configuration

The following properties can be configured in the Interaction Configuration panel.

- **Read Script from a File?** - If this attribute is set to 'Yes' then you can provide the complete path of the Groovy Script file (file should have ".groovy" extension) which you want to execute. If it is set to 'No', then you have to specify the Groovy Script in the CPS.
- **Groovy Script** - Specify the Groovy Script to be executed to modify the incoming document.
- **Script File Path** - Path of the Groovy Script File which has to be executed.

Points to Note

- Input message content and properties can be accessed using document object. For example, 'document.getText()' returns input message content. To access a message property named 'TestProperty, 'document.getRecord().getProperty('TestProperty')' returns the value of the property.
- Message properties can be set using 'setProperty' method.
- Message properties which are set on input message also appear on output message unless they are changed in the script. For example, if a property named 'TestProperty' is present on input message then output message also contains the property unless it is changed in script provided.
- Output message content can be set either by using 'setText(String value)' of document object or by specifying a return value in the script provided. If both present, return value is set as message content.

4.10.2.1.2 Sample Input and Output

The configuration can be tested by clicking the Test button in the interaction Configuration panel.

The below script show the sample input and output for the following groovy script.

```
def foo(list, value) {
 list << value
 list << 2
 list << 3
 list << 4
 return list
 }
 x = []
 foo(x, 1)
```

Figure 4.374: sample Groovy Script

```
testMessage
```

Figure 4.375: Sample Input Message

Input Message | Output Message

```
[1, 2, 3, 4]
```

Figure 4.376: Response Generated

4.10.2.2 Functional Demonstration

4.10.2.2.1 Scenario 1

Execution of Groovy Script provided.

Configure the Groovy Script adapter as described in Configuration and Testing section and use feeder and display component to send sample input and to check the response respectively. In the example given below, the script provided is same as that of in Figure 4.377.

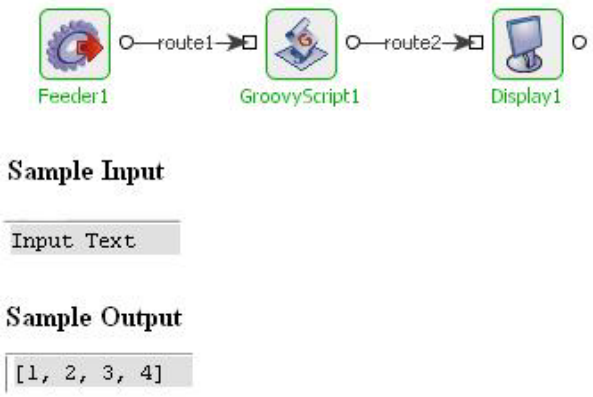


Figure 4.377: Scenario demonstration with sample input and output

4.10.2.2.2 Scenario 2

This sample shows accessing an input message. Configure the component with the following script.

```
return (document.getText());
```

Figure 5: sample Groovy Script

```
testMessage
```

Figure 6: Sample Input Message

Figure 7: Response Generated

4.10.2.3 Useful Tips

The component uses the document object to get the content and properties of the message. The result after executing of the script should be set back onto the document object.

4.10.3 Java Script

This component is used for executing JavaScript. The JavaScript to be executed is specified using the Custom Property Sheet (CPS). This component executes the script on documents it receives as input and returns the result.

The component uses bsf API to evaluate the script.

Note: The component uses the document object to get the content and properties of the message. The result after executing of the script should be set back onto the document object.

4.10.3.1 Configuration and Testing

4.10.3.1.1 Interaction Configuration

In the interaction configuration panel, the attribute Read Script from a File? can be configured. If the attribute is set to Yes then you can provide the complete path of the Java Script file (file should have .js extension) which you want to execute.

If Read Script from a File? is set to No, then you have to specify the Java Script in the CPS.

Points to Note

- Input message content and properties can be accessed using document object. For example, 'document.getText()' returns input message content. To access a message property named 'TestProperty, 'document.getRecord().getProperty('TestProperty')' returns the value of the property.
- Message properties can be set using 'setProperty' method.
- Message properties which are set on input message also appear on output message unless they are changed in the script. For example, if a property named 'TestProperty' is present on input message then output message also contains the property unless it is changed in script provided.
- Output message content can be set either by using 'setText(String value)' of document object.

4.10.3.1.2 Sample Input and Output

The configuration can be tested by clicking the Test button in the interaction Configuration panel.

The below script show the sample input and output for the following java script.

```
var time = 15;
if (time < 10)
{
document.setText("Good morning")
}
else
{
document.setText("Good Day")
```

```
}
```

```
testMessage
```

Figure 4.378: Sample Input Message

```
Good Day
```

Figure 4.379: Response Generated

4.10.3.2 Functional Demonstration

4.10.3.2.1 Scenario 1

Execution of Java Script provided.

Configure the Java Script as described in Configuration and Testing section and use feeder and display component to send sample input and to check the response respectively. In the example given below, the script provided is:

```
var r=Math.random()
if (r>0.5)
{
document.setText("<a href='http://www.w3schools.com'>Learn Web Development!</a>")
}
else
{
document.setText("<a href='http://www.refsnesdata.no'>Visit Refsnes Data!</a>")
}
```

Sample Input

```
testmessage
```

Sample Output

```
<a href='http://www.w3schools.com'>Learn Web Development!</a>
```

Figure 4.380: Scenario demonstration with sample input and output

4.10.3.3 Useful Tips

The component uses the document object to get the content and properties of the message. The result after executing of the script should be set back onto the document object.

4.10.4 Perl Script

This component is used for executing Perl Script. The Perl Script to be executed is specified using the Custom Property Sheet (CPS). This component executes the script on documents it receives as input and returns the result.

Points to note

- The input message to the PerlScript component is provided as a command line argument to the Perl script to be executed. If the input message contains white spaces, then please provide the message in courses if the whole message is required in one argument.

4.10.4.1 Configuration and Testing

4.10.4.1.1 Interaction Configuration

The following properties can be configured in the Interaction Configuration panel.

- Read Script from a File? - If this attribute is set to 'Yes' then you can provide the complete path of the Perl Script file (file should have ".pl" extension) which you want to execute. If it is set to 'No', then you have to specify the Perl Script in the CPS.
- Perl Script - Specify the Perl Script to be executed to modify the incoming document.
- Script File Path - Path of the Perl Script File which has to be executed.
- Perl Executable Path (Optional) - If you have installed Perl on your local system, you can provide the Perl executable path. If you don't provide any path here, then the default executable path is used.

4.10.4.1.2 Sample Input and Output

The configuration can be tested by clicking the **Test** button in the interaction Configuration panel.

The below script show the sample input and output for the following Perl script.

```
$top_number = 100;
$x = 1;
$total = 0;
while ( $x <= $top_number ) {
                        $total = $total + $x;    # short form: $total += $x;
                        $x += 1;          # do you follow this short form?
}
```

```
print "The total from 1 to $top_number is $total\n";
```

Figure 4.381: Sample Perl Script

4.10.4.2 Using Named Configurations

Changes are done in eStudio wrt to Named Configurations. The property Perl Executable Path which was previously present in MCF panel is modified as Perl Configuration, which can use Named Configurations as shown in Figure below.

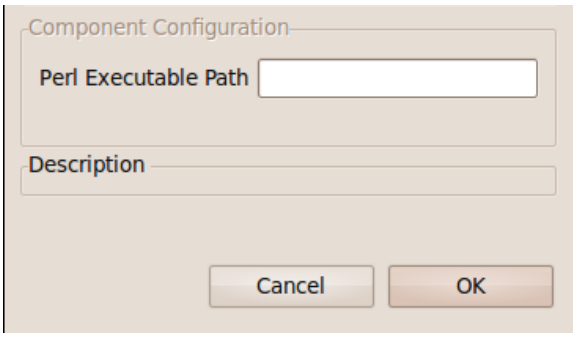


Figure: Perl Configuration

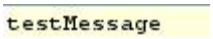


Figure 4.382: Sample Input Message

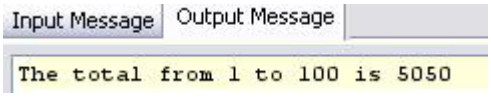


Figure 4.383: Response Generated

4.10.4.3 Functional Demonstration

4.10.4.3.1 Scenario 1

Execution of Perl Script provided.

Configure the Perl Script adapter as described in Configuration and Testing section and use feeder and display component to send sample input and to check the response respectively. In the example given below, the script provided is same as that of in Figure 4.384.

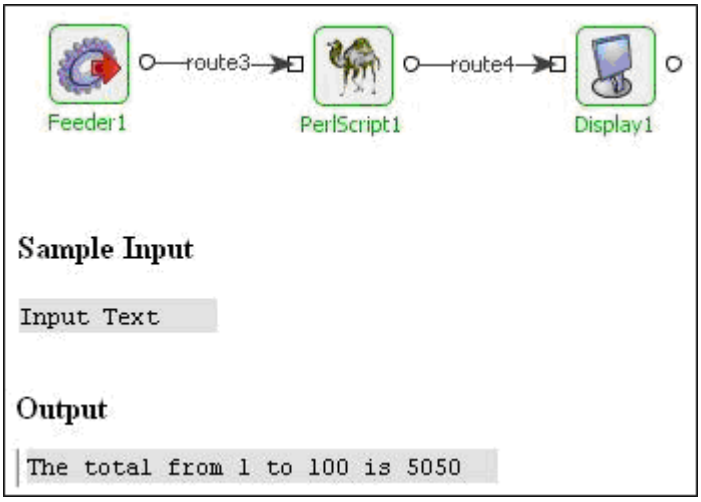


Figure 4.384: Scenario demonstration with sample input and output

4.10.4.3.2 Scenario 2

This sample shows accessing an input message. Configure the component with the following script.

```
print $ARGV[0];
```

Figure 5: Sample perl Script

```
testMessage
```

Figure 6: Sample Input Message

Figure 7: Response Generated

4.10.4.4 Useful Tips

The input message to the PerlScript component is provided as an command line argument to the Perl script to be executed. If the input message contains white spaces, then please provide the message in courses if the whole message is required in one argument.

This component can be executed on Windows platform only.

4.10.5 Python Script

This component is used for executing Python Script. The Python Script to be executed is specified using the Custom Property Sheet (CPS). This component executes the script on documents it receives as input and returns the result.

Note: The component uses the document object to get the content and properties of the message. The result after executing of the script should be set back onto the document object.

4.10.5.1 Configuration and Testing

4.10.5.1.1 Interaction Configuration

The following properties can be configured in the Interaction Configuration panel.

- Read Script from a File? - If this attribute is set to 'Yes' then you can provide the complete path of the Python Script file (file should have ".py" extension) which you want to execute. If it is set to 'No', then you have to specify the Python Script in the CPS.
- Python Script - Specify the Python Script to be executed to modify the incoming document.
- Script file path - Path of the Python Script File which has to be executed.
- Path Locations - Specifies the locations to be used in path for the imports in PythonScript.

Points to Note

- Input message content and properties can be accessed using document object. For example, 'document.getText()' returns input message content. To access a message property named 'TestProperty, 'document.getRecord().getProperty('TestProperty')' returns the value of the property.
- Message properties can be set using 'setProperty' method.
- Message properties which are set on input message also appear on output message unless they are changed in the script. For example, if a property named 'TestProperty' is present on input message then output message also contains the property unless it is changed in script provided.
- Output message content can be set either by using 'setText(String value)' of document object or by specifying a return value (using 'print') in the script provided. If both present, return value is set as message content.

4.10.5.1.2 Sample Input and Output

The configuration can be tested by clicking the **Test** button in the interaction Configuration panel.

The below screenshots show the sample input and output for the following Python script.

```
width = 20;
height = 5*9;
```

```
print width * height;
```

Figure 4.385: Sample Python Script

```
testMessage
```

Figure 4.386: Sample Input Message

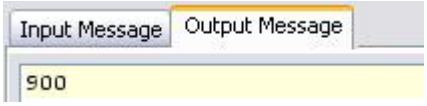


Figure 4.386: Response Generated

4.10.5.2 Functional Demonstration

4.10.5.2.1 Scenario 1

Execution of Python Script provided.

Configure the Python Script adapter as described in Configuration and Testing section and use feeder and display component to send sample input and to check the response respectively. In the example given below, the script provided is same as that of in Figure 4.387.

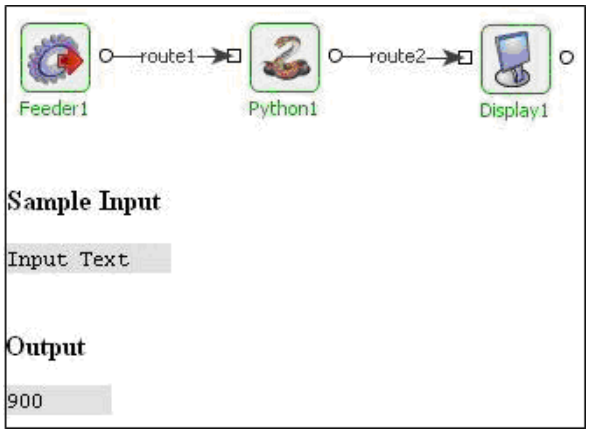


Figure 4.387: Scenario demonstration with sample input and output

4.10.5.2.2 Scenario 2

This sample shows accessing an input message. Configure the component with the following script.

```
print (document.getText());
```

Figure 5: sample Groovy Script

```
testMessage
```

Figure 6: Sample Input Message

Figure 7: Response Generated

4.10.5.3 Useful Tips

The component uses the document object to get the content and properties of the message. The result after executing of the script should be set back onto the document object.

4.11 Transformation

The Transformation category consists of components like EDI2XML, HL7Reader, HL7Writer, Text2XML, XML2EDI, XML2PDF, XML2Text, and XSLT. The following section describes each component.

4.11.1 EDI 2 XML

The EDI2XML component is used for transforming information from EDI format to XML format. This business component accepts data in EDI format and transforms it to the required XML format.

Note: The component takes EFL file as input in the CPS which describes the conversion rules. EFL files can be created or modified using the Fiorano Studio tool.

4.11.1.1 Configuration and Testing

The EDI2XML component can be configured using its Custom Proper Sheet wizard. Following is the Interaction properties panel.

Attributes	
Description	null
FunctionName	edi2xml
Execution Timeout	0 milli seconds
Error Handling	click ... to do ErrorHandling configurations
Validate Input	no
Close After Each Request	yes
Target Namespace	http://www.fiorano.com/fesb/activity/EDI2...

Figure 4.388: Sample EDI2XML configuration

Above can be tested from within the CPS by clicking on **test** button in the CPS panel.

Sample EDI format schema to be provided in EDI2XML component.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Message
```

```
PUBLIC "-//mendelson.de//DTD for m-e-c eagle//EN" "http://www.fiorano.com/dtds/m-e-
c.dtd">
<Message standard="EDIFACT" version="93" release="A" enableMissValue="0" name="Empty"
minRepeat="0" maxRepeat="1" hideElement="0">
 <Segment id="ABC" delimiter="~" name="segment" description="" minRepeat="0"
maxRepeat="1" hideElement="0">
<DataElement type="AN" default="ABC" minLength="3" maxLength="3" name="ABC"
description="" minRepeat="0" maxRepeat="1" hideElement="0"/>
 </Segment>
</Message>
```

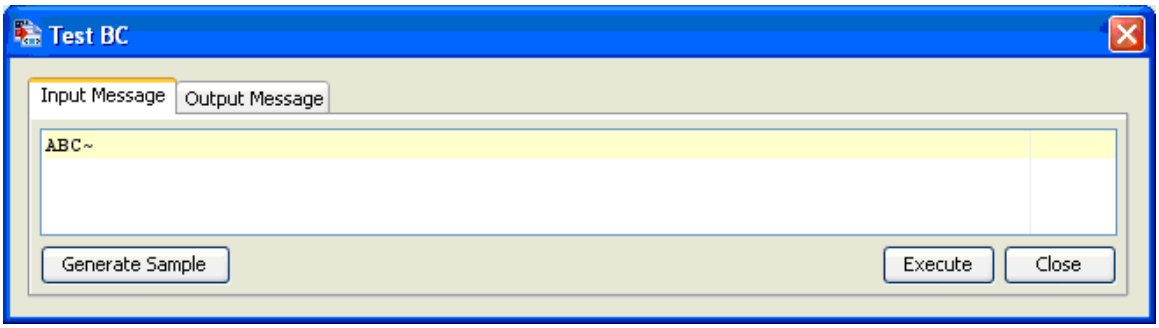
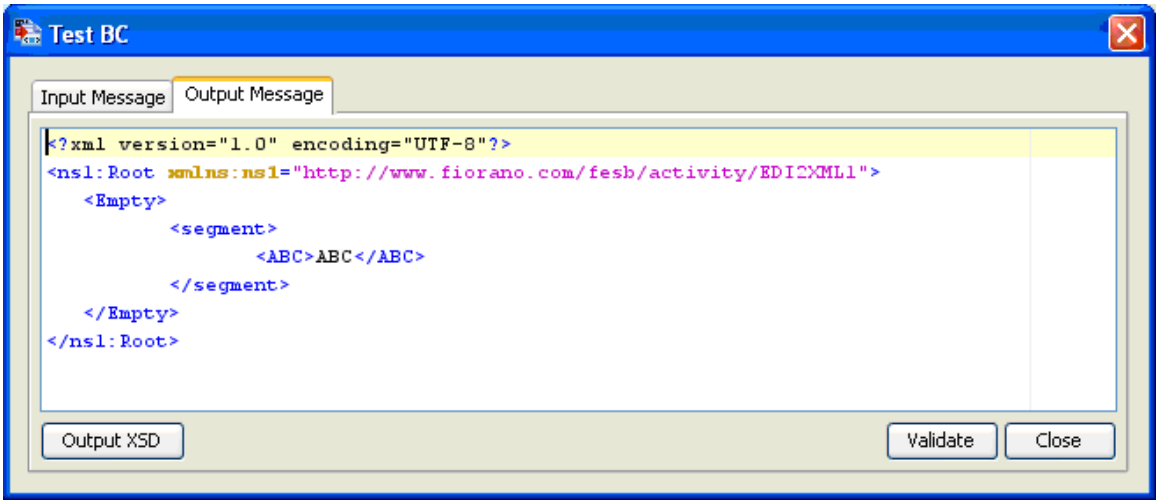


Figure 4.398: Sample EDI2XML input message


Test BC
Input Message
Output Message
<?xml version="1.0" encoding="UTF-8"?>
<ns1:Root xmlns:ns1="http://www.fiorano.com/fesb/activity/EDI2XML1">
<Empty>

<ABC>ABC</ABC>

</Empty>
</ns1:Root>
Output XSD
Validate
Close

Figure 4.399: Sample EDI2XML output message

4.11.1.2 Functional Demonstration

4.11.1.2.1 Scenario 1

Send a message in EDI format as defined in the *Configuration and Testing* section and displaying the output XML message.

Configure the EDI2XML as described in *Configuration and Testing* section and use feeder and display component to send sample input and check the response respectively.

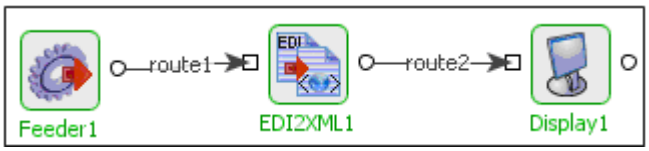


Figure 4.400: Demonstrating Scenario 1 with sample input and output

Sample Input:

ABC

Sample Output:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:Root xmlns:ns1="http://www.fiorano.com/fesb/activity/EDI2XML1">
 <Empty>
<segment>
 <ABC>ABC</ABC>
</segment>
 </Empty>
</ns1:Root>
```

4.11.2 HL7 Reader

The HL7Reader component is used to parse through documents in HL7 (Health Level Seven) format. HL7 is a standard to exchange management and integration of data that supports clinical patient care and the management, delivery and evaluation of healthcare components. This component is used to extract data, present in HL7 format and convert it to XML format. This XML format needs to comply with a specified XSD (XML Schema Definition).

Note: Please refer to http://www.hl7.org/ for more details.

4.11.2.1 Configuration and Testing

4.11.2.1.1 Interaction Configuration

In the interaction configuration panel, the following attributes can be configured.

Choose Schema Source - The steps for configuring the component are based on the source of schema specified for the HL7 file that needs to be parsed by this component. The source of schema can be selected from the Choose Source Schema dropdown list.

XSD Defined Here – If you select this you need to provide the schema in the Custom Property Sheet itself or you can also give the repository URL if there are any references in your XSD. For example, file:///E:\\tif\\components\\HL7Reader\\test\\schemas/v2.3.1/ORU_R01.XSD.

XSD Defined in Input Message - The contents of XSD are received from a feeder component or any other component that feeds the input to the HL7Reader component. The input contains the HL7 data to be transformed as well as the schema the output should conform to.

XSD URL Defined in Input Message - On selecting this option, the path of the XSD is received from a feeder component or any other component that feeds the input to the HL7Reader component.

XSD from Repository using Input Message Header - In this case, in addition to the XSD repository, the header format for the HL7 content can be configured.

Schema - Enter the XSD in the given text area or enter the path of the XSD (in case the XSD includes other XSDs).

Repository URL - Specifies the repository of the XSD that needs to be referred to. For example, file:///E:\\tif\\components\\HL7Reader\\test\\schemas/v2.3.1/ORU_R01.XSD.

Root Element - Specifies the name of the root tag of the XML that would be generated in compliance with the XSD provided in Schema Source.

Header format of HL7 input - Specifies the format of the header in the input data. In this case the Header format appears with a default value of MSH|^||||||XSD^VER|. MSH is a segment name that denotes the Message header. XSD^VER specifies the XSD that is being referred to, for example, ORU^R01. Based on the input HL7 data, the corresponding XSD is invoked. The first tag encountered in a HL7 file is the segment tag. MSH, in this case, the first line contains the following: (| - field delimiter, ^ - Component delimiter, ~ - repeat tag, \ - Escape delimiter, and - subcomponent delimiter).

4.11.2.1.2 Sample Input and Output

The configuration can be tested by clicking the Test button in the interaction Configuration panel.

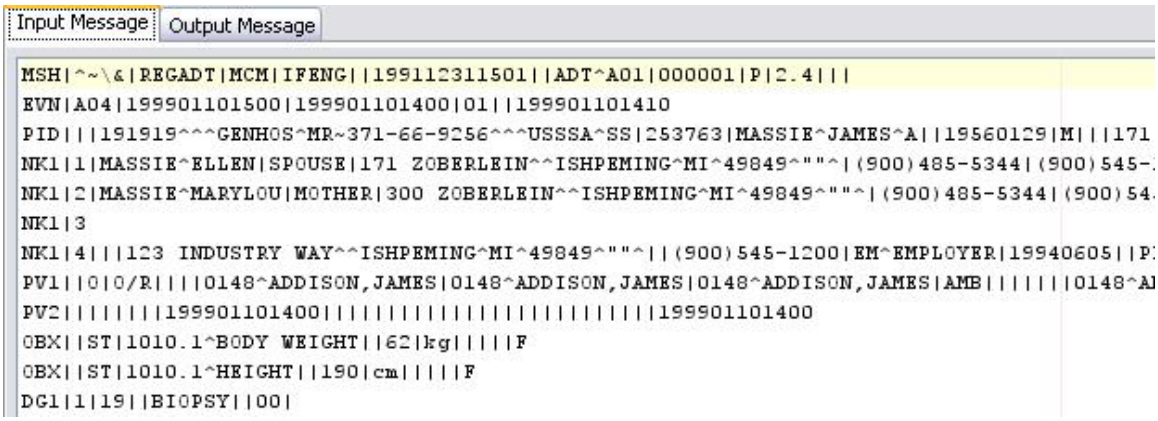


Figure 4.401: Sample Input Message

Input Message | Output Message

```
<?xml version="1.0" encoding="UTF-8"?>
<ADT_A01>
    <MSH>
        <MSH.1>|</MSH.1>
        <MSH.2>^~\&amp;</MSH.2>
        <MSH.3>REGADT</MSH.3>
        <MSH.4>MCM</MSH.4>
        <MSH.5>IFENG</MSH.5>
        <MSH.6/>
        <MSH.7>199112311501</MSH.7>
        <MSH.8/>
        <MSH.9>
            <MSG.1>ADT</MSG.1>
            <MSG.2>A01</MSG.2>
        </MSH.9>
```

Figure 4.402: Response Generated

4.11.2.2 Functional Demonstration

4.11.2.2.1 Scenario 1

This scenario explains the basic functionality of HL7Reader component.

Configure the HL7Reader as shown in the Screenshot below and use feeder and display component to send sample input and to check the response respectively. The HL7 input is sent from the feeder and the corresponding XML is generated using the schema specified in the CPS.

Schema Source	
Choose Schema Source	XSD Defined Here
Schema details	
Schema	**file:\\\D:\fioranodev\HEAD\source\b...**
Root Element	ADT_A01

Figure 4.403: Configuration used in this scenario

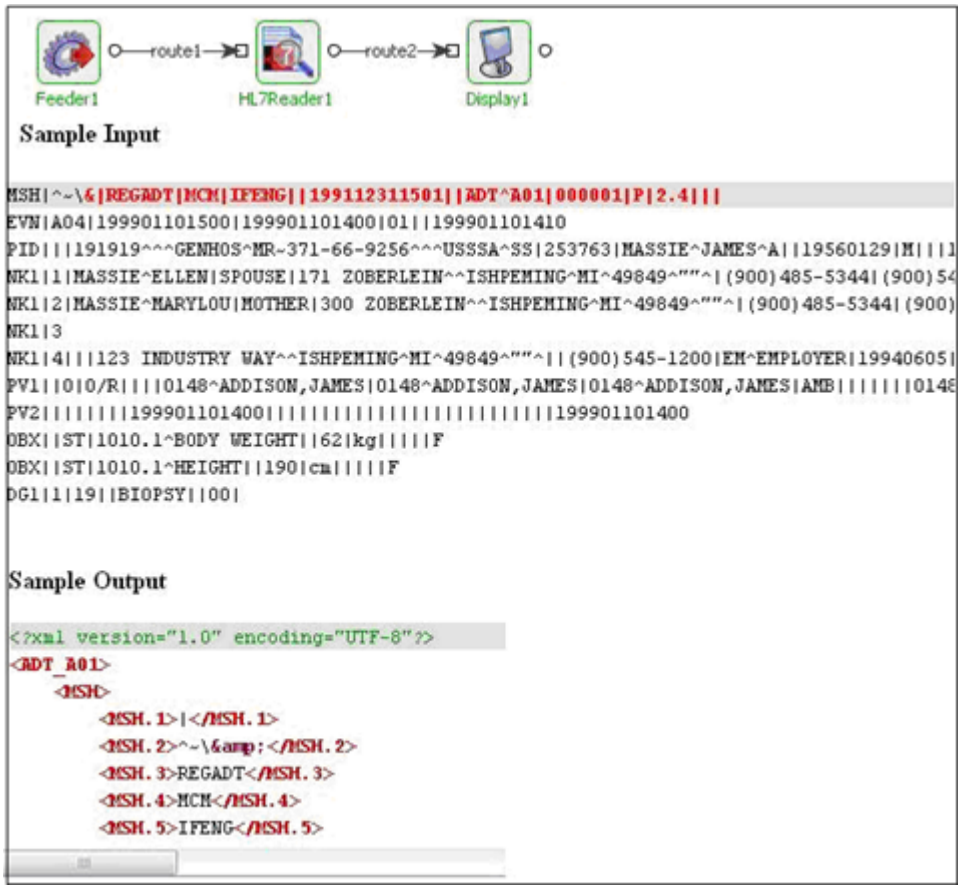


Figure 4.404: Scenario demonstration with sample input and output

4.11.2.3 Useful Tips

Please refer to http://www.hl7.org/ for more details.

4.11.3 HL7 Writer

The HL7Writer component is used to parse through XML documents and convert it to HL7 (Health Level Seven) format. HL7 is a standard to exchange management and integration of data that supports clinical patient care and the management, delivery and evaluation of healthcare components. This component is used to extract data present in XML format and convert it to HL7 format.

Note: Please refer to http://www.hl7.org/ for more details.

4.11.3.1 Configuration and Testing

4.11.3.1.1 Interaction Configuration

Only advanced parameters can be configured in the Interaction Configuration panel. No component specific details can be configured. Just open and close the Custom Property Sheet to save the default configuration.

4.11.3.1.2 Sample Input and Output

The configuration can be tested by clicking the Test button in the interaction Configuration panel.

Input Message | Output Message

```
<?xml version="1.0" encoding="UTF-8"?>
<ADT_A01>
    <MSH>
        <MSH.1>|</MSH.1>
        <MSH.2>^~\&amp;</MSH.2>
        <MSH.3>REGADT</MSH.3>
        <MSH.4>MCM</MSH.4>
        <MSH.5>IFENG</MSH.5>
        <MSH.6/>
        <MSH.7>199112311501</MSH.7>
        <MSH.8/>
        <MSH.9>
            <MSG.1>ADT</MSG.1>
            <MSG.2>A01</MSG.2>
        </MSH.9>
        <MSH.10>000001</MSH.10>
```

Figure 4.405: Sample Input Message

Input Message | Output Message

```
MSH|^~\&|REGADT|MCM|IFENG||199112311501||ADT^A01|000001|P|2.4|||
EVN|A04|199901101500|199901101400|01||199901101410
PID|||191919^^^GENHOS^MR~371-66-9256^^^USSSA^SS|253763|MASSIE^JAME
NK1|1|MASSIE^ELLEN|SPOUSE|171 ZOBERLEIN^^ISHPEMING^MI^49849^""^|(9
NK1|2|MASSIE^MARYLOU|MOTHER|300 ZOBERLEIN^^ISHPEMING^MI^49849^""^|
NK1|3
NK1|4|||123 INDUSTRY WAY^^ISHPEMING^MI^49849^""^||(900)545-1200|EM
PV1||O|O/R||||0148^ADDISON,JAMES|0148^ADDISON,JAMES|0148^ADDISON,J.
PV2||||||||199901101400||||||||||||||||||||||||||199901101400
OBX||ST|1010.1^BODY WEIGHT||62|kg|||||F
OBX||ST|1010.1^HEIGHT||190|cm|||||F
DG1|1|19||BIOPSY||00|
```

Figure 4.406: Response Generated

4.11.3.2 Functional Demonstration

4.11.3.2.1 Scenario 1

This scenario explains the basic functionality of HL7Writer component.

Configure the HL7Writer with the default configuration and use feeder and display component to send sample input and to check the response respectively. The XML input is sent from the feeder and the corresponding HL7 response is generated.

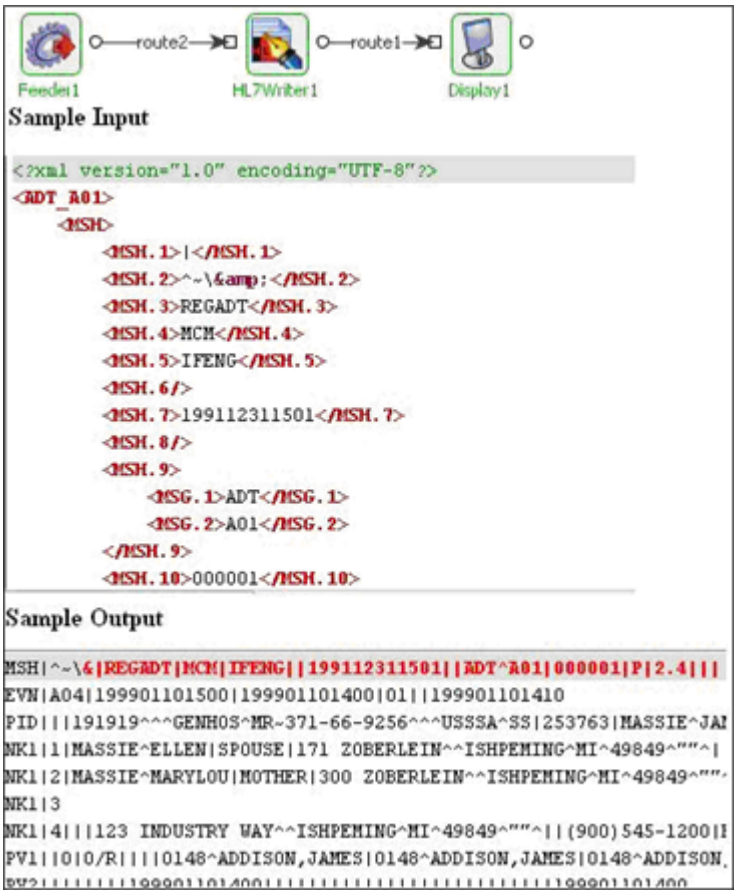


Figure 4.407: Scenario demonstration with sample input and output

4.11.3.3 Useful Tips

Please refer to http://www.hl7.org/ for more details.

4.11.4 Text2XML

The Text2XML component transforms data from any flat file format to XML format. This component accepts data in Text (delimited, positional or both) format and transforms it to the required XML format.

Note: The component takes TFL file as input in the CPS which describes the conversion rules. TFL files can be created or modified using the Fiorano Studio tool.

4.11.4.1 Configuration and Testing

4.11.4.1.1 Managed Connection Factory

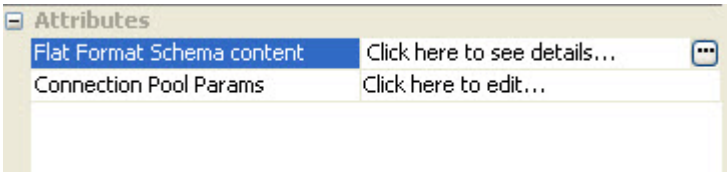


Figure 1: Managed Configuration Properties

The following attributes can be configured in the managed configuration panel.

Attributes

File Format Schema content: Flat Format Schema content (TFL file), which describes the conversion rules has to be specified here. The TFL schema can be provided using "Flat Format Schema content editor" which opens up on clicking ellipsis button against this property. This editor is shown in Figure 2. TFL files can be created using Fiorano Studio tool.

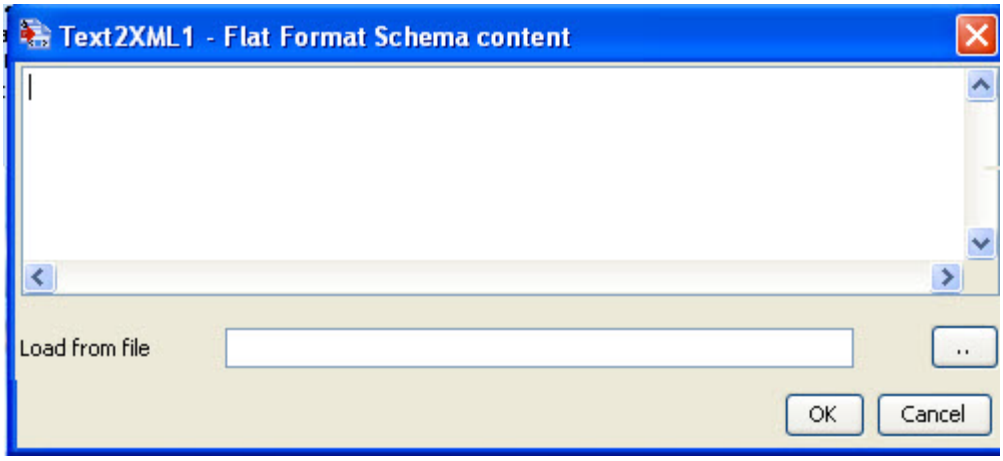


Figure 2: File Format Schema content Editor

4.11.4.1.2 Interaction Configuration

Attributes	
Enable dos2unix conversion?	**yes**
Validate Input	no
Cleanup resources (excluding connectio...	yes
Target Namespace	http://www.fiorano.com/fesb/activity/Text...
Monitoring configuration	disabled: 5000 milli seconds

Figure 3: Interaction Configuration Properties

The following attributes can be configured in the interaction configuration panel.

Attributes

Enable dos2unix conversion: This property should be used when CRLF(0D0A) should be converted to LF(0A) before processing the input message. New line is treated as CRLF(0D0A) in windows and as LF(0A) in unix.

If the TFL file contains delimiter as new line character, and we are not confident about where the flat file was generated either in windows or linux, then we need to set this property to yes. If we set this property to yes then the component converts all CRLF's to LF's in the input flat file and TFL file and then it applies the conversion rules specified in TFL file.

The transformation can be tested by clicking the Test button in the Interaction Configuration Panel. Sample message is generated depending on the Flat Format schema provided in the Managed Connection Factory panel.

4.11.4.2 Sample Input and Output

Sample CSV file format schema is shown in Figure 4. The structure of the schema is shown in Figure 5. Sample schema contains Employee records and each record contains EmployeeName, EmployeeID and EmployeeAge fields. This sample schema has Record Delimiter as CRLF and Filed delimiter as comma(,).

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE FileLayout
  PUBLIC "-//Fiorano Software Technologies Pvt. Ltd.//DTD FileSchema 1.0//EN" "http://www.fiorano.com/dtds/fileschema_1_0.dtd">
<FileLayout>
  <FileSections name="EmployeeSchema">
     <GlobalParsingInfo>
        <RecordDelimiter>
           <Delimiter type="custom-delimiter" escapeCharType="Character">0d0a</Delimiter>
        </RecordDelimiter>
        <DefaultFieldDelimiter>
           <Delimiter type="custom-delimiter" escapeCharType="Character" escapeChar="5c">2c</Delimiter>
        </DefaultFieldDelimiter>
     </GlobalParsingInfo>
     <Record xmlType="Element" minOccurs="-1" maxOccurs="-1" parsingType="DELIMITED" childCount="0" name="Employee">
        <RecordIdentifier type="None"/>
        <FieldDelimiter>
           <Delimiter type="DefaultFieldDelimiter" escapeCharType="Character"/>
        </FieldDelimiter>
        <RecordContent>
           <Field xmlType="Element" trimspaces="None" alignment="LEFT" mapIfNull="NO" name="EmployeeName">
              <DataType format="">String</DataType>
           </Field>
           <Field xmlType="Element" trimspaces="None" alignment="LEFT" mapIfNull="NO" name="EmployeeID">
              <DataType format="">String</DataType>
           </Field>
           <Field xmlType="Element" trimspaces="None" alignment="LEFT" mapIfNull="NO" name="EmployeeAge">
              <DataType format="">Integer</DataType>
           </Field>
        </RecordContent>
     </Record>
  </FileSections>
</FileLayout>
```

Record Delimiter set here as CRLF (new Line)

Field Delimiter set here as , (comma)

Figure 4: Sample CSV Schema Format

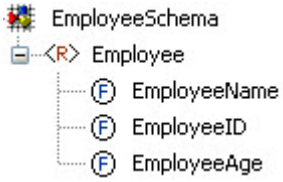


Figure 5: Structure for the Sample CSV SchemaFormat

Sample input is shown in Figure 6. It contains CSV formatted Employee records, which are separated by a new line character. The fields are separated by comma.

Input Message | Output Message

```
SCOTT,ID121,34
BOB,ID34,28
MICHAEL,ID234,41
AYRTON,ID230,27
```

Figure 6: Sample input

Output generated for the sample input is shown in Figure 6. The output xml structure is same as the structure of the CSV Schema Format.

Input Message | Output Message

```
<?xml version="1.0" encoding="UTF-8"?>
<EmployeeSchema xmlns="http://www.fiorano.com/fesb/activity/Text2XML1">
<Employee>
    <EmployeeName>SCOTT</EmployeeName>
    <EmployeeID>ID121</EmployeeID>
    <EmployeeAge>34</EmployeeAge>
</Employee>
<Employee>
    <EmployeeName>BOB</EmployeeName>
    <EmployeeID>ID34</EmployeeID>
    <EmployeeAge>28</EmployeeAge>
</Employee>
<Employee>
    <EmployeeName>MICHAEL</EmployeeName>
    <EmployeeID>ID234</EmployeeID>
    <EmployeeAge>41</EmployeeAge>
</Employee>
<Employee>
    <EmployeeName>AYRTON</EmployeeName>
    <EmployeeID>ID230</EmployeeID>
    <EmployeeAge>27</EmployeeAge>
</Employee>
</EmployeeSchema>
```

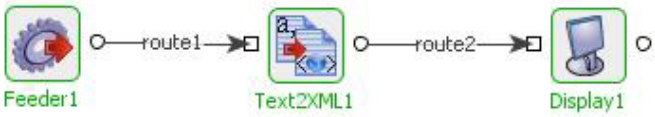
Figure 7: Sample Output

4.11.4.3 Functional Demonstration

4.11.4.3.1 Scenario 1

The scenario demonstrates the transformation of comma separated values (CSV) to XML.

Configure the Text2XML component as described in Configuration and Testing section and use feeder and display component to send sample input and check the response respectively. **CSV File Schema** is used in this scenario.



Sample Input

```
Alex,ABCDEF,24
```

Sample Output

```
<?xml version="1.0" encoding="UTF-8"?><Employee_Schema xmlns="http://www.fiorano.com/
fesb/activity/Text2XML1"><Employee><EmployeeName>Alex</EmployeeName><EmployeeID>ABCDEF
</EmployeeID><EmployeeAge>24</EmployeeAge></Employee></Employee_Schema>
```

Figure 8: Demonstrating Scenario 1 with sample input and output.

4.11.4.4 Use Case Scenario

In the Bond Trading sample Event Process, Text2XML component is used to convert the data from CSV format to the XML format.

The event process demonstrating this scenario is bundled with the installer. Documentation of the scenario and instructions to run the flow can be found in the Help tab of flow when opened in Studio.

4.11.4.5 Useful Tips

The documentation related to creating the Text Format Layout (TFL) files can be found under section 3.10 Text Schema Editor.

4.11.5 XML 2 EDI

The XML2EDI component is used for transforming information from XML format to EDI format. This component accepts data in XML format and transforms it to the required EDI format. It uses Eagle library to do the transformation.

Note: The component takes EFL file as input in the CPS which describes the conversion rules. EFL files can be created or modified using the Fiorano Studio tool.

4.11.5.1 Configuration and Testing

The XML2EDI component can be configured using its Custom Proper Sheet wizard. Following is the Interaction properties panel.

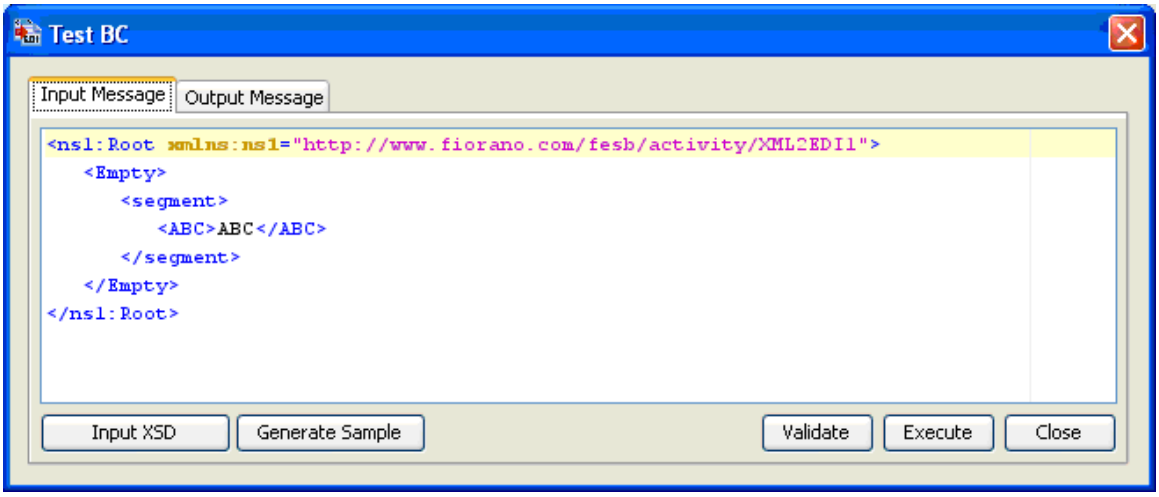
Attributes	
Description	null
FunctionName	xml2edi
Execution Timeout	0 milli seconds
Error Handling	click ... to do ErrorHandling configurations
Validate Input	no
Close After Each Request	yes
Target Namespace	http://www.fiorano.com/fesb/activity/XML2...

Figure 4.412: Sample XML2EDI configuration

Above configuration shown in the Figure 4.412 can be tested from within the CPS by clicking on test button in the CPS panel.

Sample EDI format schema to be provided in XML2EDI component.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Message
PUBLIC "-//mendelson.de//DTD for m-e-c eagle//EN" "http://www.fiorano.com/dtds/m-e-c.dtd">
<Message standard="EDIFACT" version="93" release="A" enableMissValue="0" name="Empty" minRepeat="0" maxRepeat="1" hideElement="0">
 <Segment id="ABC" delimiter="~" name="segment" description="" minRepeat="0" maxRepeat="1" hideElement="0">
<DataElement type="AN" default="ABC" minLength="3" maxLength="3" name="ABC" description="" minRepeat="0" maxRepeat="1" hideElement="0"/>
 </Segment>
</Message>
```


Test BC
Input Message
Output Message
<ns1:Root xmlns:ns1="http://www.fiorano.com/fesb/activity/XML2EDI1">
<Empty>

<ABC>ABC</ABC>

</Empty>
</ns1:Root>
Input XSD
Generate Sample
Validate
Execute
Close

Figure 4.413: Sample XML2EDI input message

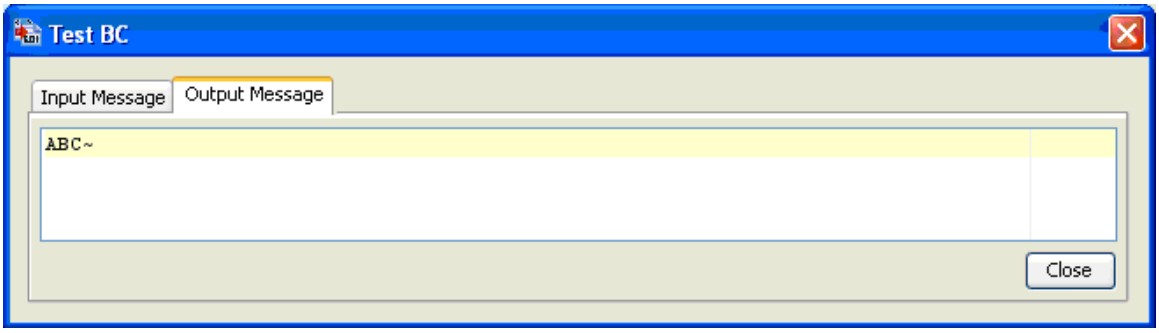


Figure 4.414: Sample XML2EDI output message

4.11.5.2 Functional Demonstration

4.11.5.2.1 Scenario 1

Send a message in EDI format as defined in the *Configuration and Testing* section and displaying the output XML message.

Configure the XML2EDI as described in the *Configuration and Testing* section and use feeder and display component to send sample input and check the response respectively.

Figure 4.415: Demonstrating Scenario 1 with sample input and output

Sample Input:

```
<ns1:Root xmlns:ns1="http://www.fiorano.com/fesb/activity/XML2EDI1">
 <Empty>
<segment>
 <ABC>ABC</ABC>
</segment>
 </Empty>
</ns1:Root>
```

Sample Output:

ABC~

4.11.6 XML2PDF

This component creates PDF files using XML and XSL Style sheet (or XSL-FO) as input.

It uses Apache FOP library to create PDF files.

4.11.6.1 Configuration and Testing

The following properties can be configured in the Custom Property Sheet of the component.

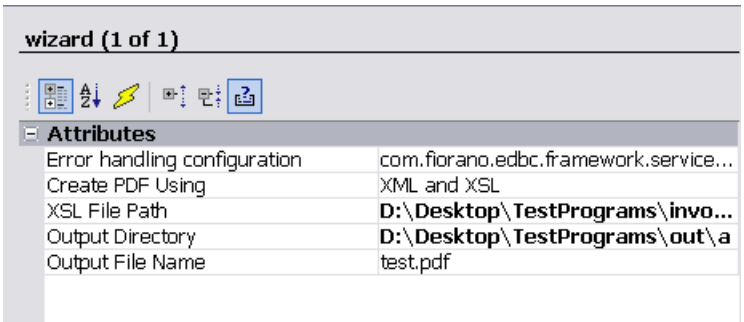


Figure 1: Component configuration

Error Handling Configuration: The remedial actions to be taken when a particular error occurs can be configured here.

Click on the ellipsis button against this property to configure Error Handling properties for different types of Errors.

By default, the options Log to Error Logs and Send to error port are enabled.

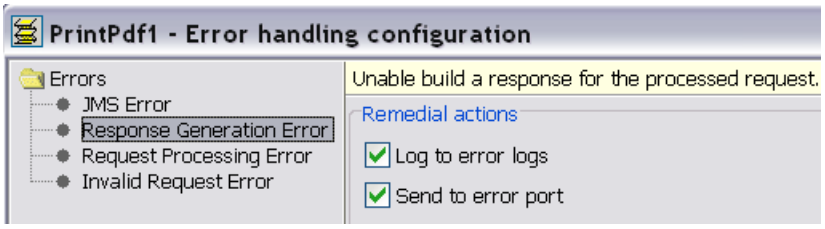


Figure 2: Error handling configuration

Create PDF Using: The input to the component can be configured here. The component accepts data in 2 formats XML and XSL and "XSL-FO".

When XML and XSL is selected, the path of the XSL style sheet has to be specified in the Custom Property Sheet and XML data has to be sent as an input to the component through its input port.

When XSL-FO is selected, XSL-FO has to be sent as an input to the component through its input port.

XSL File Path: This property is visible when the input type is "XML and XSL". The path of the XSL style sheet has to be specified here.

This property has a file filter and only files with .xsl extension will be shown in the file chooser.

Output Directory: The output directory in which PDF files have to be generated should be specified here.

Output File Name: The name for the output file that has to be created. By default this name is used for the output file. If a different file name has to be used, then a JMS Property "filename" has to be set on the input message to the component.

These properties can also be configured as named configurations and can be used across multiple instances. Figure 3 shows the properties as named configuration.

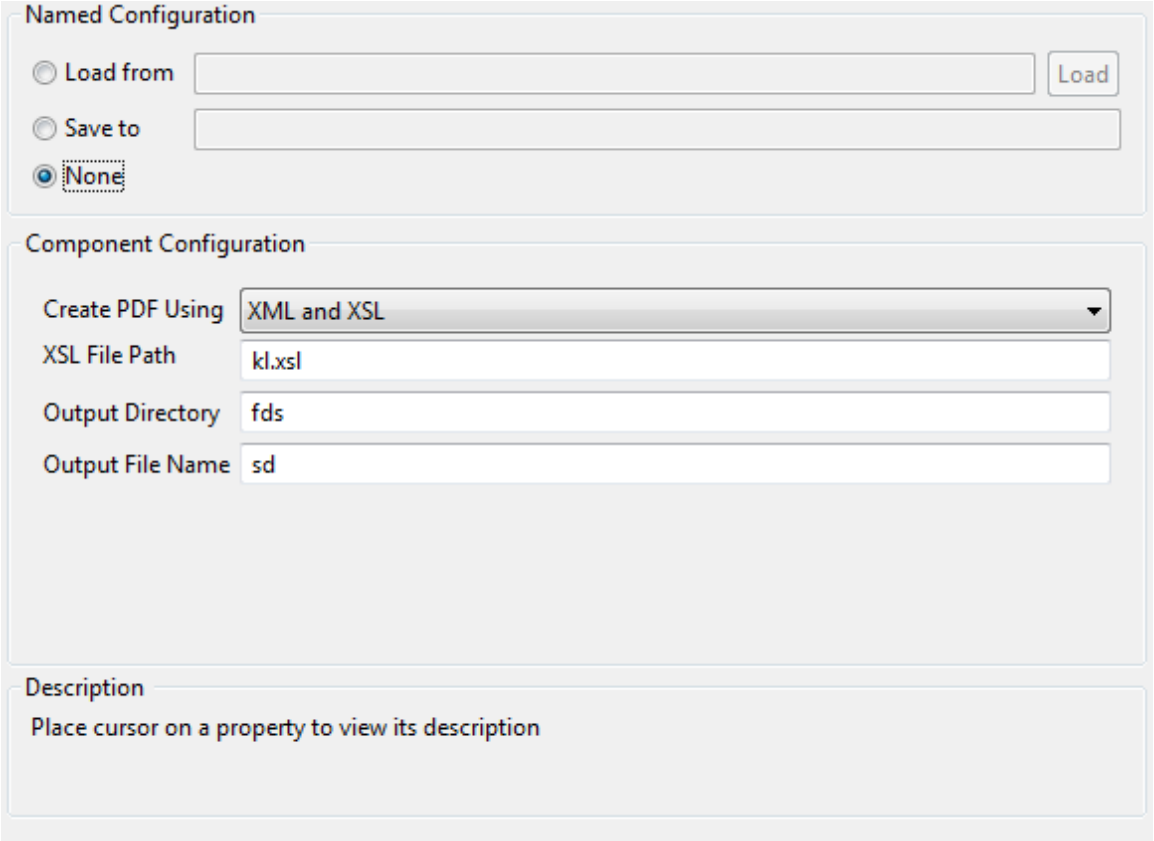


Figure 3: Named configuration

4.11.6.1.1 Sample Input and Output

XML and XSL

Configure XML2PDF component to use XML and XSL as input type as explained in Configuration and Testing section.

Launch the flow and send the sample XML input through Feeder component.

If the PDF creation is successful, the component sends an XML message which contains OutputDirectory, OutputFile and Message elements on to its output.

If there is any error in generating the PDF file, an error message indicating the exceptions will be sent to the component's ON_EXCEPTION port.

Figure 4 shows the sample input and output for the scenario mentioned.

Sample Input sent from Feeder

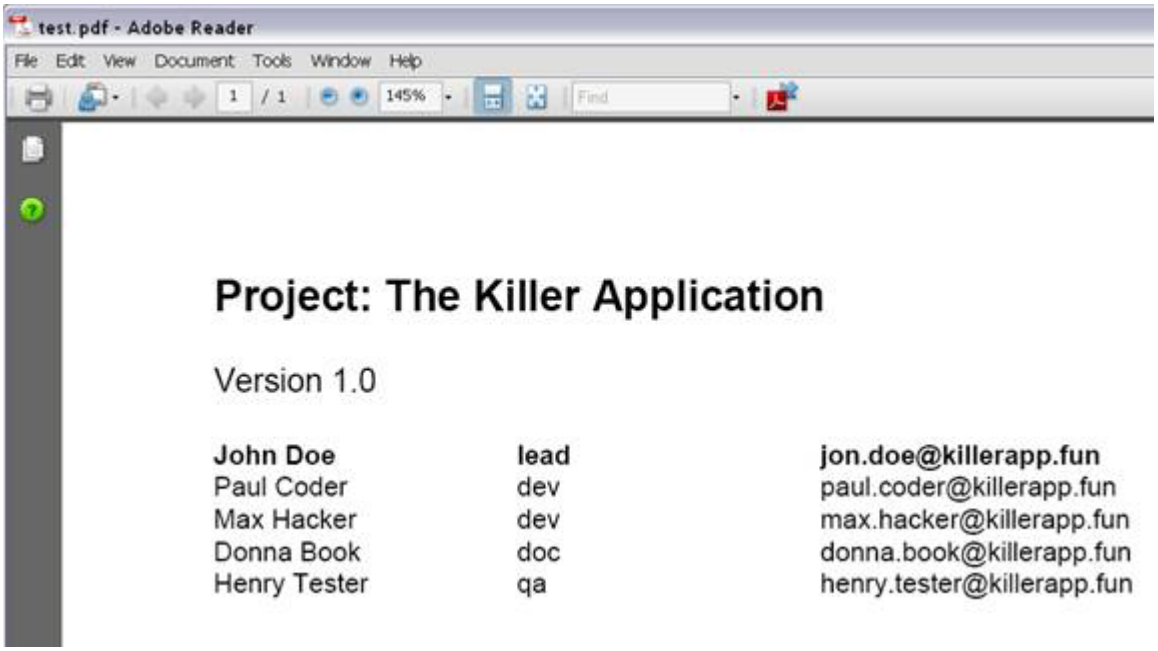
```
<?xml version="1.0" encoding="UTF-8"?>
<projectteam>
  <projectname>The Killer Application</projectname>
  <member>
    <name>John Doe</name>
    <function>lead</function>
    <email>jon.doe@killerapp.fun</email>
  </member>
  <member>
    <name>Paul Coder</name>
    <function>dev</function>
    <email>paul.coder@killerapp.fun</email>
  </member>
  <member>
    <name>Max Hacker</name>
```

Output Generated

Text

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ns1:Output xmlns:ns1="http://www.fiorano.com/fesb/XML2PDF/Out">
3     <OutputDirectory>D:\Desktop\TestPrograms\out\a</OutputDirectory>
4     <OutputFile>test.pdf</OutputFile>
5     <Message>PDF creation successful</Message>
6 </ns1:Output>
```

PDF File Created



XSL –FO

Configure XML2PDF to use XSL-FO as input type. Launch the flow and send the sample XSL-FO input through Feeder component. Also use a different output file name by setting filename JMS property on the input message.

If the PDF creation is successful, the component sends an XML output message which contains OutputDirectory, OutputFile and Message elements on to its output as shown below.

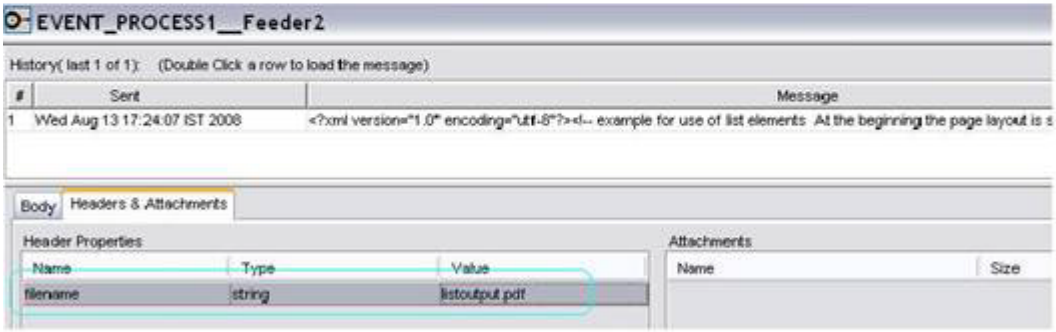
If there is any error in generating the PDF file, an error message indicating the exceptions will be sent to the component's ON_EXCEPTION port.

Figure 5 shows the sample input and output for the scenario mentioned.

Sample XSL-FO Input from feeder

```
<?xml version="1.0" encoding="utf-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="simple" page-height="29.7cm" page-width="21cm"
                   margin-top="2cm" margin-bottom="2cm" margin-left="2.5cm" margin-right="2.5cm">
      <fo:region-body margin-top="0cm"/>
      <fo:region-before extent="2cm"/>
      <fo:region-after extent="1.5cm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-reference="simple">

    <fo:flow flow-name="xsl-region-body">
      <fo:block font-size="18pt" font-family="sans-serif" line-height="24pt" space-after.optimum="15pt"
            background-color="blue" color="white" text-align="center" padding-top="3pt">
        How to use list elements
      </fo:block>

      <fo:block font-size="16pt"
            font-family="sans-serif"
```

Configuring filename JMS Property through feeder



Output Generated

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:Output xmlns:ns1="http://www.fiorano.com/fesb/XML2PDF/Out">
    <OutputDirectory>D:\Desktop\TestPrograms\out\a</OutputDirectory>
    <OutputFile>listoutput.pdf</OutputFile>
    <Message>PDF creation successful</Message>
</ns1:Output>
```

PDF File

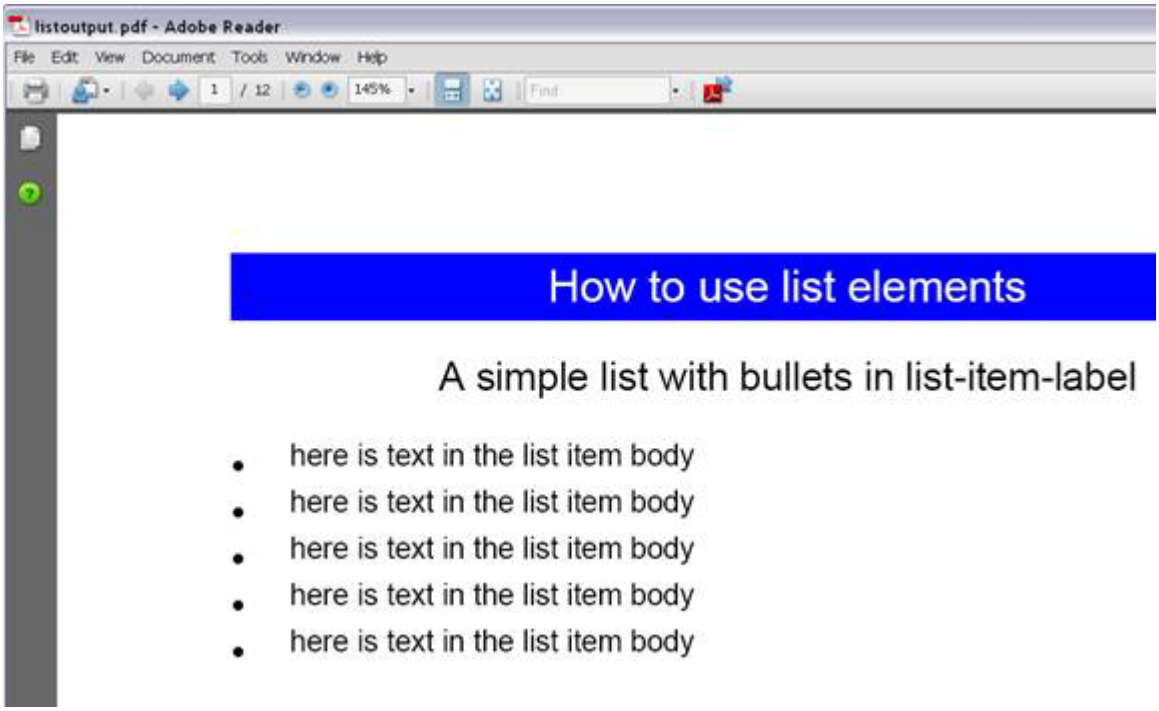


Figure 5: Sample input and output

4.11.7 XML 2 Text

The XML2Text component transforms data from XML format to flat file format. This component accepts data in XML format and transforms it to the required Text (delimited, positional or both) format.

Note: The TFL file, used in the configuration of the XML2Text component, which describes the conversion rules, may be created or modified using the Fiorano Studio tool.

4.11.7.1 Configuration and Testing

4.11.7.1.1 Managed Connection Factory

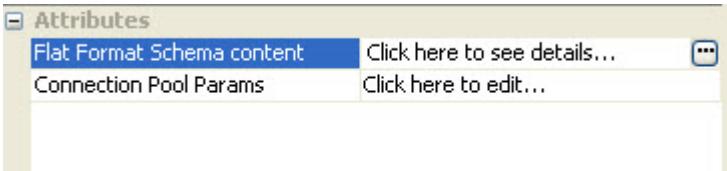


Figure 1: Managed Configuration Properties

The following attributes can be configured in the managed configuration panel.

Attributes

File Format Schema content: Flat Format Schema content (TFL file), which describes the conversion rules has to be specified here. The TFL schema can be provided using "Flat Format Schema content editor" which opens up on clicking ellipsis button against this property. This editor is shown in Figure 2.

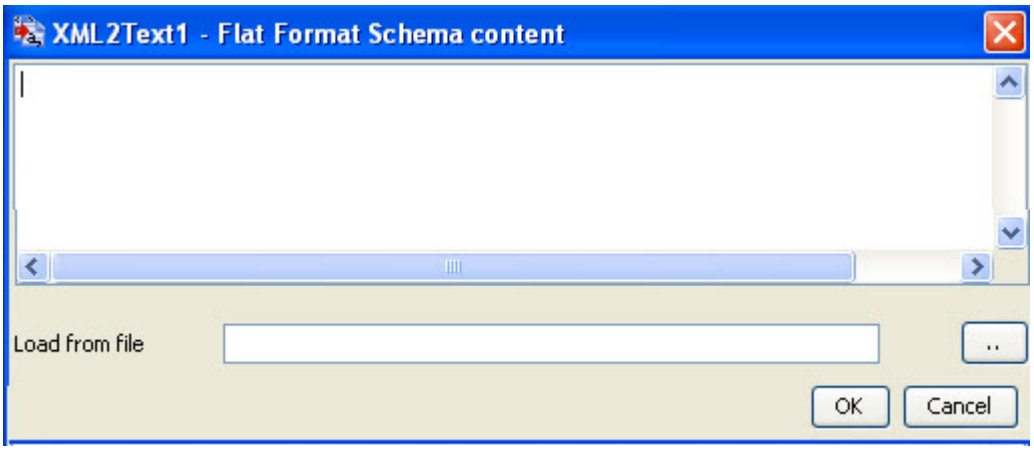


Figure 2: File Format Schema content Editor

4.11.7.1.2 Interaction Configuration

In the Interaction Configuration panel, If Prefix namespace for only root element attribute is selected, the schema generated will have elementformdefault as unqualified and the child elements will belong to the namespace of root element, otherwise each element will have its own namespace prefix. In the Target Namespace attribute, the target namespace of the input XML to be converted can be specified. There are also advanced settings like Monitoring Configuration, Cleanup resources after each document, Validate Input and so on can be provided (if needed). Click on the Expert Properties icon to view these attributes.

4.11.7.2 Sample Input and Output

Sample CSV file format schema is shown in Figure 3. The structure of the schema is shown in Figure 4. Sample schema contains Employee records and each record contains EmployeeName, EmployeeID and EmployeeAge fields. This sample schema has Record Delimiter as CRLF and Filed delimiter as comma(,).

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE FileLayout
  PUBLIC "-//Fiorano Software Technologies Pvt. Ltd.//DTD FileSchema 1.0//EN" "http://www.fiorano.com/dtds/fileschema_1_0.dtd">
<FileLayout>
  <FileSections name="EmployeeSchema">
    <GlobalParsingInfo>
      <RecordDelimiter>
        <Delimiter type="custom-delimiter" escapeCharType="Character">0d0a</Delimiter>
      </RecordDelimiter>
      <DefaultFieldDelimiter>
        <Delimiter type="custom-delimiter" escapeCharType="Character" escapeChar="5c">2c</Delimiter>
      </DefaultFieldDelimiter>
    </GlobalParsingInfo>
    <Record xmlType="Element" minOccurs="-1" maxOccurs="-1" parsingType="DELIMITED" childCount="0" name="Employee">
      <RecordIdentifier type="None"/>
      <FieldDelimiter>
        <Delimiter type="DefaultFieldDelimiter" escapeCharType="Character"/>
      </FieldDelimiter>
      <RecordContent>
        <Field xmlType="Element" trimspaces="None" alignment="LEFT" mapIfNull="NO" name="EmployeeName">
          <DataType format="">String</DataType>
        </Field>
        <Field xmlType="Element" trimspaces="None" alignment="LEFT" mapIfNull="NO" name="EmployeeID">
          <DataType format="">String</DataType>
        </Field>
        <Field xmlType="Element" trimspaces="None" alignment="LEFT" mapIfNull="NO" name="EmployeeAge">
          <DataType format="">Integer</DataType>
        </Field>
      </RecordContent>
    </Record>
  </FileSections>
</FileLayout>
```

Record Delimiter set here as CRLF (new Line)

Field Delimiter set here as , (comma)

Figure 3: Sample CSV Schema Format

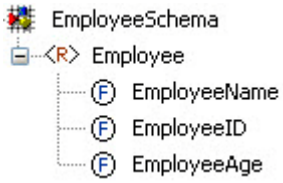


Figure 4: Structure for the Sample CSV SchemaFormat

4.11.7.2.1 Sample Input

The input XML structure is same as that of the sample CSV file format schema which is shown in Figure 4. Sample input is shown in Figure 5. It contains multiple Employee elements under EmployeeSchema element. Each Employee element contains EmployeeName, EmployeeAge and EmployeeID elements.

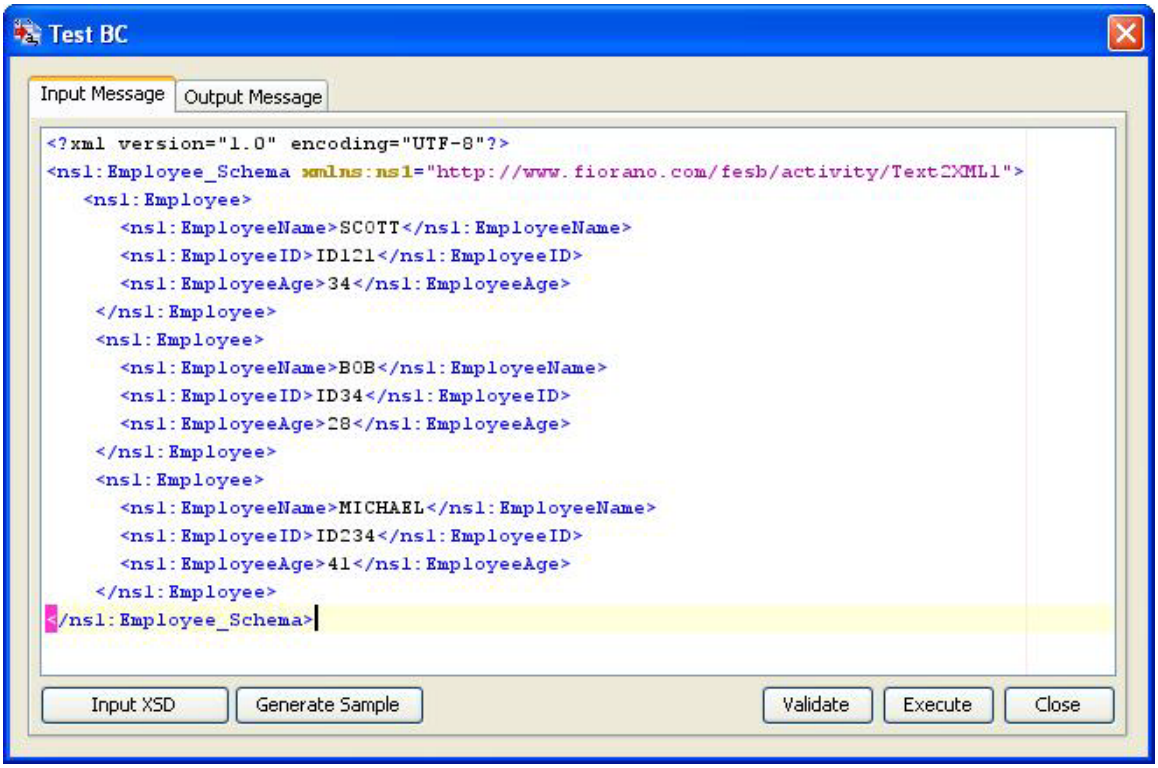


Figure 5: Sample Input

4.11.7.2.2 Sample Output

Sample output contains CSV formated Employee Records, which are separated by new line character (CRLF) and the fields are separated by a comma (,).

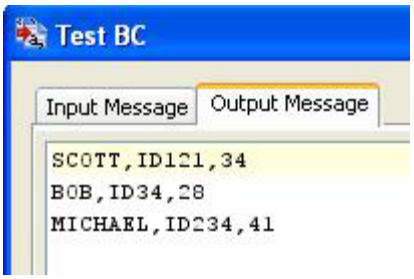


Figure 6: Sample Output

4.11.7.3 Functional Demonstration

4.11.7.3.1 Scenario 1

This scenario demonstrates the transformation of XML into comma-separated values (CSV).

Configure the XML2Text component as described in Configuration and Testing section and use feeder and display components to send sample input and check the response respectively. **CSV File Schema** is used in this scenario.

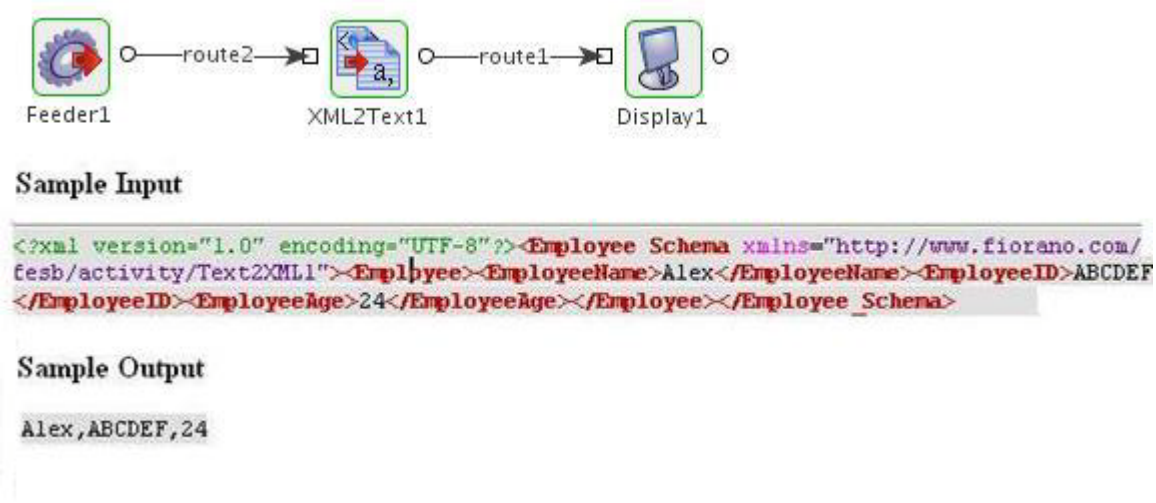


Figure 7: Demonstrating Scenario 1 with sample input and output.

4.11.7.4 Useful Tips

The documentation related to creating Text Format Layout (TFL) files can be found under section 3.10 Text Schema Editor.

4.11.8 XSLT

The XSLT component allows user to configure source and target document structures using Fiorano Mapper and create a XSL used for transforming documents. Alternatively, it allows users to define XSL created using external tools. Documents passed to the component are transformed using the XSL defined.

4.11.8.1 Interaction Configurations

The configuration for XSLT is defined in the **Custom Property Sheet** panel as shown in Figure 1.

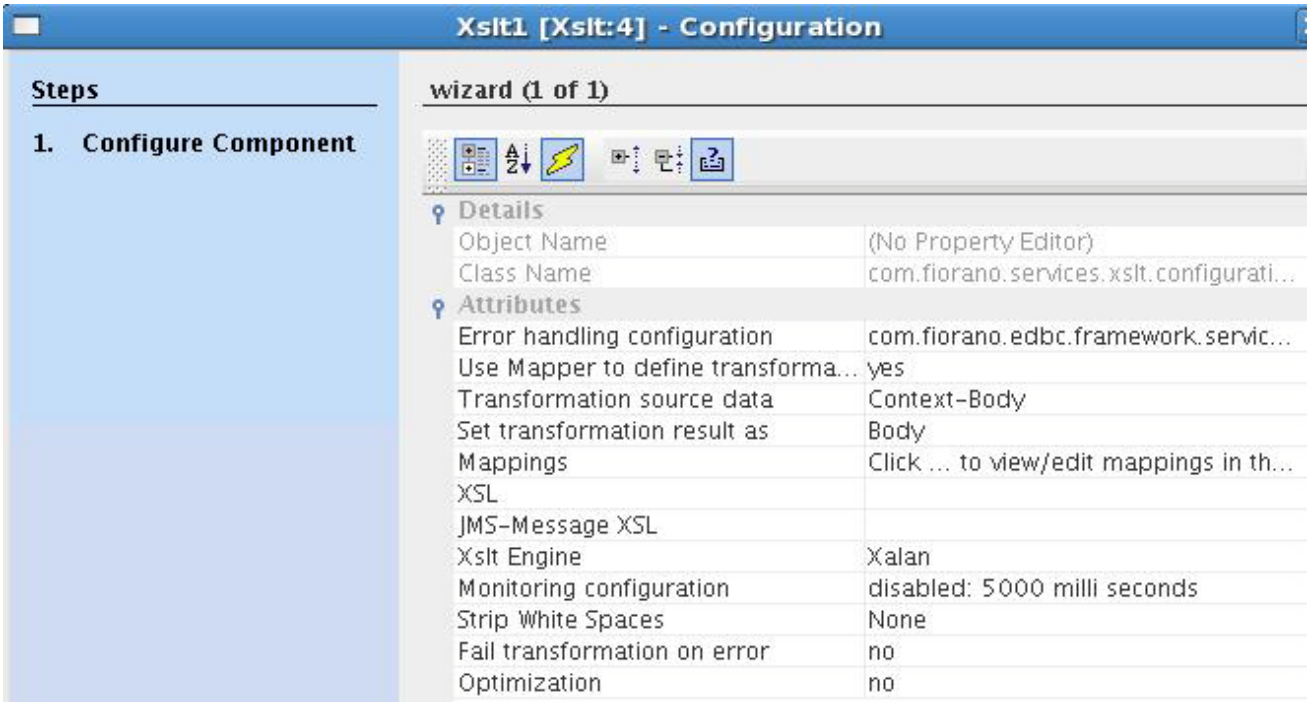


Figure 1: Custom Property Sheet with expert view enabled

4.11.8.1.1 Attributes

Error Handling Configuration

Click on the ellipsis button against Error Handling Configuration property to configure Error Handling properties.

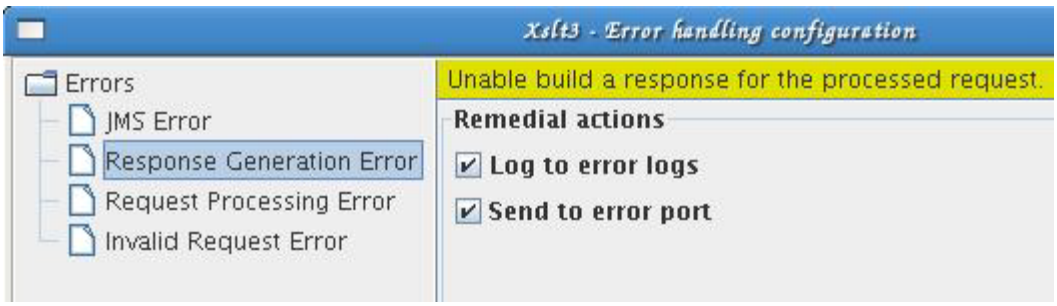


Figure 2: Error Handling Configuration

The remedial actions to be taken when a particular error occurs can be configured here.

The default actions configured are:

- Log to error logs
- Send to error port

4.11.8.1.1.1 Use Mapper to define transformation

This property determines the means of defining XSL that is used for transformation.

- **Yes:** When this value is selected property **Mappings** is visible and property **XSL** is made expert. Fiorano Mapper is used to define transformation.
- **No:** When this value is selected property Mappings is not visible and property XSL is visible as a normal property. Fiorano Mapper cannot be used to define transformation and XSL for transformation has to be manually provided.

Note:

Changing this value from yes to no does the following changes

- Removes value defined for property **Mappings**
- Changes the value of Transformation source data to Body**,** if this value is not changed explicitly by the user even once after the CPS is opened.

Any mappings previously defined using Fiorano Mapper will have to be redone. However the XSL(s) computed from mappings and set against properties **XSL** and **JMS-Message XSL** will still be present and can be used for transformation if the property value for property **Transformation source data** is not changed. If the value for property **Transformation source data** is automatically changed it should be reverted back manually.

Changing the value back to yes does not restore the value of property **Mappings** and removes values for properties **XSL** and **JMS-Message XSL. These** XSL(s) have to be redefined.

4.11.8.1.1.2 Transformation source data

This property determines the source for XML instance(s) from the input message on which XSL transformation should be applied for generating output XML. The source for XML instance(s) can be:

- **Body:** The transformation is applied to XML instance that is taken from the message body of the input message. Input structure in Fiorano Mapper contains the XSD/DTD defined for message body (same as structure on the input port of the component).
- **Context:** The transformation is applied to XML instance that is taken from the application context property of input message. Input structure in Fiorano Mapper contains the XSD/DTD defined for application context.
- **Context-Body:** The transformation is applied to XML instances that are taken from both message body and application context property of input message. Input structure in Fiorano Mapper contains XSD/DTD defined for application context as well as XSD/DTD for input body.

  Note: In this case, XML instance of application context is treated as primary source. Elements in primary source can be referenced directly in XSL, where as elements of other structure should be referenced as document (<StructureName>)/<ElementName>.

4.11.8.1.1.3 Set transformation result as

This property determines where the result of transformation of source data using XSL is set in the output message.

- **Body** - The result of the transformation is set as message body on the output message.
- **Context** - The result of the transformation is set as the application context property of the output message.

4.11.8.1.1.4 Mappings

The property defines the Fiorano Mapper project (contents of .tmf file) that can be created using Fiorano Mapper. XSL required for transformation is created automatically based on the mappings defined in Fiorano Mapper. Click the ellipsis button to open the Fiorano Mapper tool for visually defining the XSL.

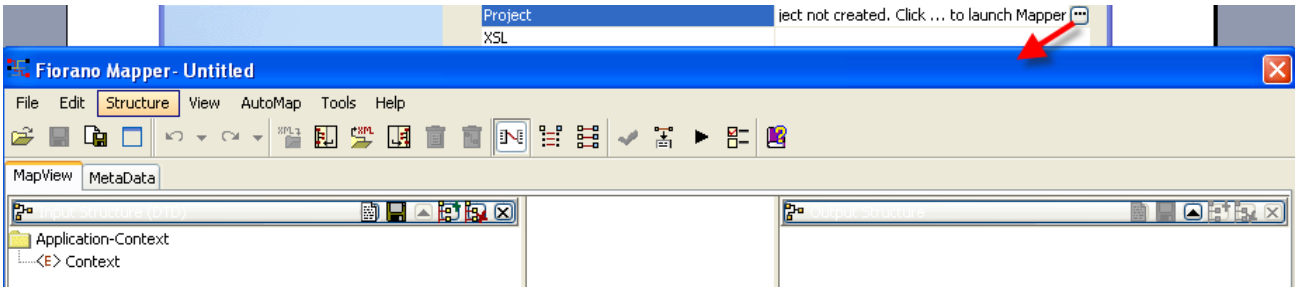


Figure 3: Launching Fiorano Mapper

Every time the Fiorano Mapper is closed after saving the mappings defined, XSL(s) computed using the mappings defined are set against the properties **XSL** and JMS-Message XSL**.** Any previously set XSL(s) content against these properties are overwritten.

Loading input and output structures in Fiorano Mapper

- Maximum number of structures that can loaded in the **Input Structure** of Fiorano Mapper is determined by the value for property **Transformation source data**.
    - **Body** – one input structure that describes the XML instance present in message body of input message.
    - **Context** – one input structure that describes the XML instance present in application context of input message.
    - **Context-Body** – Two input structures, first structure that describes the XML instance present in application context of input message and second structure that describes the XML instance present in message body of input message. The order of these structures should strictly be as described.
- Maximum of two structures can be loaded in the **Output Structure** of Fiorano Mapper. First structure that describes output XML instance which is the result of transformation and second structure is **JMS-Message** structure available from **Import Output Structure...** The order of these structures should strictly be as described and the second structure should only be **JMS-Message** structure, any other structure results in erroneous behavior at component's runtime.
- When the Fiorano Mapper is launched, input structures are loaded as described below.
    - If value for property **Transformation source data** is set to Body, **Input Structure** in Fiorano Mapper will be loaded with structure as described below.

        - Previously configured structure, if the component is already configured using Fiorano Mapper. If the structure defined on the output port of other component connected to input port of this component is changed, it is logged in the **Messages** window in Fiorano Mapper, but the previously configured structure is retained.
        - One of the structures on other component's output port which is connected to this component's input port, if the component is not previously configured using Fiorano Mapper.
        - No structures, if the component is not previously configured and the component's input port is not connected to any other component's output port which has a structure defined. In such a case, structure can be loaded manually either by typing in/copying the structure or from a file system.
    - If value for property **Transformation source data** is set to Context, **Input Structure** in Fiorano Mapper will be loaded with structure as described below.
        - Structure defined for application context in Event Process. If application context structure is changed after the previous configuration, then the new structure is loaded.
        - Default application context structure (<!ELEMENT Context (#PCDATA)>) if application context structure is not defined.
    - If value for property **Transformation source data** is set to Context-Body, **Input Structure** in Fiorano Mapper will be loaded with structures in order as described below.
        - Structure for application context as described for the case where Transformation source data is set to Context**.**
        - Structure for message body as described for the case where **Transformation source data** is set to Body.
- When the Fiorano Mapper is launched, output structures are loaded as described below.
    - If value for property **Set transformation result as** is set to Body, **Output Structure** in Fiorano Mapper will be loaded with structure as described below
        - Previously configured structure, if the component is already configured using Fiorano Mapper. If the structure defined on the input port of other component connected to output port of this component is changed, it is logged in the Messages window in Fiorano Mapper, but the previously configured structure is retained.
        - One of the structures on other component's input port which is connected to this component's output port, if the component is not previously configured using Fiorano Mapper.
        - No structures, if the component is not previously configured and the component's output port is not connected to any other component's input port which has a structure defined. In such a case, structure can be loaded manually either by typing in/copying the structure or from a file system.
    - If value for property **Set transformation result as** is set to Context, the **Output Structure** in Fiorano Mapper will be loaded with structure as described below.
        - Structure defined for application context in Event Process. If application context structure is changed after the previous configuration, then the new structure is loaded.

        - Default application context structure (<!ELEMENT Context (#PCDATA)>) if application context structure is not defined.
    - If the **JMS-Message** structure is loaded during previous configuration, then it is loaded as second structure.
- When structure of connected component's output or input port is changed, but previously configured structure is restored, to update the structure right-click the structure that should be changed and click **Update with imported structure...** and select **IN_PORT** or **OUT_PORT** respectively.

Note: When the structure is changed, mappings for nodes that are not at same Xpath location before and after the change is discarded.

- When the Fiorano Mapper is closed structures defined for message body, if any, in **Input Structure** or **Output Structure** is set on the input port or the output port of the component respectively.

4.11.8.1.1.5 XSL

This property defines the XSL that is used to transform source data from source defined against property **Transformation source data** to required output that is set on target defined against property Set transformation result as. When the value for property Use Mapper to define transformation is set to yes this value is automatically populated after defining mappings in Fiorano Mapper. When the value for property Use Mapper to define transformation is set to no**,** XSL should be manually provided here**.**

Note: If the XSL contains any custom java functions, jar files containing such functions should be added as resources to XSLT component.

4.11.8.1.1.6 Structure name for Input

This property is enabled when **Use Mapper to define transformation** is set to **no**. This property defines name of the input structure that is referenced in XSL provided.

Example: If the XSL has reference to Input as <xsl:variable name="document_IN_PORT" select="document('STRUCTURE_NAME')"/> then, the **Structure name for Input** must be provided as **STRUCTURE_NAME**

4.11.8.1.1.7 JMS-Message XSL

This property defines the XSL that is used to transform source data from source defined against property **Transformation source data** to required interim XML which defines JMS message properties and text that has to be set. The component parses this interim XML and sets the required JMS properties and their values on the output message. When the value for property Use Mapper to define transformation is set to yes this value is automatically populated after defining mappings in Fiorano Mapper. When the value for property **Use Mapper to define transformation** is set to no, XSL which generates XML corresponding to structure shown in Figure 4 should be manually provided here.

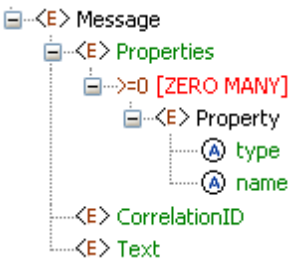


Figure 4: Structure of XML which is result of applying JMS-Message XSL

Note: JMS properties can be set alternatively using JMS Message Functions in Funclet tab of Fiorano Mapper.

Defining JMS-Message XSL using Fiorano Mapper

1. Define first output structure which represents the structure of message body or application context that has to be set on output message based on property **Set transformation result as**
2. Click Import Output Structure and choose JMS-Message as shown in Figure 5.

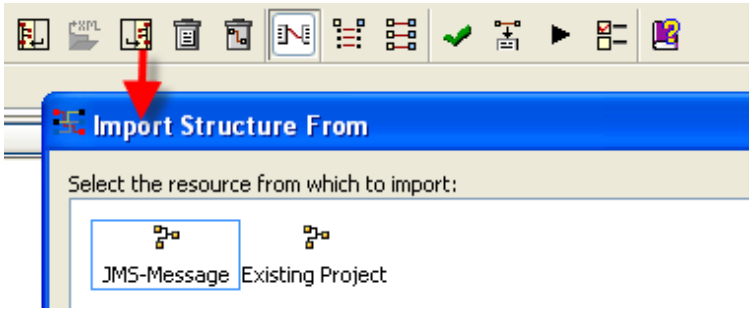


Figure 5: Loading JMS-Message structure on output

3. An output structure **JMS-Message** is added in the **Output Structure** as shown in figure 6.

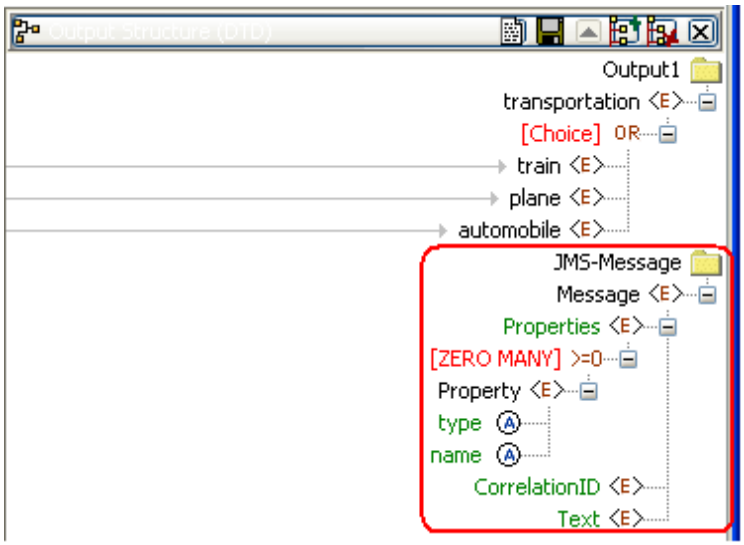


Figure 6: JMS-Message Structure loaded on output

4. For the property that has to be set on the output message, the name and type of the property should be mapped to the attributes name and type respectively. The value of the property should be mapped to the element Property. Figure 7 shows the mappings for name, type, and Property respectively.

Note: The type values can be Byte, Short, Integer, Long, Float, Double, and Boolean. Any other value defined for type is treated as String type.

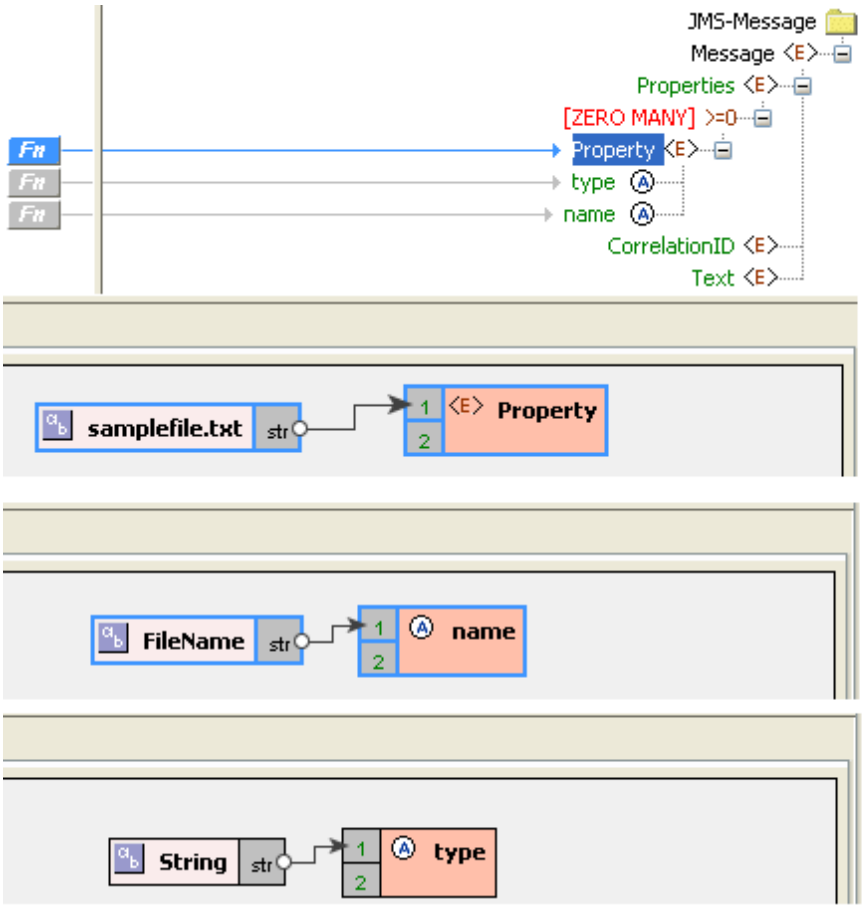


Figure 7: Mappings for defining a String property with name "FileName" and value "samplefile.txt"

5. Multiple properties can be added by duplicating the element Property Right-click the property element, select Configure and click the Duplicate option as shown in Figure 8.

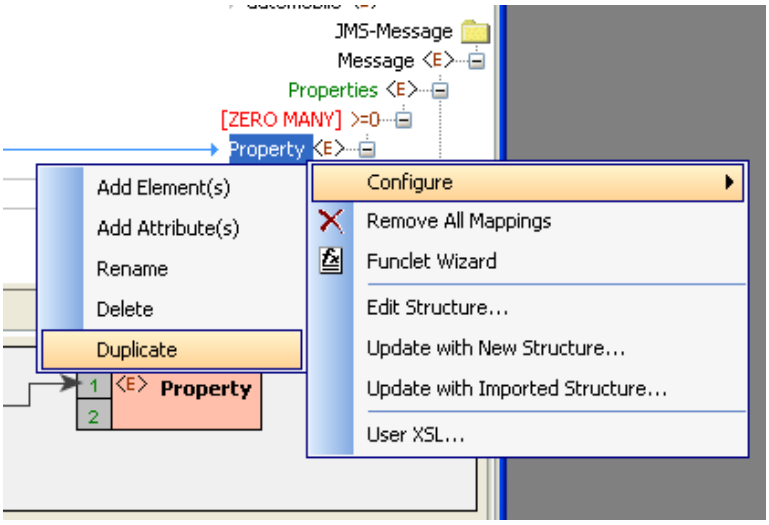


Figure 8: Duplicating Property node

6. To set the content of message body on the output message, required content should be mapped to Text as shown in Figure 9.

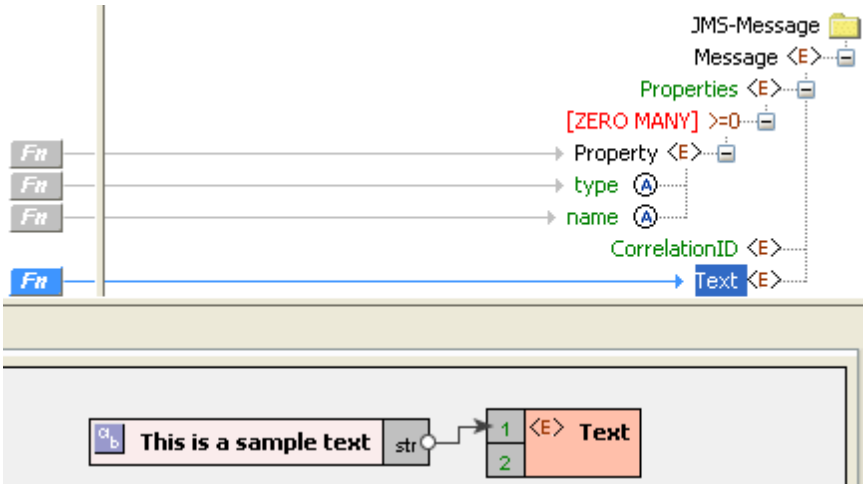


Figure 9: Setting message body

Note: When content is mapped to Text element of **JMS-Message** structure and **Set transformation result as** property is set to Body, content mapped to Text element of **JMS-Message** structure takes precedence of result of transformation and the body of output message contains content mapped to Text element of **JMS-Message** structure.

4.11.8.1.1.8 Xslt Engine

This property along with **Transformer factory class** property when this property's value is Other determines the transformer implementation that should be used to perform the transformation.

Xalan (2.7.0) and Saxon (8.4) transformer implementations are bundled with Fiorano environment for performing transformations.

- **Xalan:** Xalan implementation (org.apache.xalan.processor.TransformerFactoryImpl) is used to perform transformation.

  Note: Xalan (2.7.0) does not support XSLT 2.0

- **Saxon:** Saxon implementation (net.sf.saxon.TransformerFactoryImpl) is used to perform transformation.

  Note: Saxon implementation does not support custom functions

- **Other:** This option should be used when a custom transformer implementation has to be used. Selecting this option shows property Transformer factory class which can be used to provide the transformation factory implementation that should be used.

4.11.8.1.1.9 Transformer factory class

This property determines the fully qualified name of the class which should be used to perform transformation when the transformer implementation other than that is provided by Saxon or Xalan has to be used. The class provided should be an implementation of javax.xml.transform.TransformerFactory.

Resources (jar files) containing the java class specified against this property should be added as resources to XSLT component.

4.11.8.1.1.10 Strip White Spaces

This property determines whether elements in input XML which have only whitespace content should be stripped of the content unconditionally before the transformation is done.

Note: This property works only with Saxon.

- **None** - Attribute to strip whitespace is not set at all. Behavior dependents on the transformer's implementation. For Saxon implementation shipped with Fiorano the behavior is same as that of False.
- **True** - Whitespace content is stripped from input XML before the transformation is done.
- **False** - Whitespace content is retained as it is.

  Example: Figure 10 shows a sample transformation defined.

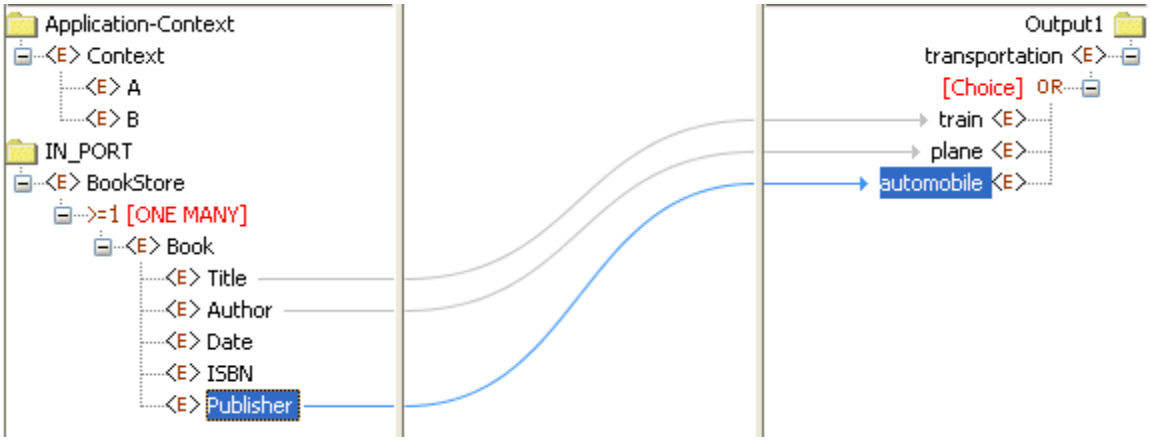


Figure 10: Sample transformation

Xslt Engine is chosen as Saxon and **Strip White Spaces** property is set to True as shown in Figure 10. Input XML in Figure 11 **contains** Title element which has whitespace.

Xslt Engine	**Saxon**
Strip White Spaces	True
Fail transformation on error	no

Figure 11: Saxon engine with Strip White Spaces set to True

Input XML, shown in Figure 12, contains only whitespace for element Title. After the transformation train element of output XML does not whitespace though there is a simple mapping from input XML's Title element.

Input Message | Output Message

```
<ns1:BookStore xmlns:ns1="http://www.books.org">
   <ns1:Book>
      <ns1:Title>     </ns1:Title>
      <ns1:Author>Author</ns1:Author>
      <ns1:Date>Date</ns1:Date>
      <ns1:ISBN>ISBN</ns1:ISBN>
      <ns1:Publisher>Publisher</ns1:Publisher>
   </ns1:Book>
</ns1:BookStore>
```

Input Message | Output Message

```
<?xml version="1.0" encoding="UTF-8"?>
<ns3:transportation xmlns:ns3="http://www.travel.org" xmln
   <ns3:train/>
   <ns3:plane>Author</ns3:plane>
   <ns3:automobile>Publisher</ns3:automobile>
</ns3:transportation>
```

Figure 12: Input XML with whitespace in element Title and output XML without whitespace for train

4.11.8.1.1.11 Fail transformation on error

This property determines how the problems that occur during the transformation have to be handled. There are three levels of problems that can occur during the transformation. Warnings, errors, and fatal errors.

- **no**
    - Transformer warnings are only logged at log level WARNING
    - Transformer errors are only logged at log level WARNING

- Transformer fatal errors are reported as errors in component

- **yes**
  - Transformer warnings are logged at level WARNING and are also treated as errors in component if **Throw fault on warnings** remedial action is enabled under **Request Processing Error** in **Error Handling** panel (Figure 2)
  - Transformer errors are reported as errors in component and remedial actions, if any, defined in Error Handling panel are taken.
  - Transformer fatal error are reported as errors in component.

4.11.8.1.1.12 Optimization

This property determines whether some internal structures can be cleared so as to make some additional memory available for transformation.

When property **Set transformation result as** is set to Body, message body is cleared from input message after message body is loaded into input for transformer and before transformation begins. In this case, Text-Content and Byte-Content functions of **JMS Message functions** in Fiorano Mapper (shown in Figure 13) cannot be used.

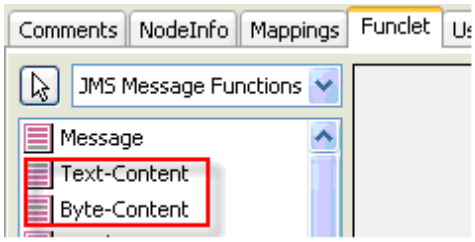


Figure 13: JMS Message functions in Fiorano Mapper

When property **Set transformation result as** is set to Context, content in application context is cleared from input message after content in application context is loaded into input for transformer and before transformation begins.

Note: This property comes into effect only when there is no **JMS-Message XSL** defined.

4.11.8.2 Functional Demonstration

4.11.8.2.1 Scenario 1

The scenario demonstrates a simple XSLT mapping. In this mapping,

1. The String Hello is appended before the Name in the Input message
2. Element Message is mapped to Message
3. Email is ignored
4. The response is sent to the output port

Configure the XSLT component as described in Configuration and Testing section and use feeder and display component to send sample input and check the response respectively.

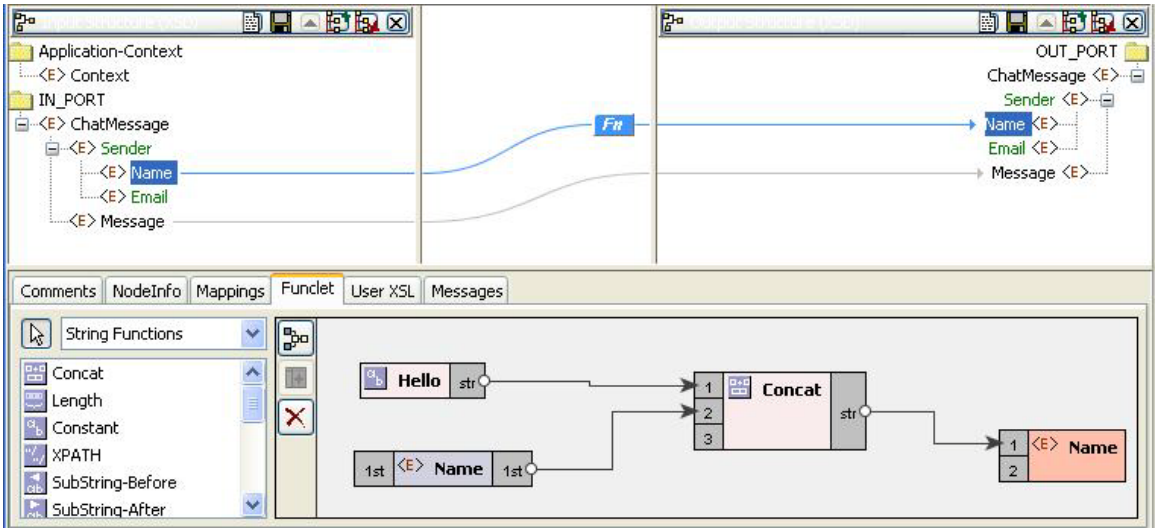


Figure 14: Mapping used

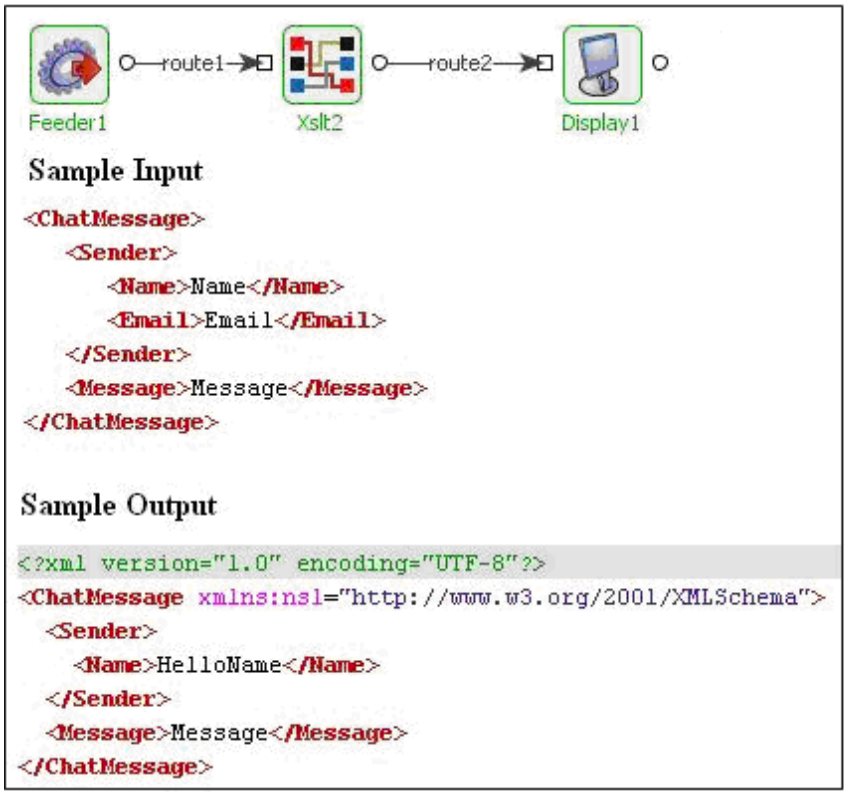


Figure 15: Demonstrating scenario 1 with sample input and sample output

4.11.8.3 Use Case Scenario

In EAI Demo sample, XSLT is used in extracting the Email ID, Order ID from the input XML and some mappings are defined between them with the POP3 schema elements and to construct the message body.

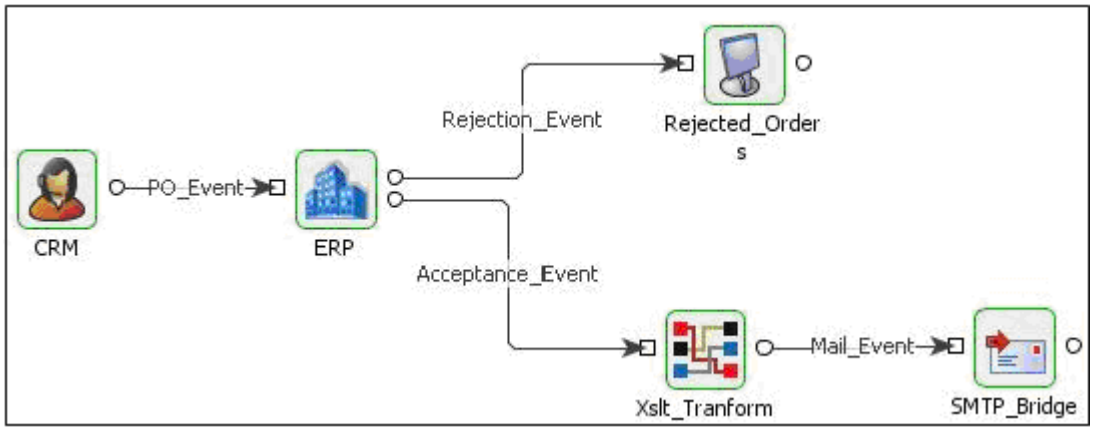


Figure 16: Sample use-case scenario

The Event Process demonstrating this scenario is bundled with the installer.

Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Studio.

4.11.8.4 Useful Tips

Suitable JDBC drivers required for Lookup functions have to be added as **Service Dependencies** or as **Resources** to **XSLTFunctions** System Lib.

4.11.9 JSONConverter

In recent times, usage of JSON format has been increased for serializing and transmitting structured data over a network connection like data transmission between server and a web application. Since JSON format is not widely adopted by all organizations yet, and as most of the data exchange is done through XML in earlier networks which have not deployed JSON, conversion from JSON to XML or vice-versa has become essential for data inter-operability. The JSONConverter component serves the need to convert data from either JSON format to XML format or from XML format to JSON format.

4.11.9.1 Configuration

Convert JSON To XML ? (Yes/No)

- If **Yes**, Converts Input JSON to XML.
- If **No**, Converts Input XML to JSON.

Convert JSON To XML ? Yes

General	
Error handling configuration	com.fiorano.edbc.framework.servic...
Convert JSON to XML ?	yes
Show JSON types as attributes?	yes
Output XSD Schema	null

Figure 1: Component configuration - To convert JSON to XML

Show JSON types as attributes ? (Yes/No)

Whether JSON element types should be included as attributes in Output XML.

Output XSD Schema

Output port Schema can be provided here using schema editor which opens upon clicking the ellipsis button. It can be used to generate sample output XML for the component.

Convert JSON To XML ? No

General	
Error handling configuration	com.fiorano.edbc.framework.servic...
Convert JSON to XML ?	no
Skip NameSpaces ?	yes
Remove NameSpace Prefix from ...	yes
Indent Factor	4
Input XSD Schema	null

Figure 2: Component configuration - To convert XML to JSON

Skip Namespaces? (Yes/No)

Whether to skip adding namespace declarations to XML elements while conversion.

Remove Namespace Prefix From Elements? (Yes/No)

Whether the namespace prefix from XML elements should be removed when reading for XMLSerializer

Indent Factor

String Indent value for JSON Object

Input XSD Schema

Input port Schema can be provided here using schema editor which opens upon clicking the ellipsis button. It can be used to generate sample input XML for the component.

Error Handling Configuration

Documentation about Error Handling Configuration can be found in section 10.7 Error Handling

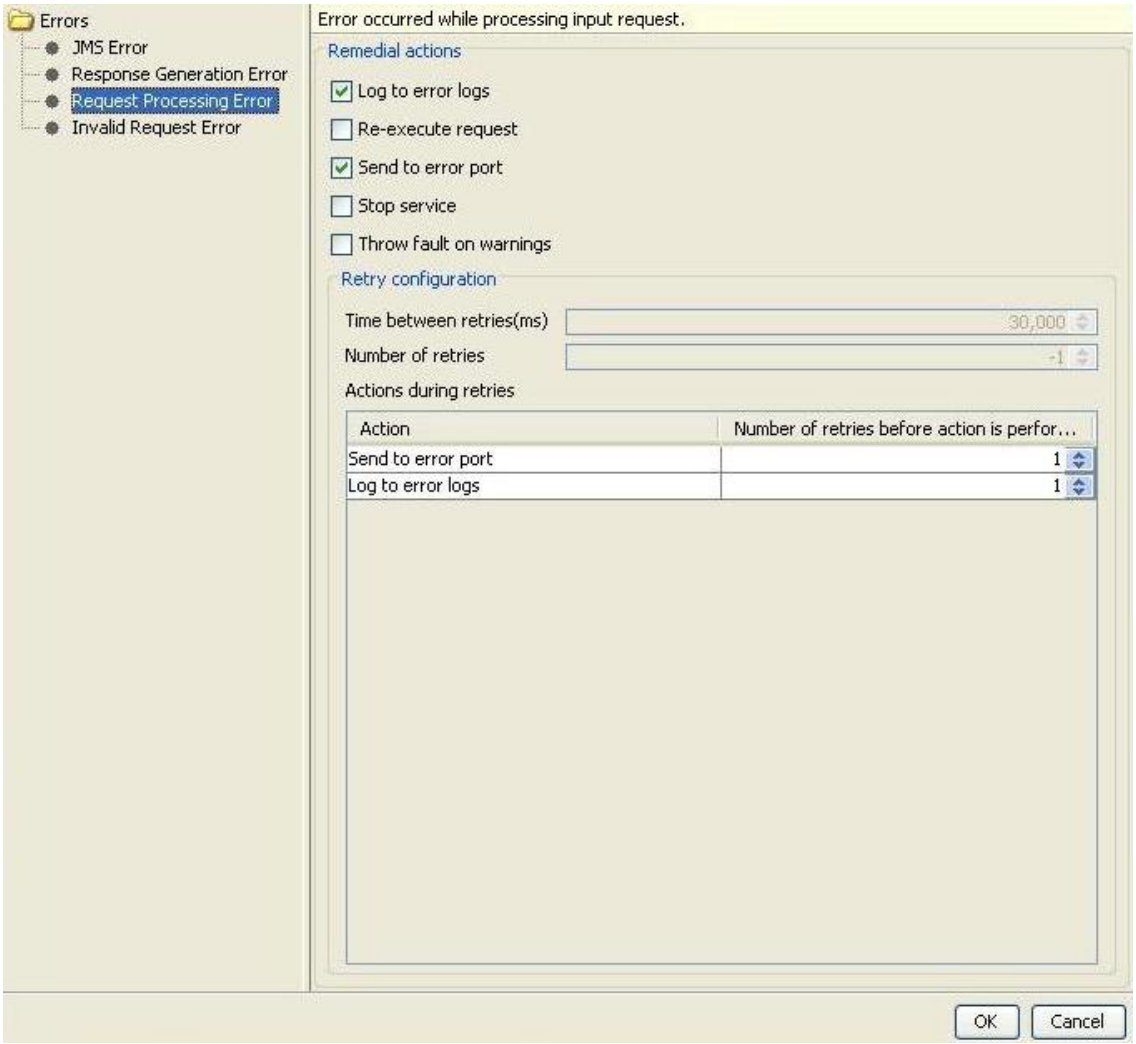


Figure 3: Error Handling Configuration

4.11.9.2 Functional Demonstration

4.11.9.2.1 Scenario1: JSON to XML

This scenario shows how JSONConverter can be used to convert data in JSON Format to XML Format.

Create flow by connecting JSONConverter to feeder and display components. Send sample input(valid JSON) to check the response(transformed XML message).Use Configuration from **Figure-1**. The JSON input is sent from the feeder and the corresponding XML output can be seen in display.

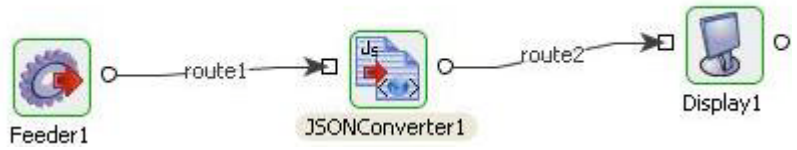


Figure 4: Sample Event Process

```
{"ns1:BookStore": {
    "@xmlns:ns1": "http://www.books.org",
    "ns1:Book":     {
        "ns1:Title": "Title",
        "ns1:Author": "Author"
    }
}}
```

Figure 5: Sample Input

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:BookStore xmlns:ns1="http://www.books.org"><ns1:Book><ns1:Author>Author</ns1:Author><ns1:Title>Title</ns1:Title></ns1:Book></ns1:Bool
```

Figure 6: Sample Output

4.11.9.2.2 Scenario 2: XML to JSON

This scenario shows how JSONConverter can be used to convert data in XML Format to JSON Format.

Create flow by connecting JSONConverter to feeder and display components. Send sample input(valid XML) to check the response(transformed JSON message).Use Configuration from **Figure-2**. The XML input is sent from the feeder and the corresponding JSON output can be seen in display.

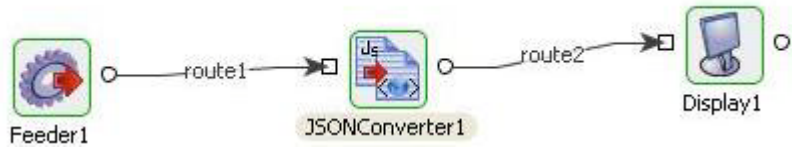


Figure 7: Sample Event Process

```
<ns1:BookStore xmlns:ns1="http://www.books.org">
   <ns1:Book>
      <ns1:Title>Title</ns1:Title>
      <ns1:Author>Author</ns1:Author>
      <ns1:Date>Date</ns1:Date>
      <ns1:ISBN>ISBN</ns1:ISBN>
      <ns1:Publisher>Publisher</ns1:Publisher>
   </ns1:Book>
</ns1:BookStore>
```

Figure 8: Sample Input

```
{"ns1:BookStore": {
    "@xmlns:ns1": "http://www.books.org",
    "ns1:Book":     {
        "ns1:Title": "Title",
        "ns1:Author": "Author",
        "ns1:Date": "Date",
        "ns1:ISBN": "ISBN",
        "ns1:Publisher": "Publisher"
    }
}}
```

Figure 9: Sample Output

4.11.9.3 Useful Tips

Output port and Input port XSD schemas can be used for mapping transformations for corresponding JSONtoXML and XMLtoJSON conversions.

4.12 Util

The Util category consists of components like Compression, Decompression, Decryption, DiskUsageMonitorService, Display, Encryption, and Feeder. The following section describes each component.

4.12.1 Compression

The Compression component is used to compress the incoming data and send it forward. It makes use of the APIs available in the java.util.zip package. The component compresses the text and attachment contained in the incoming document.

Note: Currently the component's compression algorithm is not configurable.

4.12.2 Decompression

The Decompression component is used to decompress the incoming data and send it forward. It makes use of the APIs available in the java.util.zip package. The component decompresses the text and the hash table values existing in the incoming document, and then sends the document forward.

Note: Currently the component's decompression algorithm is not configurable.

4.12.3 Decryption

The Decryption component is used for decrypting data, based on a key (that is entered by the user) and an algorithm. To decrypt the data accurately, user must know the correct key and algorithm, using which the data has been encrypted.

The supported algorithms are DES, PGP and Base64.

Note: The Decryption component uses Cryptix as the Security provider. Hence you have to configure the JRE so that the JVM can pick up the Cryptix security provider.

4.12.3.1 Configuration and Testing

The component has the following attributes which can be configured from its Configuration Property sheet. Figure 1 illustrates the panel with expert properties

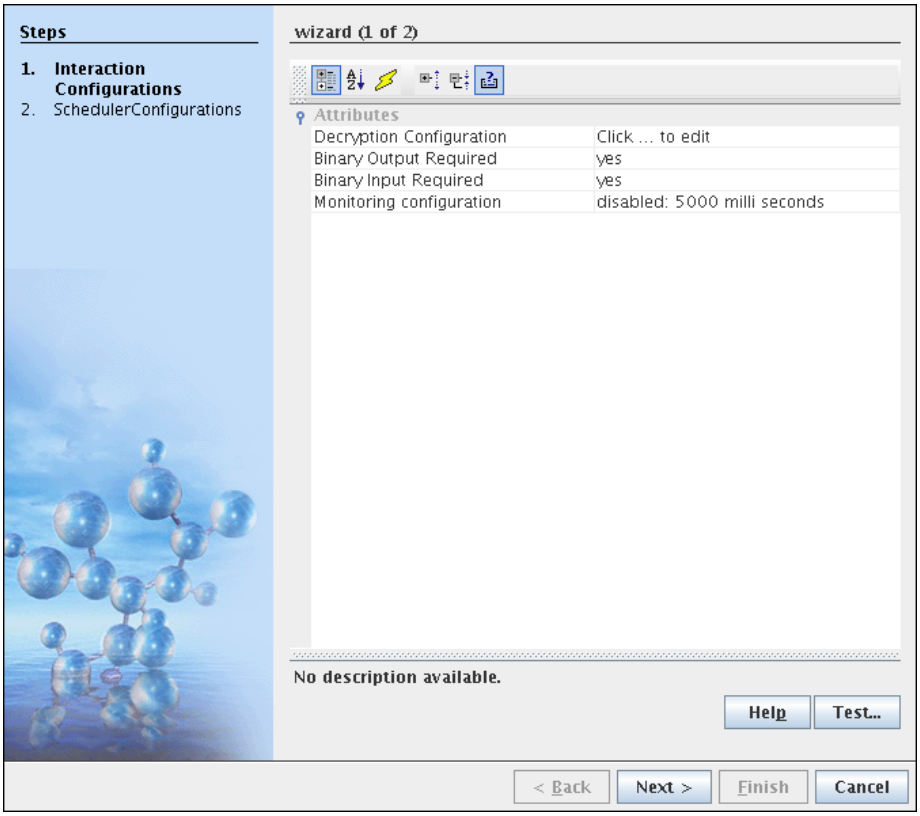


Figure 1: Configurable properties for Decryption component

4.12.3.1.1 Decryption Configuration

Decryption Details consists Data Encryption Details and Verification Details.

Data Encryption Details

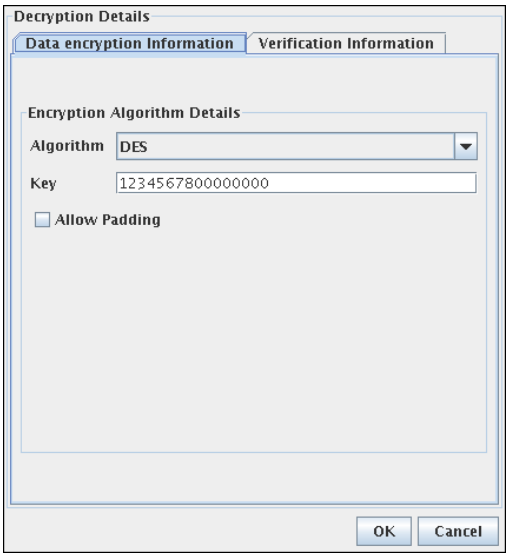


Figure 2: Encryption Algorithm Details

- **Algorithm**: The supported algorithms are DES, PGP, Base64, TripleDES, AES256, AES128, RC2-40, RC2-64, and RC2-128.
- **Key**: Specify key which is used to encrypt the incoming data. Specify 32 digit key for AES256, 16 digit key for AES128, 16 digit key for DES, 24 digit key for TripleDES, 5 digit key for RC2_40, 8 digit key for RC2_64 and 16 digit key for RC2_128
- **Allow Padding**: Choose yes to allow padding to the key to make exactly the value required for the mentioned algorithm. It has to be choosed only when we know that encryted data key is also padded.
- **Passphrase**: Required for signing in case of PGP algorithm.
- **Public Key File**: Specify public key file used in case of PGP Algorithm
- **Private Key File**: Specify private key file used in case of PGP Algorithm

Verification Information

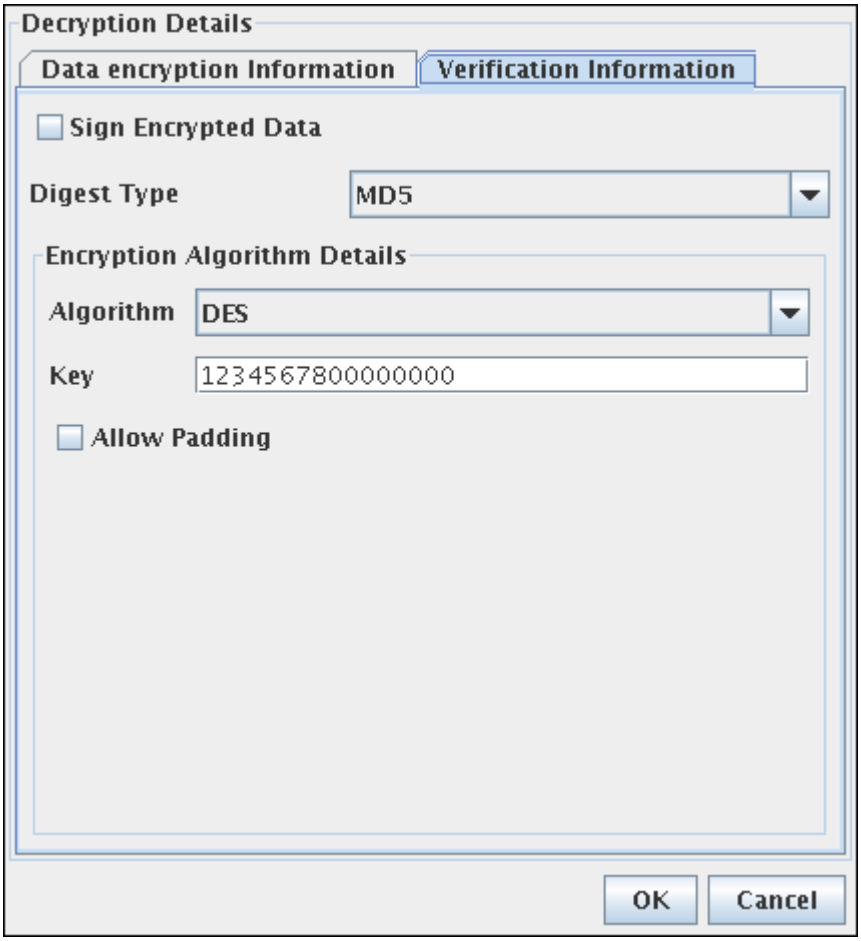


Figure 3: Digest type used and decryption algorithm used for verifying digested data can be provided as given above.

- **Sign Encrypted Data**: Select to verify only when you know that encrypted data has been signed.
- **Digest Type**: The supported algorithms for signing/digesting data are are MD5 and SHA-1.
- **Encryption Algorithm Details**: Specify the encryption details which are used to encrypt the signed data. It is done in the same way as shown in Data Encryption Details. Algorithm for data encryption and digest data encryption need not be same.

4.12.3.1.2 Binary Output Required

Click yes if Binary Output is needed. Else decrypted data will be in string format.

4.12.3.1.3 Binary Input Required

Click yes if input is Binary.

Additional Configuration

- **Cleanup resources**: Closes interaction after each request
- **Error Handling**: Specify error handling configuration.

4.12.3.1.3 Monitoring configuration

When enabled, component publishes USER_EVENTs containing following statistics to FPS_USER_EVENTS_TOPIC: Minimum Execution Time: Least amount of time taken to process any message during the last publish interval period Maximum Execution Time: Highest amount of time taken to process any message during the last publish interval period Count: Number of messages processed during the last publish interval period. Through put: Through put of the component during the last publish interval period.

4.12.3.1.4 Using Named Configurations

Configure the Decryption component as described in Configuration and Testing section and use feeder and display component to send sample input and check the response respectively.

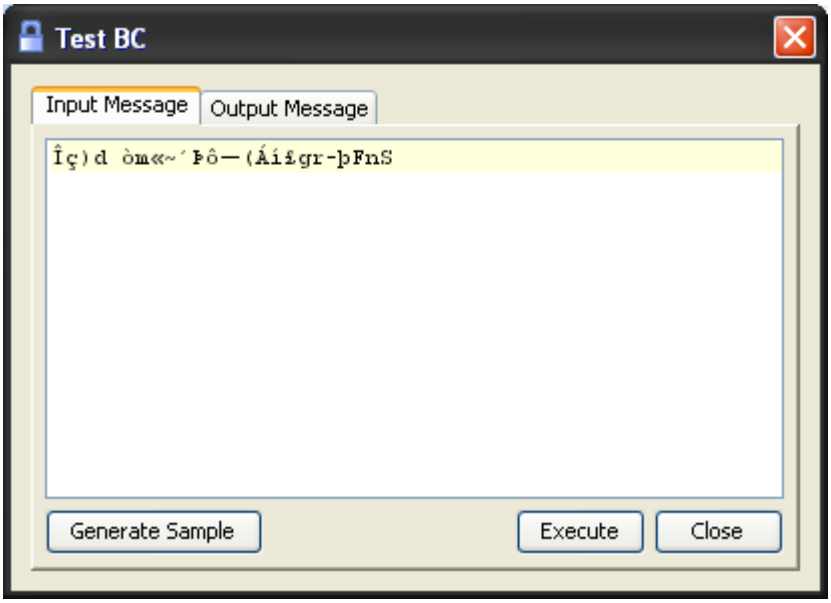


Figure 4.448: Sample input

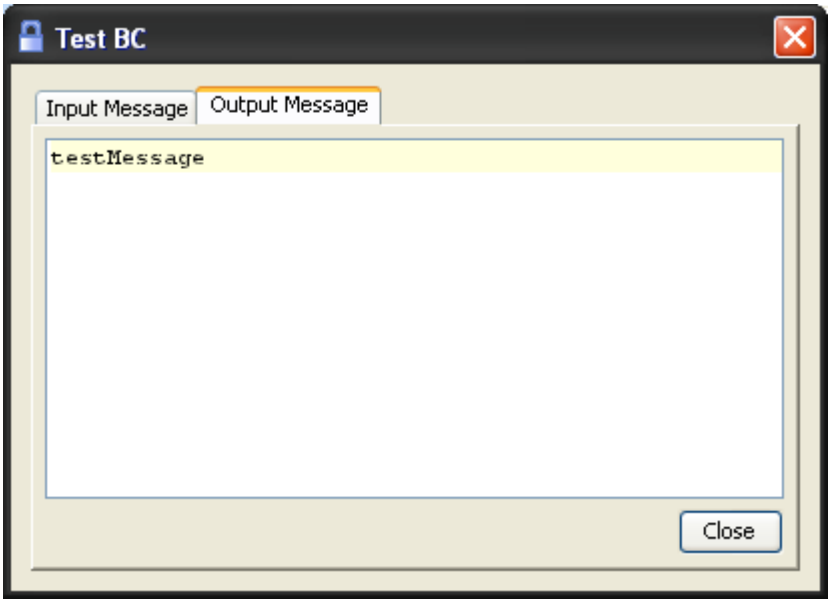


Figure 4.449: Sample output

4.12.3.2 Functional Demonstration

4.12.3.2.1 Scenario 1

Encryption of data received from input.

Configure the Decryption component as described in *Configuration and Testing* section and use feeder and display component to send sample input and check the response respectively.

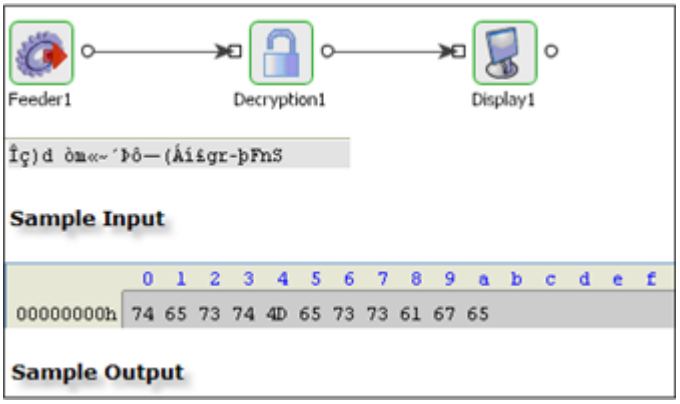


Figure 4.450: Demonstrating Scenario 1 with sample input and output

4.12.3.3 Use Case Scenario

In a bond trading scenario, request for quotes (RFQ) are sent in encrypted form to the appropriate exchanges and the received encrypted responses are decrypted at the user end

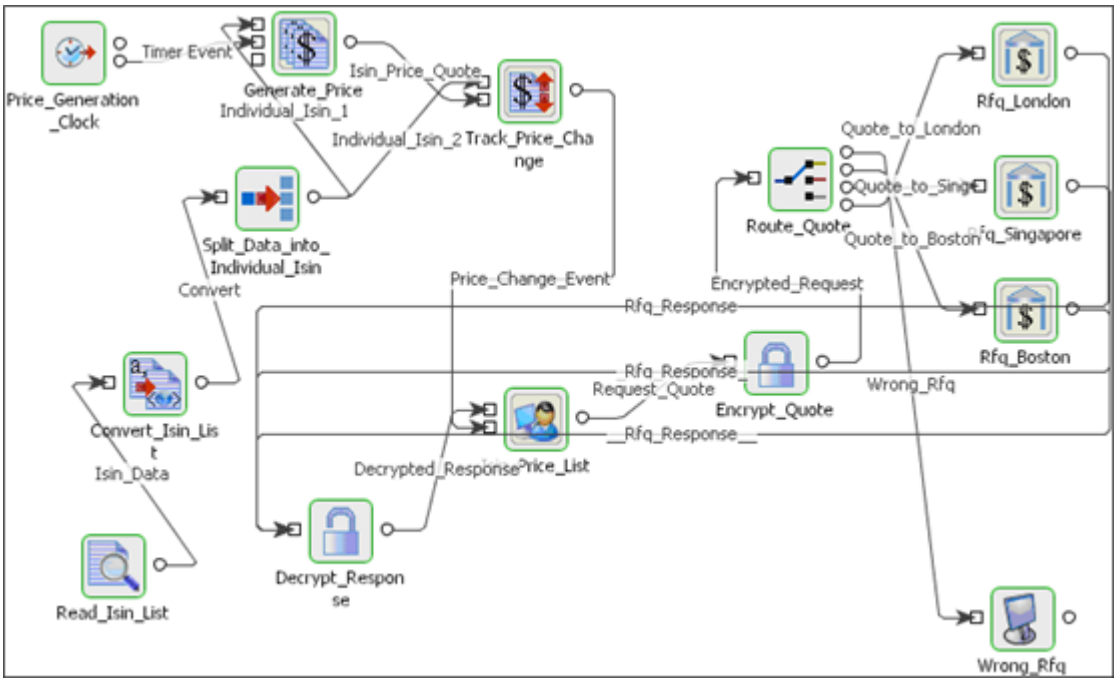


Figure 4.451: Bond Trading Scenario

The event process that demonstrates this scenario is bundled with the installer. Note encryption and decryption components may not be present.

Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Studio.

4.12.3.4 Useful Tips

The Decryption component uses Cryptix as the Security provider. Hence you need to configure the JRE so that the JVM can pick up the Cryptix security provider.

4.12.4 DiskUsageMonitorService

The DiskUsageMonitorService component can be used to monitor the hard disk usage of a particular drive or path on a host machine. This component also triggers sending of alerts to its output port whenever the usage reaches a maximum limit specified as a percentage

Points to note

- This component can only be used on Windows, Solaris and Linux platforms.
- This component cannot be launched in-memory of the peer server.

4.12.4.1 Configuration and Testing

The following parameters can be configured from the Custom Property Sheet of the component.

Name	Description
Disk to Monitor	Specifies the path of the disk that you intend to monitor.
Disk Usage Limit (%)	Specifies the maximum disk usage allowed. This value may be specified as a percentage. If the disk usage crosses the limit specified by this parameter, alerts are sent to the output port.
Monitoring Interval (sec)	Specifies the time duration, in seconds, for which this component waits to poll the disk to check its usage.
Display Service GUI	The user interface of the component is displayed if selected.

You can validate the above configured parameters by clicking on the **Validate** button.

4.12.4.2 Functional Demonstration

4.12.4.2.1 Scenario 1

Scenario demonstration of Disk Usage Monitor Service component which is configured to monitor the disk usage of 'C drive' on a windows machine.

Configure the parameters mentioned in section 2 and use a Display component to receive the alerts when the disk usage limit exceeds.

The Disk Usage Limit is configured for 80% in the following example.

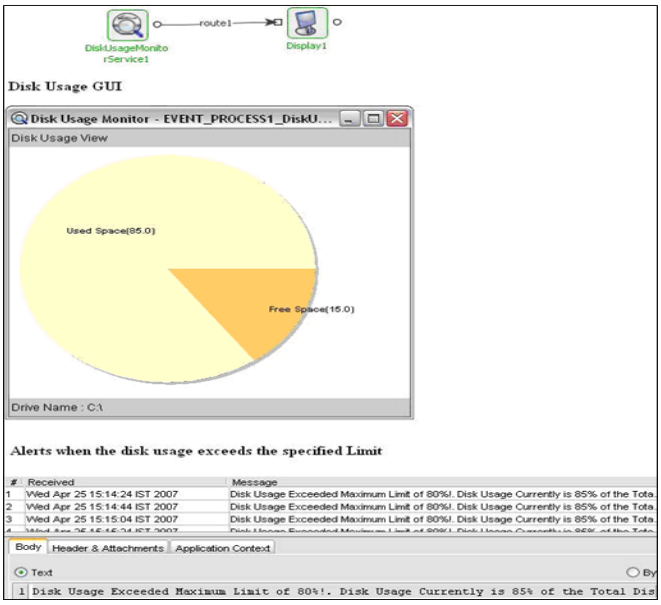


Figure 4.452: Scenario demonstration showing the Disk Usage GUI and Alerts

4.12.4.3 Useful Tips

This component cannot be launched in-memory of the peer server.

4.12.5 Display

The **Display** component is used to display messages passing through it. It reads the contents of the incoming message, displays them, and then forwards them as is. Also it is used to display an exception/error message which comes from the exception port of a component

Note: This component cannot be launched in-memory of the peer server.

4.12.5.1 Configuration and Testing

The Display component can be configured using its Custom Proper Sheet wizard.

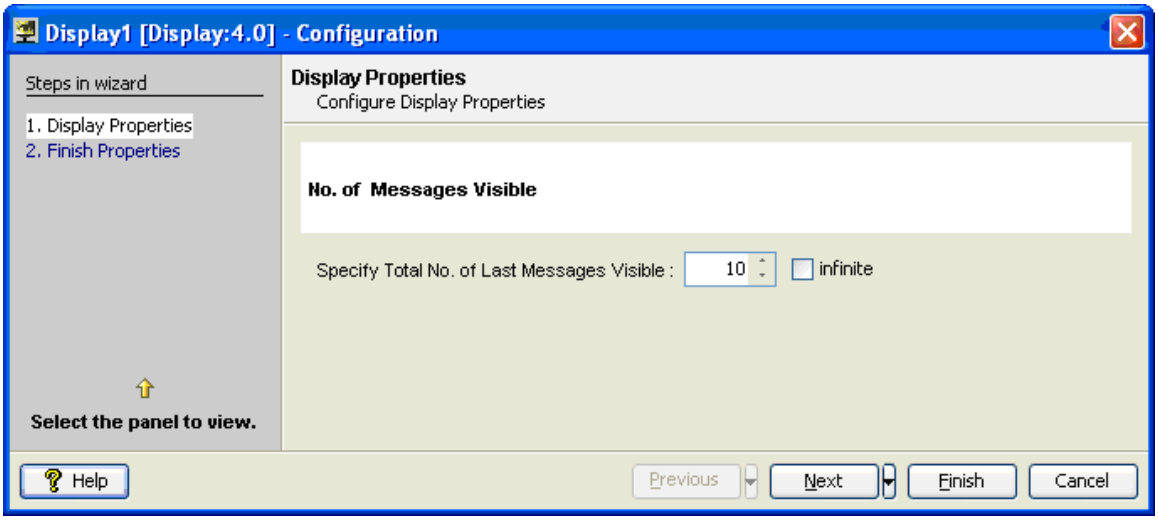


Figure 4.453: Sample Display configuration

4.12.5.2 Functional Demonstration

4.12.5.2.1 Scenario 1

Send input message using **Chat** component, the output message from the **Chat** component is displayed in the **Display** component.

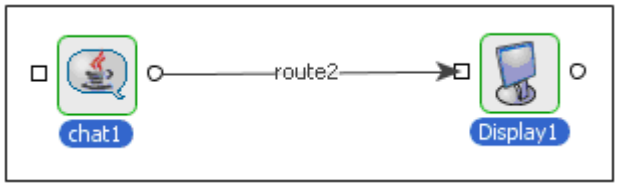


Figure 4.554: Demonstrating Scenario 1 with sample input and output

Sample Input:

Hello

Sample Output:

```
<ChatMessage><Sender><Name>FioranoESB
Demo</Name><Email>fesb@fiorano.com</Email></Sender><Message>Hello</Message></ChatMessa
ge>
```

4.12.5.2.2 Scenario 2

Send some garbage input message using Feeder to **CBR** component, **Display** component is show the Exception message sent by the **CBR**.

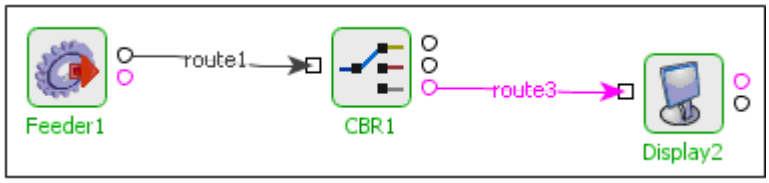


Figure 4.455: Garbage input message

Sample Input:

Garbage

Sample Output:

```
<?xml version="1.0" encoding="UTF-8"?><ns1:Error
xmlns:ns1="http://www.fiorano.com/fesb/activity/fault"><errorCode/><errorMessage>error
_processing_messagefiorano.jms.common.FioranoException: Invalid input
XML</errorMessage><errorDetail/></ns1:Error>
```

4.12.5.3 Use Case Scenario

In a revenue control packet scenario transaction files are read and then transformed, after which DB is updated. The result is shown in the **Result_Display** component. In case of error, error message is displayed in the **Error_Display**

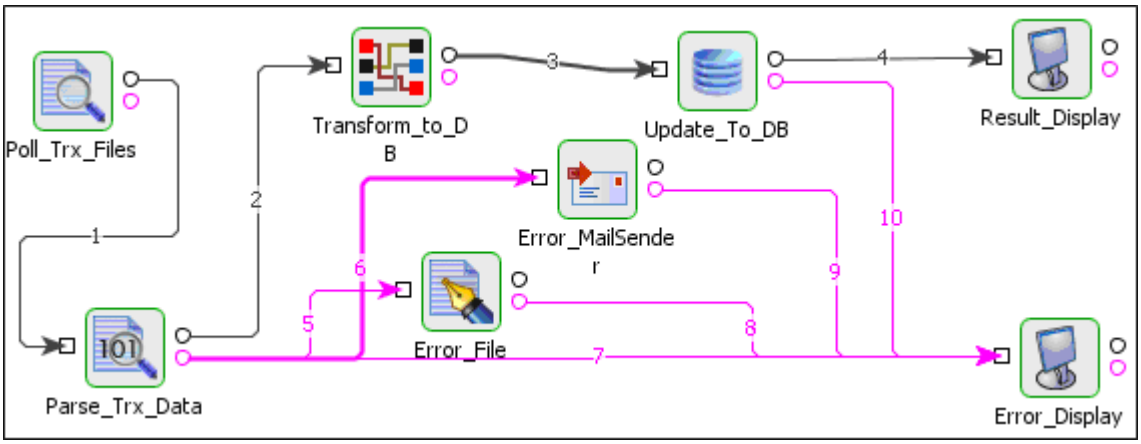


Figure 4.456: Revenue control packet scenario

The event process demonstrating this scenario is bundled with the installer.

Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Studio.

4.12.6 Encryption

The Encryption component is used for encrypting data, based on a key (that is entered by the user) and an algorithm.

This component also can handle digest data. If the message is signed/digested by the sender, the recipient can verify if the message is from the correct sender and if the message didn't change during transmission.

4.12.6.1 Configuration and Testing

The component has the following attributes which can be configured from its Configuration Property sheet. Figure 1 illustrates the panel with expert properties

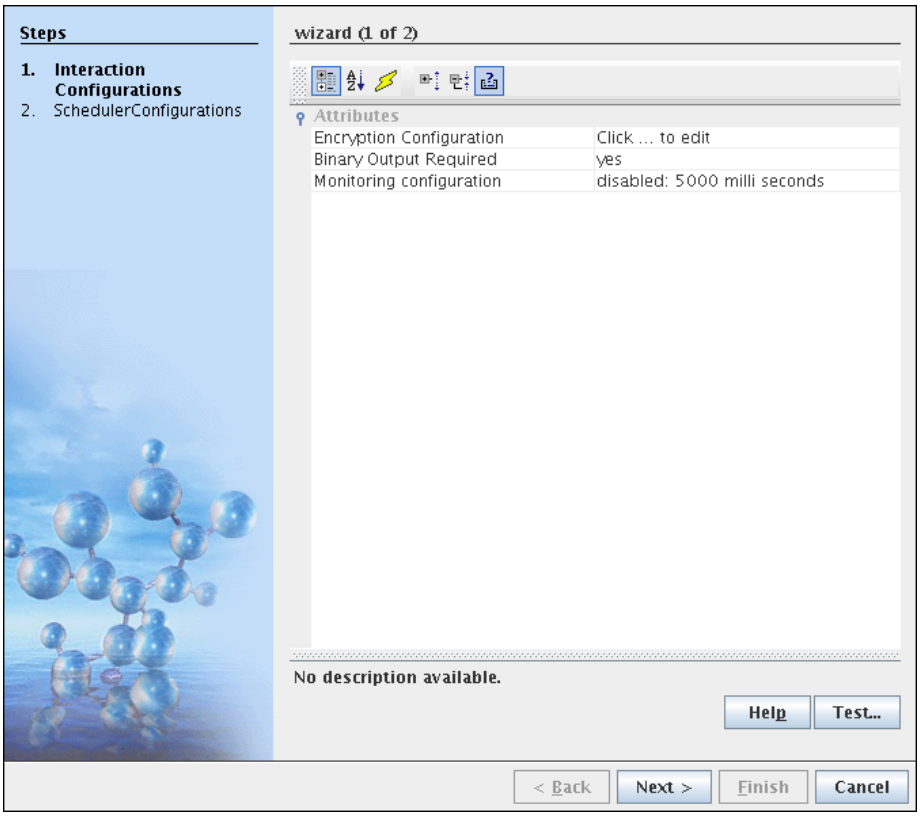


Figure 1: Configurable properties for Encryption component

4.12.6.1.1 Encryption Configuration

Encryption Details consists Data Encryption Information and Signing Information .

Data Encryption Details

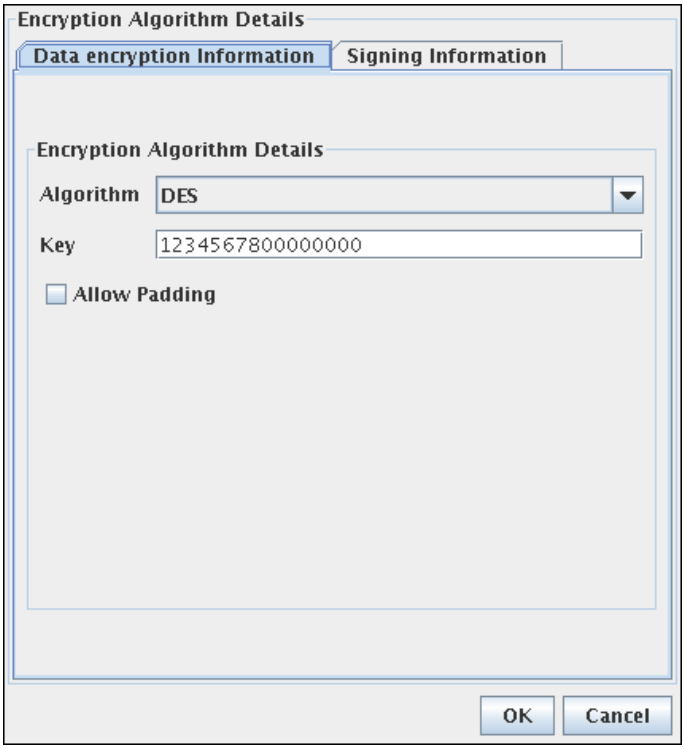


Figure 2: Encryption Algorithm Details

- **Algorithm**: The supported algorithms are DES, PGP, Base64, TripleDES, AES256, AES128, RC2-40, RC2-64, and RC2-128.
- **Key**: Specify key which is used to encrypt the incoming data. Specify 32 digit key for AES256, 16 digit key for AES128, 16 digit key for DES, 24 digit key for TripleDES, 5 digit key for RC2_40, 8 digit key for RC2_64 and 16 digit key for RC2_128
- **Allow Padding**: Choose yes to allow padding to the key to make exactly the value required for the mentioned algorithm. It has to be choosed only when we know that encryted data key is also padded.
- **Passphrase**: Required for signing in case of PGP algorithm.
- **Public Key File**: Specify public key file used in case of PGP Algorithm
- **Private Key File**: Specify private key file used in case of PGP Algorithm

Signing Information

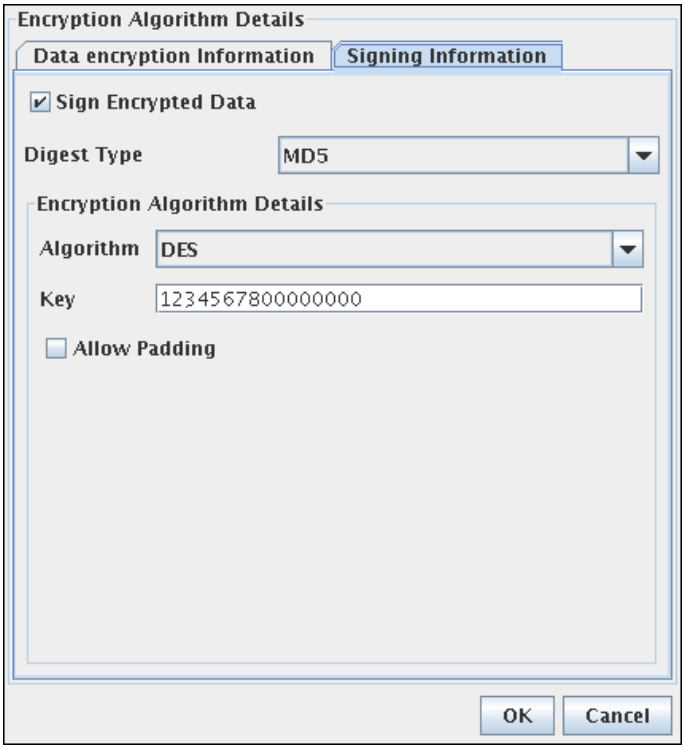


Figure 3: Digest type used and Encryption algorithm used for verifying digested data can be provided as given above.

- **Sign Encrypted Data**: Select to verify only when you know that encrypted data has been signed.
- **Digest Type**: The supported algorithms for signing/digesting data are are MD5 and SHA-1.
- **Encryption Algorithm Details**: Specify the encryption details which are used to encrypt the signed data. It is done in the same way as shown in Data Encryption Details. Algorithm for data encryption and digest data encryption need not be same.

4.12.6.1.2 Binary Output Required

Click yes if Binary Output is needed. Else encrypted data will be in xml format.

Additional Configuration

- **Cleanup resources**: Closes Interaction after each request
- **Error Handling**: Specify error Handling configuration.

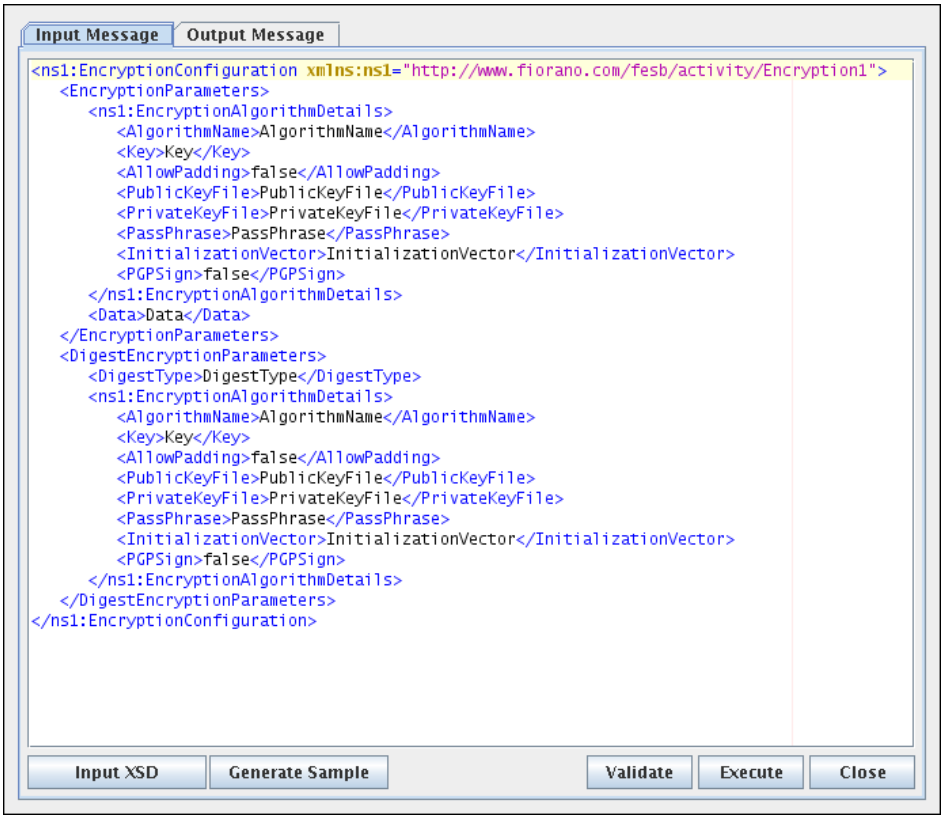
4.12.6.1.3 Monitoring configuration

When enabled, component publishes USER_EVENTs containing following statistics to FPS_USER_EVENTS_TOPIC: Minimum Execution Time: Least amount of time taken to process any message during the last publish interval period Maximum Execution Time: Highest amount of time taken to process any message during the last publish interval period Count: Number of messages processed during the last publish interval period. Through put: Through put of the component during the last publish interval period.

4.12.6.1.4 Using Named Configurations

Changes are done in eStudio wrt to Named Configurations.With the implementation of Named Configurations, Encryption Configuration can be saved and retrieved.

Input to Encryption component can be in string or xml format. XML format is shown in Figure 4. The values of Encryption Algorithm Parameters and Digest Parameters can be mentioned or else these values are taken from cps.



Input Message | Output Message

```
<ns1:EncryptionConfiguration xmlns:ns1="http://www.fiorano.com/fesb/activity/Encryption
   <EncryptionParameters>
      <ns1:EncryptionAlgorithmDetails>
         <AlgorithmName>AES256</AlgorithmName>
         <Key>12345678901234567890123456789012</Key>
      </ns1:EncryptionAlgorithmDetails>
      <Data>INPUTDATA</Data>
   </EncryptionParameters>
   <DigestEncryptionParameters>
      <DigestType>MD5</DigestType>
      <ns1:EncryptionAlgorithmDetails>
         <AlgorithmName>TripleDES</AlgorithmName>
         <Key>1234567890ABCDEF12345678</Key>
      </ns1:EncryptionAlgorithmDetails>
   </DigestEncryptionParameters>
</ns1:EncryptionConfiguration>
```

Input XSD | Generate Sample | Validate | Execute | Close

Figure 4: Sample input

Input Message | Output Message

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:EncryptedDataDetails xmlns:ns1="http://www.fiorano.com/fesb/activity/Encryption1">
   <EncryptedData>04CXnNa47wHA653m52zpWQ==</EncryptedData>
   <DigestDataDetails>
      <DigestedData>jtvGPeyc0ol8DDra6LLrpsdQf3XGZN/gB3ZS9GS/vqQoKCvt7dFJLw==</DigestedD
      <DigestType>MD5</DigestType>
   </DigestDataDetails>
</ns1:EncryptedDataDetails>
```

Output XSD | Validate | Close

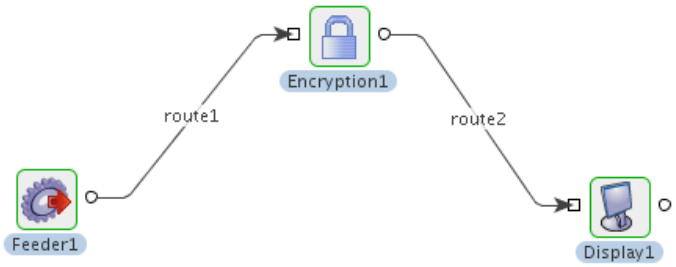
Figure 5: Sample output

4.12.6.3 Functional Demonstration

4.12.6.3.1 Scenario 1

Encryption of data received from input.

Configure the Encryption component as described in *Configuration and Testing* section and use feeder and display component to send sample input and check the response respectively.




```
Data
```

Sample Input

```
Message type :TextMessage

           0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h 3B 80 97 9C D6 B8 EF 01 C0 EB 9D E6 E7 6C E9 59
00000010h
00000020h
00000030h
00000040h
00000050h
00000060h
00000070h
```

Sample Output in binary format

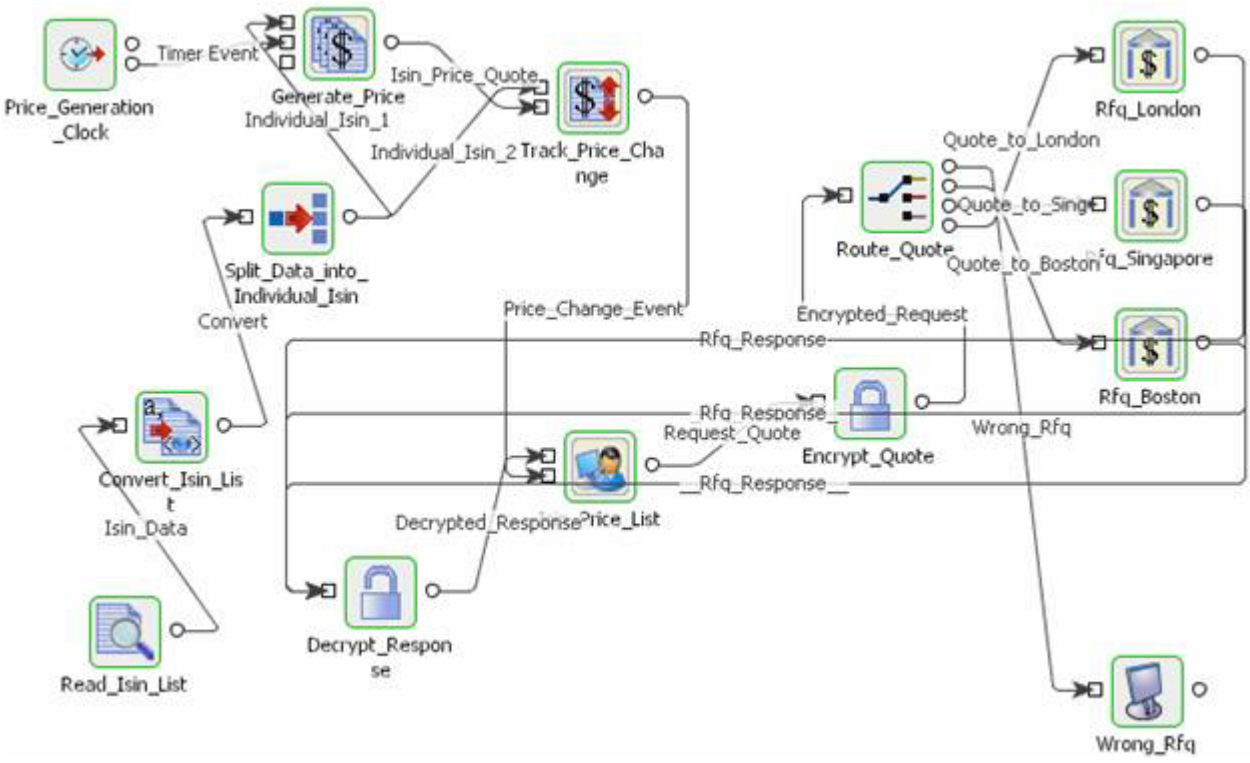
```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:EncryptedDataDetails xmlns:ns1="http://www.fiorano.com/fesb/activity/Encryption1">
   <EncryptedData>04CXnNa47wHA653m52zpWQ==</EncryptedData>
   <DigestDataDetails>
      <DigestedData>jtvGPeycOol8DDra6LLrpsdQf3XGZN/gB3ZS9GS/vqQoKCvt7dFJLw==</DigestedData>
      <DigestType>MD5</DigestType>
   </DigestDataDetails>
</ns1:EncryptedDataDetails>
```

Sample Output in xml format

Figure 6: Demonstrating Scenario 1 with sample input and output.

4.12.6.4 Use Case Scenario

In a bond trading scenario, request for quotes (RFQ) are sent in encrypted form to the appropriate exchanges and the received encrypted responses are decrypted at the user end



The event process that demonstrates this scenario is bundled with the installer. Note encryption and decryption components may not be present.

Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Studio.

4.12.7 Feeder

The Feeder component is used to feed data to any other component connected to its output port.

Note: This component cannot be launched in-memory of the peer server.

4.12.7.1 Configuration and Testing

We can have the Feeder send text messages or XML messages to the component(s) connected to its output port. Figure 4.457 shows the CPS of the Feeder configured to send XML messages defined by the provided XSD.

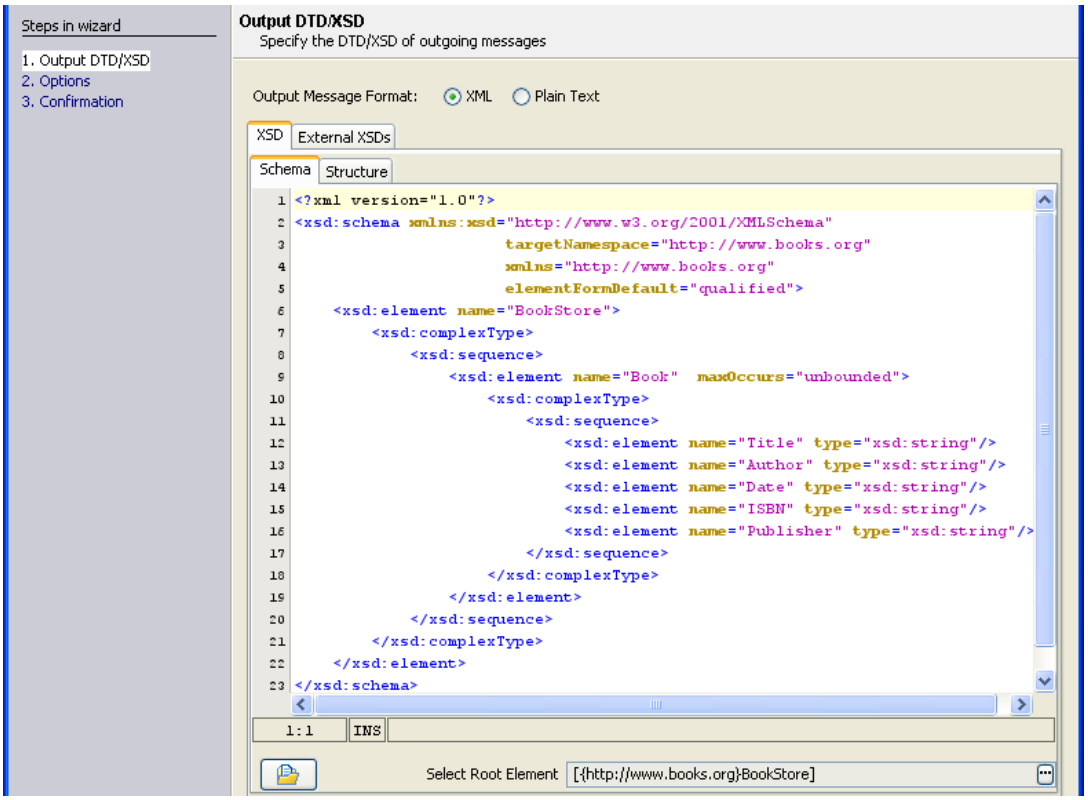


Figure 4.457: Specifying the XSD of the outgoing messages

Feeder provides the feature of auto generation of XML messages corresponding to the XSD provided in the first panel of the CPS. Figure 4.458 shows a sample XML generated for the XSD provided in Figure 4.457. If a different XML is specified, it can be validated against the XSD using the Validate button visible in Figure 4.458.

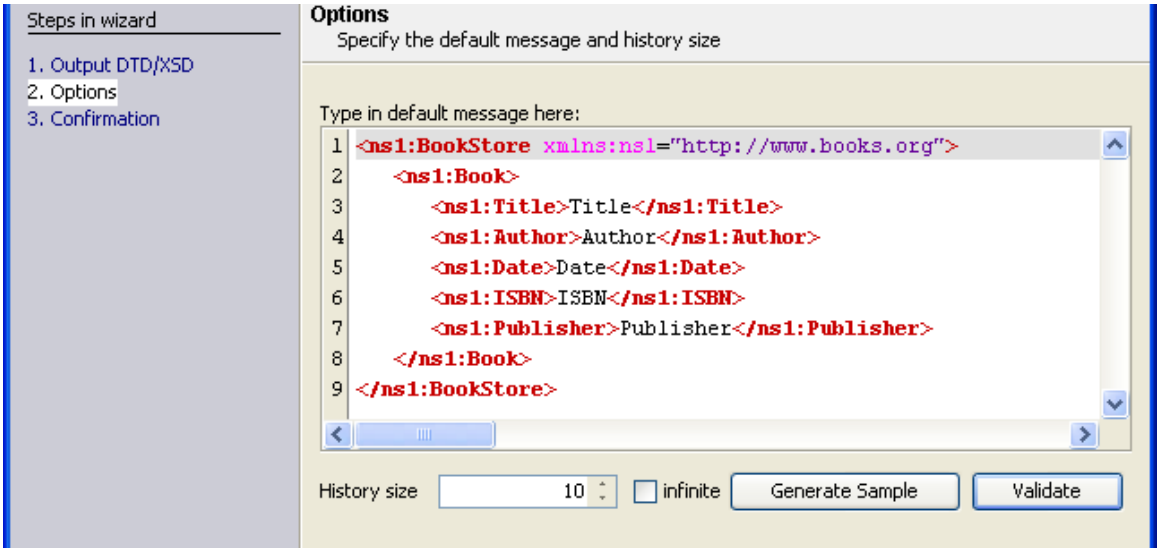


Figure 4.458: Auto generation of sample XML messages

4.12.7.2 Input and Output

4.12.7.2.1 Input Schema

Feeder does not have any input schema as it does not have an input port.

4.12.7.2.2 Output Schema

The XSD provided in the CPS is itself the output schema of the component.

4.12.7.3 Functional Demonstration

4.12.7.3.1 Scenario 1

Sending XML messages corresponding to the provided XSD.

Configure the Feeder as shown in Figure 4.457 and Figure 4.458. Connect a Display to its output port as shown in Figure 4.459.

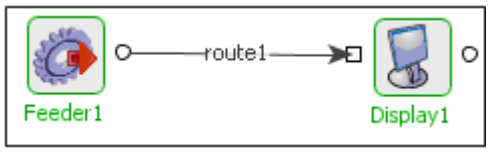


Figure 4.459: Sample event process depicting the functionality of Feeder

Now, we have two pop-up frames on the screen one each for Feeder and Display components. Click the **Send** button in the Feeder frame shown in Figure 4.460 to get the output in the Display component as shown in Figure 4.461.

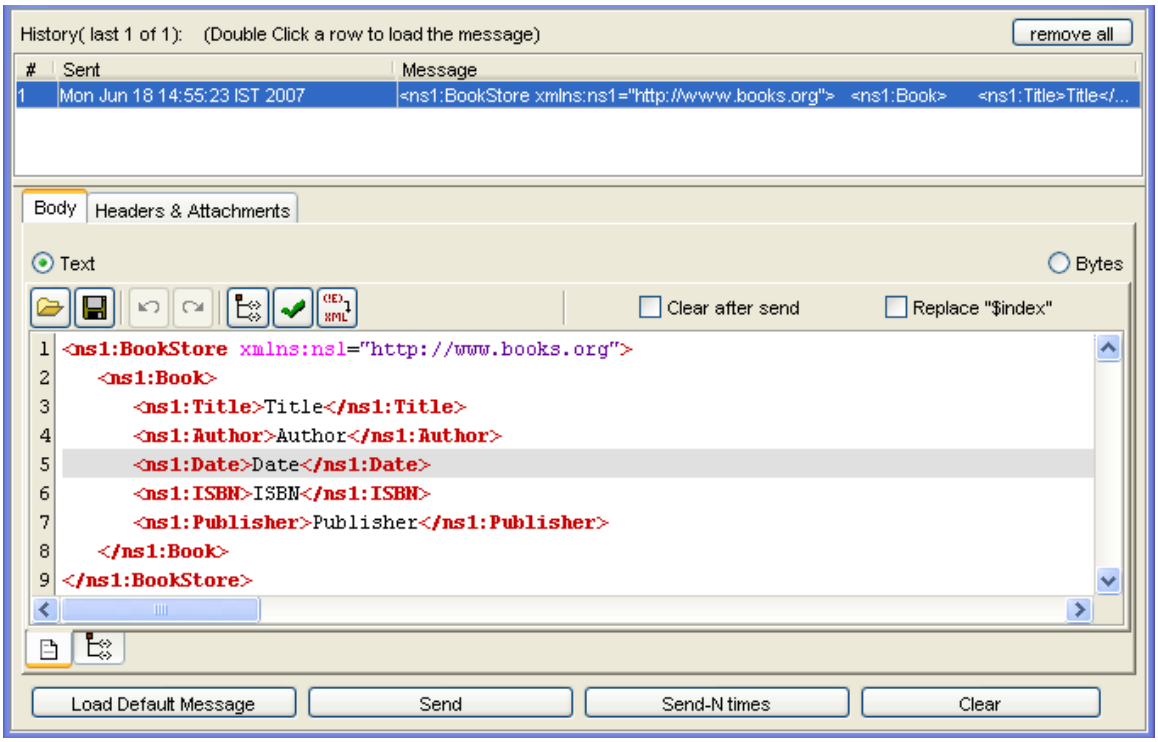


Figure 4.460: Sending a sample XML message from the Feeder

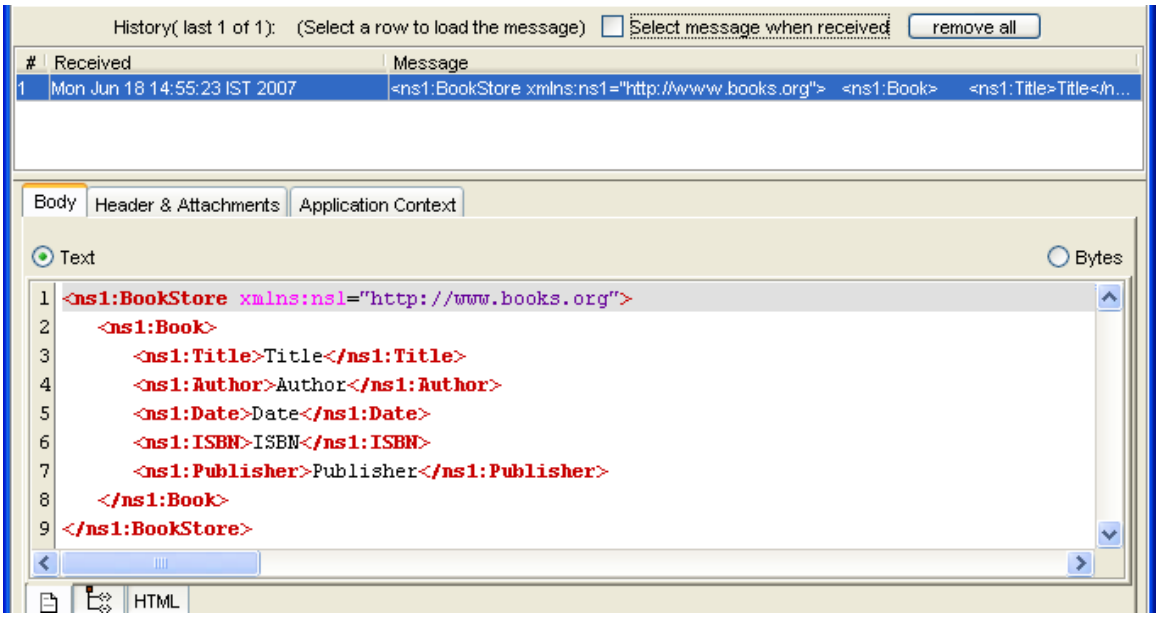


Figure 4.461: Display component showing the XML message received from Feeder

4.12.7.4 Use Case Scenario

In the sample event process Hospitality Service, the Feeder component is used to send the operation type.

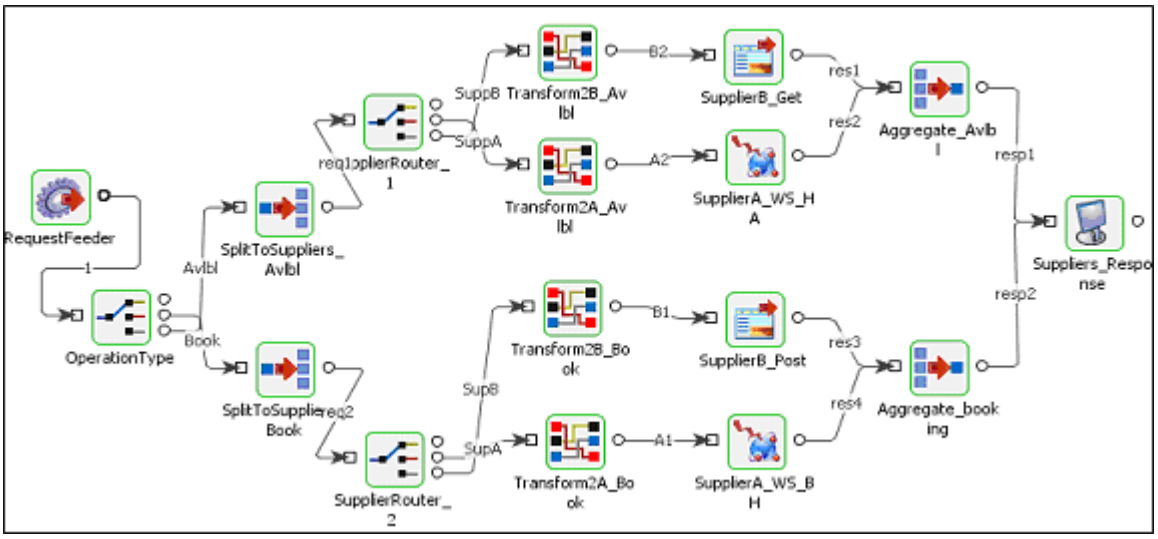


Figure 4.462: A sample event process

4.12.7.5 Useful Tips

Feeder is very useful in testing the functionality of components by feeding in the data they require.

This component cannot be launched in-memory of the peer server.

The Replace "$index" feature of the Feeder could be used to auto generate unique keys for every outgoing message. The $index in the outgoing message would be replaced by the number of the message that is currently being sent. For example, for the message shown in Figure 4.463, the first outgoing message would be as shown in Figure 4.464.

Clear after send Replace "$index"

```
1  <ns1:Cache xmlns:ns1="http://www.fiorano.com/SOA/bc/cache">
2     <ns1:CacheEntry>
3        <ns1:Keys>
4           <ns1:name0>$index1</ns1:name0>
5        </ns1:Keys>
6        <ns1:Data>
7           <ns1:name2>name2</ns1:name2>
8           <ns1:name1>name1</ns1:name1>
9        </ns1:Data>
10    </ns1:CacheEntry>
11    <ns1:CacheEntry>
12       <ns1:Keys>
13          <ns1:name0>$index2</ns1:name0>
14       </ns1:Keys>
15       <ns1:Data>
16          <ns1:name2>name2</ns1:name2>
17          <ns1:name1>name1</ns1:name1>
18       </ns1:Data>
19    </ns1:CacheEntry>
20 </ns1:Cache>
```

Figure 4.463: Using the feature 'Replace $index'

```
<ns1:Cache xmlns:ns1="http://www.fiorano.com/SOA/bc/cache">
   <ns1:CacheEntry>
      <ns1:Keys>
         <ns1:name0>1</ns1:name0>
      </ns1:Keys>
      <ns1:Data>
         <ns1:name2>name2</ns1:name2>
         <ns1:name1>name1</ns1:name1>
      </ns1:Data>
   </ns1:CacheEntry>
   <ns1:CacheEntry>
      <ns1:Keys>
         <ns1:name0>1</ns1:name0>
      </ns1:Keys>
      <ns1:Data>
         <ns1:name2>name2</ns1:name2>
         <ns1:name1>name1</ns1:name1>
      </ns1:Data>
   </ns1:CacheEntry>
</ns1:Cache>
```

Figure 4.464: The first outgoing message when the message is sent

4.12.8 PrintPDF

The PrintPdf component can be used to print a PDF file on to a local or remote printer. This component uses Adobe executable to print a PDF

Points to note

- This component can be used on Windows platform only.
- This component cannot be launched manually on the peer server.

4.12.8.1 Configuration and Testing

The following parameters can be configured from the Custom Property Sheet of the component.

Name	Description
Error handling Configuration	Allows to configure actions to be taken when an exception occurs during execution
Adobe Executable	The Executable path of the adobe application.
Default Printer	The Printer Name is used as a default when no name is specified in the input.
Wait Time	The time in milli seconds the component should wait for the printer to print the file.
Validate Printer	If yes, the printer name specified is validated.

4.12.8.1.1 Error Handling Configuration

The remedial actions to be taken when a particular error occurs can be configured here.

Click on the ellipsis button against this property to configure Error Handling properties for different types of Errors.

By default, the options **Log to Error Logs** and **Send to error port** are enabled.

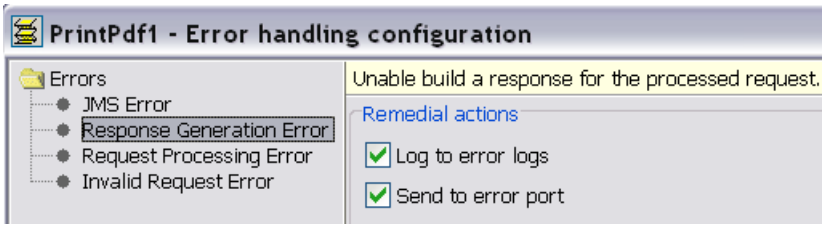


Figure 1: Error handling configuration

4.12.8.1.2 Using Named Configurations

Changes are done in eStudio wrt to Named Configurations. With the implementation of Named Configurations, Component Configuration panel is changed as shown in Figure below:

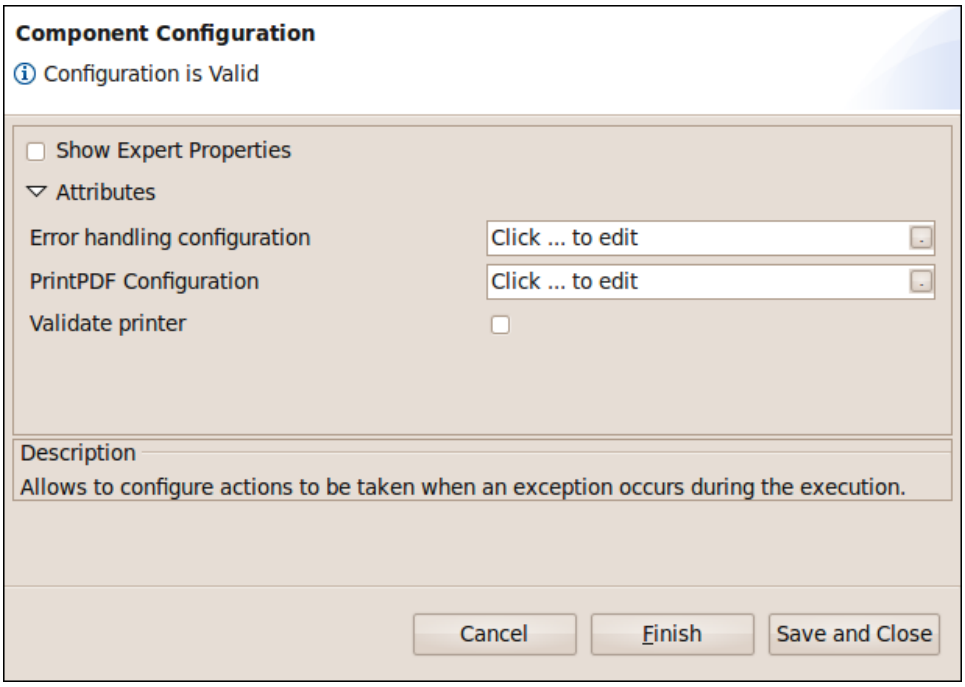


Figure: Modified Component Configuration

Some properties are grouped as one property in the modified configuration panel. Adobe Executable, Default Printer and Wait Time which were present previously in the Component Configuration panel are grouped under the name PrintPdf Configuration.

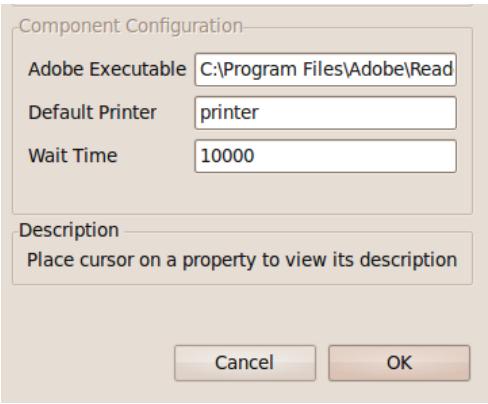


Figure: PrintPdf Configuration

4.12.8.1.3 Sample Input and Output

Configure the parameters mentioned in section 4.12.8.1 Configuration and Testing. A Feeder component can be used to provide input and a Display component to receive the output from the PrintPdf component.

The component is configured to print a PDF file to a local printer in this example.

Launch the flow and send the sample input through Feeder component.

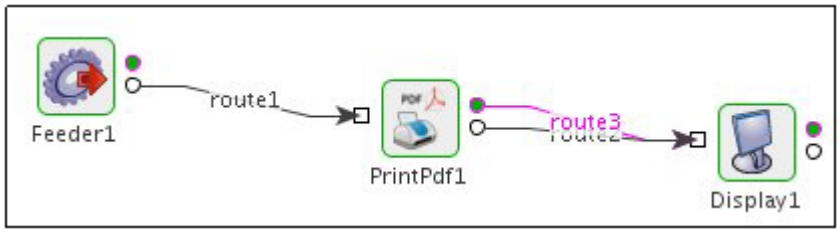
The PDF file is opened using the specified Adobe executable before it is sent to the printer. The file is closed automatically after the specified wait time and the component sends an XML message which contains FilePath, PrinterName and Message elements on to its output.

An error document will be sent to the component's ON_EXCEPTION port if one of the following cases is encountered

- If the printer name specified is not valid
- If the path of the PDF file given in the input is not valid
- If the path of the Adobe executable provided is not valid

Note: There will be no error document if the print is not successful

Figure 2 shows the sample input and output for the scenario mentioned.



Sample Input sent from Feeder

```
<ns1:PdfPrintMessage xmlns:ns1="http://www.fiorano.com/fesb/PrintPDF/In">
   <ns1:FilePath>D:\Test\PDFs\new.pdf</ns1:FilePath>
   <ns1:PrinterName>printer</ns1:PrinterName>
   <ns1:WaitTime>10000</ns1:WaitTime>
</ns1:PdfPrintMessage>
```

Output Generated

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:PrintPDF xmlns:ns1="http://www.fiorano.com/fesb/PrintPDF/Out">
   <ns1:FilePath>D:\Testing\invoi.pdf</ns1:FilePath>
   <ns1:PrinterName>hp LaserJet 3015 PCL 6</ns1:PrinterName>
   <ns1:Message>Print command executed successfully.</ns1:Message>
</ns1:PrintPDF>
```

4.12.8.2 Recommendations

- Configure the component for a local printer connected to the production box where FPS runs.
- Use Adobe Version 8.0 or later for printing the file.

4.12.8.3 Limitations

- When the print fails, it does not report the failure
- After a system restart, print does not work for some initial tries.
- FPS on which the component is running should not be launched as NTService.
- The system has to be restarted if printing stops after processing many requests.

4.12.9 NEFTConverter

The NEFTConverter component transforms data from NEFT format to XML format and vice versa.

4.12.9.1 Configuration and Testing

The NEFTConverter component can be configured using its Custom Property Sheet wizard.

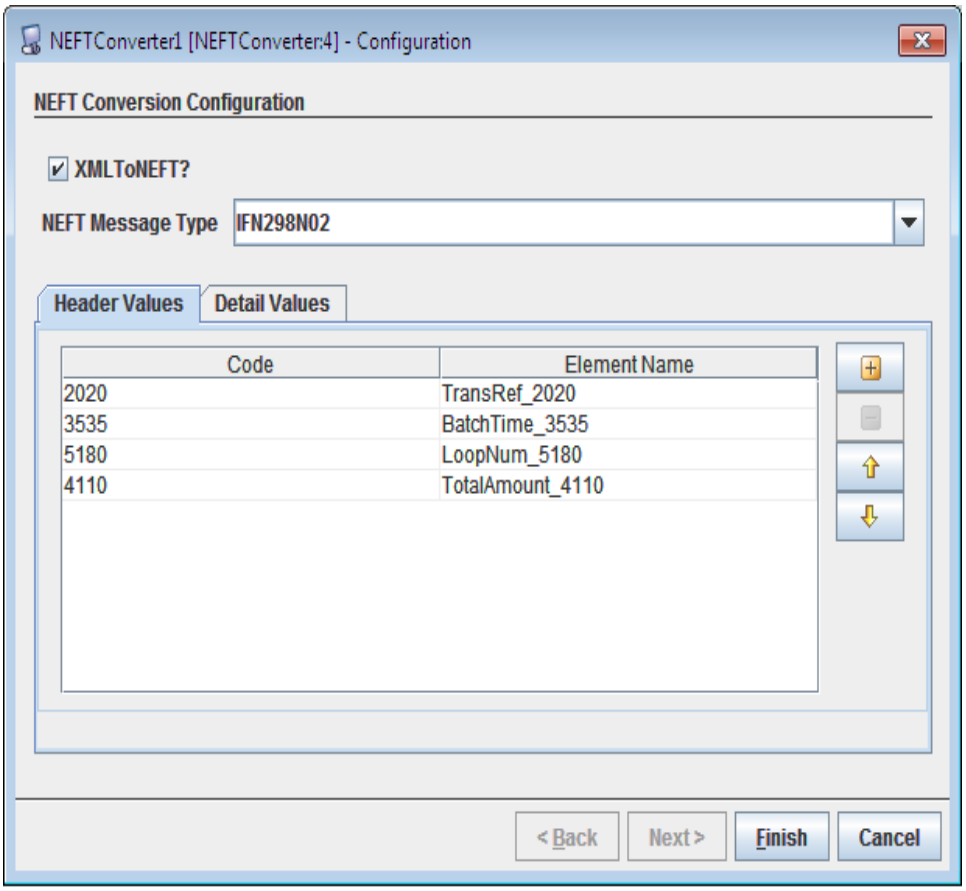


Figure 1: Sample NEFTConverter configuration

4.12.9.1.1 NEFTConverter CPS

The window mainly contains three parts

- Conversion type
- NEFT Message type
- Values panel

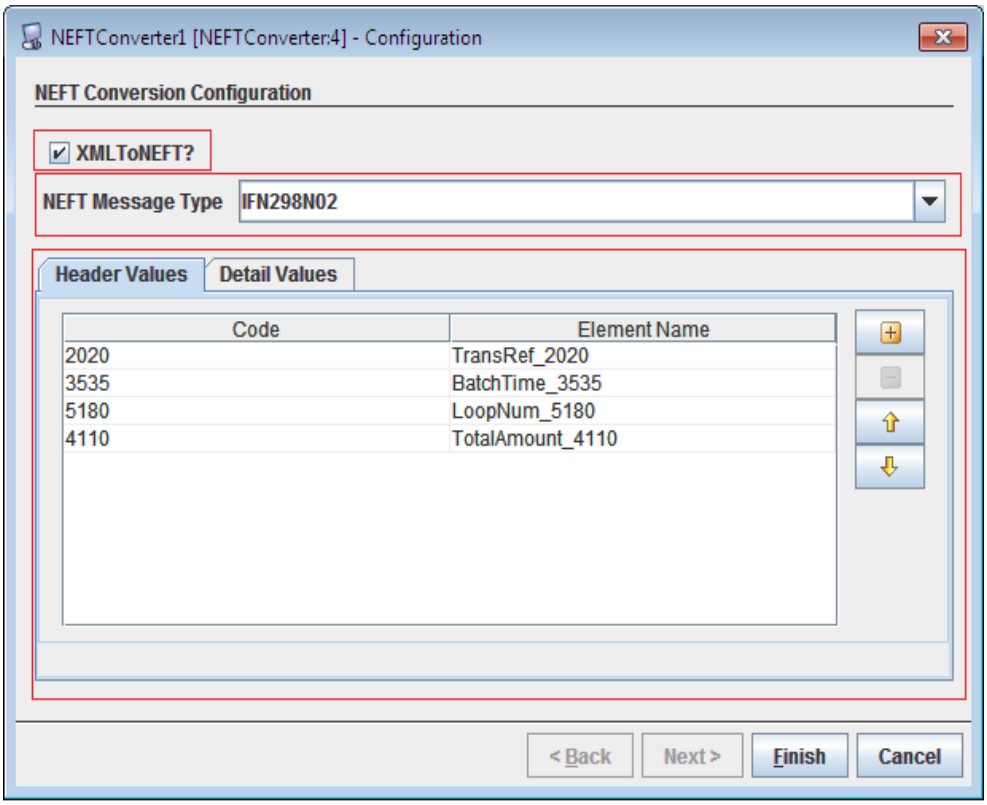


Figure 2: CPS Panels

Conversion type

This checkbox is used to choose the type of conversion i.e., XMLToNEFT or NEFTToXML. By default it is enabled, which means that the type of conversion is from XML to NEFT format.

NEFT Message type

This dropdown lists the available NEFT Message formats. When a Message format is selected, the corresponding default Header and Detail values are loaded into the Values panel.

Values panel

This panel has two tabs, Header values and Detail values. When selected, each tab displays the corresponding values of the respective fields.

4.12.9.2 Functional Demonstration

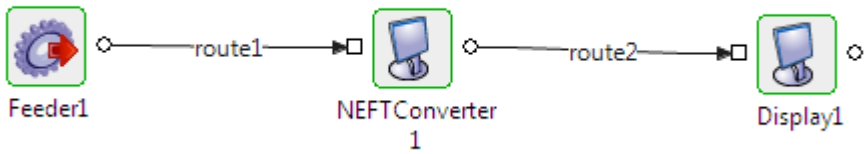


Figure 3: Demonstrating scenarios with sample input and output.

4.12.9.2.1 Scenario 1

Connect Feeder component to the input port and Display component to the output port of the NEFTConverter. Configure the component to XMLtoNEFT mode, and set the schema in the Feeder input.

Sample Input

```
<ns1:neftfield xmlns:ns1="http://www.fiorano.com/fesb/activity/neftfield_IFN298N02">
<ns1:header>
<ns1:TransRef_2020>TransRef_2020</ns1:TransRef_2020>
<ns1:BatchTime_3535>BatchTime_3535</ns1:BatchTime_3535>
<ns1:LoopNum_5180>LoopNum_5180</ns1:LoopNum_5180>
<ns1:TotalAmount_4110>TotalAmount_4110</ns1:TotalAmount_4110>
</ns1:header>
<ns1:detail>
<ns1:TransRef_2020>TransRef_2020</ns1:TransRef_2020>
<ns1:RelatedRefNo_2006>RelatedRefNo_2006</ns1:RelatedRefNo_2006>
<ns1:SNDBankIFSC_5756>SNDBankIFSC_5756</ns1:SNDBankIFSC_5756>
<ns1:SNDcusAccType_6305>SNDcusAccType_6305</ns1:SNDcusAccType_6305>
<ns1:SNDcusAccNo_6021>SNDcusAccNo_6021</ns1:SNDcusAccNo_6021>
<ns1:SNDcusAccName_6091>SNDcusAccName_6091</ns1:SNDcusAccName_6091>
<ns1:SNDcusEmailMob_5629>SNDcusEmailMob_5629</ns1:SNDcusEmailMob_5629>
<ns1:OriginRemittance_7002>OriginRemittance_7002</ns1:OriginRemittance_7002>
<ns1:BeniBranchIFSC_5569>BeniBranchIFSC_5569</ns1:BeniBranchIFSC_5569>
<ns1:BeniCusType_6310>BeniCusType_6310</ns1:BeniCusType_6310>
<ns1:BeniCusAccNo_6061>BeniCusAccNo_6061</ns1:BeniCusAccNo_6061>
<ns1:BeniCusAccName_6081>BeniCusAccName_6081</ns1:BeniCusAccName_6081>
<ns1:BeniCusputress_5565>BeniCusputress_5565</ns1:BeniCusputress_5565>
<ns1:RemittanceInfo_7495>RemittanceInfo_7495</ns1:RemittanceInfo_7495>
<ns1:ReasonCode_6346>ReasonCode_6346</ns1:ReasonCode_6346>
<ns1:RejectionReason_6366>RejectionReason_6366</ns1:RejectionReason_6366>
<ns1:Amount_4038>Amount_4038</ns1:Amount_4038>
<ns1:ValueDate_3380>ValueDate_3380</ns1:ValueDate_3380>
<ns1:RemmitanceDate_3375>RemmitanceDate_3375</ns1:RemmitanceDate_3375>
</ns1:detail>
</ns1:neftfield>
```

Sample Output

```
:2020:TransRef_2020
:3535:BatchTime_3535
:5180:LoopNum_5180
:4110:TotalAmount_4110
:2020:TransRef_2020
:2006:RelatedRefNo_2006
:5756:SNDBankIFSC_5756
:6305:SNDcusAccType_6305
:6021:SNDcusAccNo_6021
:6091:SNDcusAccName_6091
:5629:SNDcusEmailMob_5629
:7002:OriginRemittance_7002
:5569:BeniBranchIFSC_5569
:6310:BeniCusType_6310
:6061:BeniCusAccNo_6061
:6081:BeniCusAccName_6081
:5565:BeniCusputress_5565
:7495:RemittanceInfo_7495
:6346:ReasonCode_6346
:6366:RejectionReason_6366
:4038:Amount_4038
:3380:ValueDate_3380
:3375:RemmitanceDate_3375
```

4.12.9.2.2 Scenario 2

Connect Feeder component to the input port and Display component to the output port of the NEFTConverter. Configure the component to NEFTtoXML mode, and set the NEFT Message format as the default input in the Feeder.

Sample Input

```
:2020:TransRef_2020
```

```
:3535:BatchTime_3535
:5180:LoopNum_5180
:4110:TotalAmount_4110
:2020:TransRef_2020
:2006:RelatedRefNo_2006
:5756:SNDBankIFSC_5756
:6305:SNDcusAccType_6305
:6021:SNDcusAccNo_6021
:6091:SNDcusAccName_6091
:5629:SNDcusEmailMob_5629
:7002:OriginRemittance_7002
:5569:BeniBranchIFSC_5569
:6310:BeniCusType_6310
:6061:BeniCusAccNo_6061
:6081:BeniCusAccName_6081
:5565:BeniCusputress_5565
:7495:RemittanceInfo_7495
:6346:ReasonCode_6346
:6366:RejectionReason_6366
:4038:Amount_4038
:3380:ValueDate_3380
:3375:RemmitanceDate_3375
```

Sample Output

```
<ns1:neftfield xmlns:ns1="http://www.fiorano.com/fesb/activity/neftfield_IFN298N02">
<ns1:header>
<ns1:TransRef_2020>TransRef_2020</ns1:TransRef_2020>
<ns1:BatchTime_3535>BatchTime_3535</ns1:BatchTime_3535>
<ns1:LoopNum_5180>LoopNum_5180</ns1:LoopNum_5180>
<ns1:TotalAmount_4110>TotalAmount_4110</ns1:TotalAmount_4110>
</ns1:header>
<ns1:detail>
<ns1:TransRef_2020>TransRef_2020</ns1:TransRef_2020>
<ns1:RelatedRefNo_2006>RelatedRefNo_2006</ns1:RelatedRefNo_2006>
<ns1:SNDBankIFSC_5756>SNDBankIFSC_5756</ns1:SNDBankIFSC_5756>
<ns1:SNDcusAccType_6305>SNDcusAccType_6305</ns1:SNDcusAccType_6305>
<ns1:SNDcusAccNo_6021>SNDcusAccNo_6021</ns1:SNDcusAccNo_6021>
<ns1:SNDcusAccName_6091>SNDcusAccName_6091</ns1:SNDcusAccName_6091>
<ns1:SNDcusEmailMob_5629>SNDcusEmailMob_5629</ns1:SNDcusEmailMob_5629>
<ns1:OriginRemittance_7002>OriginRemittance_7002</ns1:OriginRemittance_7002>
<ns1:BeniBranchIFSC_5569>BeniBranchIFSC_5569</ns1:BeniBranchIFSC_5569>
<ns1:BeniCusType_6310>BeniCusType_6310</ns1:BeniCusType_6310>
<ns1:BeniCusAccNo_6061>BeniCusAccNo_6061</ns1:BeniCusAccNo_6061>
<ns1:BeniCusAccName_6081>BeniCusAccName_6081</ns1:BeniCusAccName_6081>
<ns1:BeniCusputress_5565>BeniCusputress_5565</ns1:BeniCusputress_5565>
<ns1:RemittanceInfo_7495>RemittanceInfo_7495</ns1:RemittanceInfo_7495>
<ns1:ReasonCode_6346>ReasonCode_6346</ns1:ReasonCode_6346>
<ns1:RejectionReason_6366>RejectionReason_6366</ns1:RejectionReason_6366>
<ns1:Amount_4038>Amount_4038</ns1:Amount_4038>
<ns1:ValueDate_3380>ValueDate_3380</ns1:ValueDate_3380>
<ns1:RemmitanceDate_3375>RemmitanceDate_3375</ns1:RemmitanceDate_3375>
</ns1:detail>
</ns1:neftfield>
```

4.13 Web

The Web category consists of components like HTTPAdapter, HttpReceive, HttpStub, and SimpleHTTP. The following section describes each component.

4.13.1 HTTPAdapter

The HTTPAdapter component enables the user to get content from an external HTTP Server (Web Server).

The Hyper Text Transfer Protocol (HTTP) is one of the most widely used protocols on the Internet today. Various Event Processes are being hosted on the World Wide Web, due to which Event Process developers face the challenge of integrating their existing solutions to work with HTTP. The Fiorano HTTPAdapter component helps Event Process developers to achieve this task with minimum knowledge of this protocol.

The Fiorano HTTPAdapter component enables Event Processes to use either the Get or the Post method. This component can function in conjunction with other Fiorano components to solve business problems in a highly productive manner.

The Get implementation

Consider a stock portfolio management company that employs several consultants who advise clients on investing their money. The Shares and Scripts section consists of three clerks who track the stock quotes in three different verticals: IT, Automobiles, and Pharmaceuticals. The task of sending requests for stock quotes can be accomplished by running multiple instances of the Get implementation. Each instance constantly provides information for a different stock quote. The results can then be sent to other components for further processing.

The Post implementation

Consider an enterprise implementing an Enterprise Resource Planning (ERP) system to automate its inventory management system across its suppliers and customers. The suppliers are generally not in the vicinity of the manufacturing facility and hence it is not possible to have a dedicated link between the two parties. In this case, the supplier can provide web access to the enterprise through a secure interface and the Post method can be used to post purchase order information at the supplier server.

Points to note

- The component never tries to guess the content type like most browsers.
- If the HTTP Server is secured using Basic or Digest authentication and required details are not provided in the component CPS, a runtime UI is displayed to capture the same.
- If the response has an application/xml content type, then in the Http Response configuration, the Response Body has to be set as "Headers". The XML can be extracted from the binary data using XSLT component.

4.13.1.1 Configuration and Testing

4.13.1.1.1 Managed Connection Factory

Connection details are configured in the **Managed Connection Factory** (**MCF**) panel. Figure 1 illustrates the panel with expert properties view enabled.

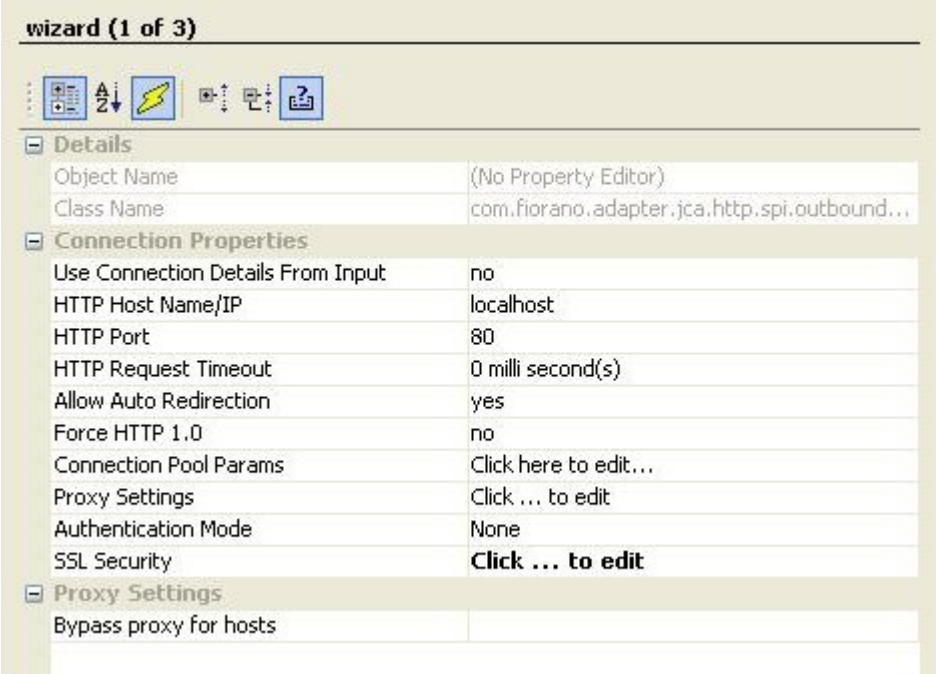


Figure 1: Connection configuration details in MCF panel

4.13.1.1.1.1 Connection Properties

Use Connection Details From Input

Parameters to create the connection can be specified in the input message when this property is set to true. If this property is selected the validation errors in the managed connection factory panel of the CPS are treated as warnings. So user can bypass this step without giving valid configuration and complete the configuration of the component. If valid properties are not provided even in the input message exception will be thrown at runtime.

HTTP Host Name/IP

The host name or the IP address where the Web Server is located. The URI relative to this host name/IP has to be configured in the request details editor in Interaction Configurations. It can also be sent in the URI element of the input message.

HTTP Port

The port number on which Web Server is running.

HTTP Request Timeout

Specifies the time for which this component will wait for an HTTP request to be acknowledged. The default value is 0, specifies infinite timeout.

Allow Auto Redirection

Determines the action to be taken if the requested page sends the request to another page.

- **yes -** This is used to allow automatic redirection of request page if required.
- **no** - Do no allow automatic redirection of request page.

Force HTTP1.0

- **yes** - HTTP1.0 protocol is used while sending requests to the web server.
- **no** - HTTP1.1 protocol is used while sending requests to the web server.

Authentication Mode

This property enables the user to specify authentication information, if any.

- **None -** This is the default selection. This is used when the requested resource does not require any authentication.
- **Basic** - A method designed to allow client program to provide credentials in the form of user name and password while making a request. These credentials are passed in plain text format using Base64 encoding.
- **Digest** - In this method, request is encrypted using MD5 cryptographic hashing and sent to web server.

  When Authentication Mode is either Basic or Digest, following credentials have to be specified.
- **Username** - The name of the user with privileges to access the protected resource.
- **Password** - The password for the user name specified above.

Proxy Settings

ByPass Proxy for Hosts

The semi-colon separated list of hostnames/IP addresses for which the request must be sent without passing the proxy specified, if any. If no proxy is defined, this property is ignored.

4.13.1.2 Interaction Configuration

The business logic configuration details are configured in the **Interaction Configurations panel**. Figure 2 illustrates the panel with expert properties view enabled.

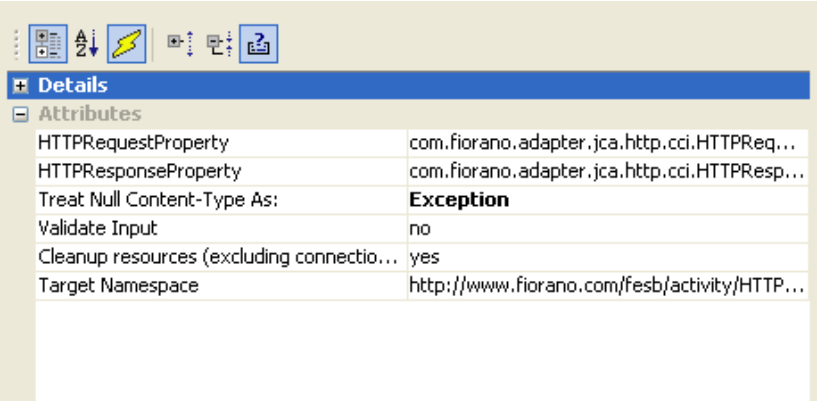


Figure 2: Business logic configuration in Interaction Configurations panel

4.13.1.2.1 Attributes

4.13.1.2.1.1 HTTPRequestProperty

This property defines properties of the request that is sent to Web Server.

Click the ellipsis button to launch an editor for providing these configurations.

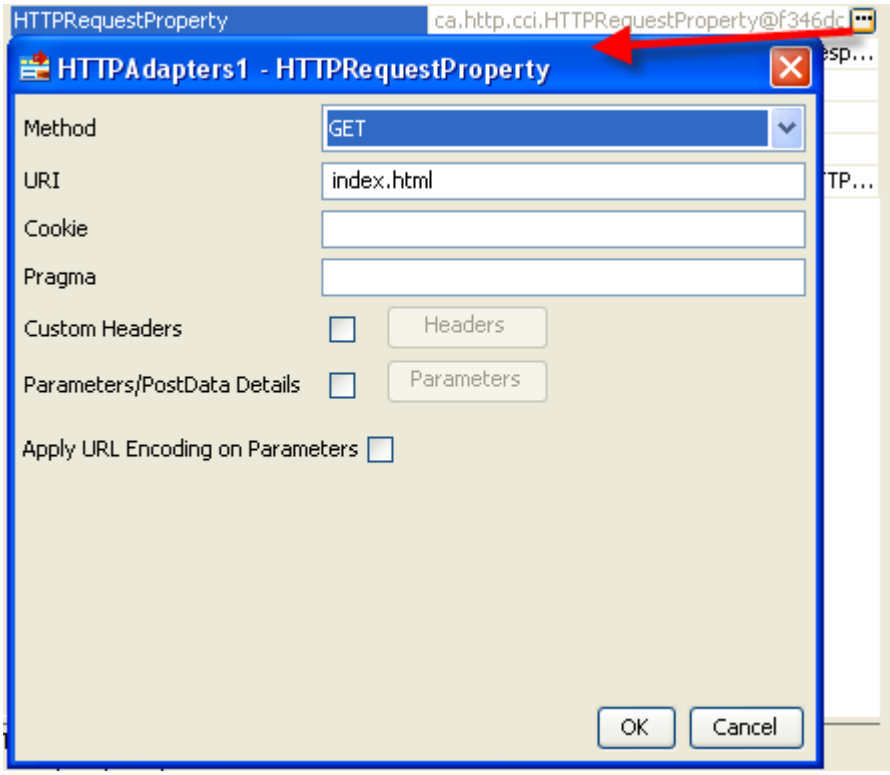


Figure 3: Launching editor for configuring Request properties

Method

The **HTTP** method that is used to send the request to the server. This is either set to GET or POST.

The usage of these methods are described in the sections The GET Implementation and The POST Implementation respectively.

The Request Properties depends on the HTTP method specified. Figure 3 shows the request properties when HTTP method is set to GET. Request properties when HTTP method is set to POST are shown in Figure 6.

URI

The Uniform Resource Identifier of the resource that is being requested by the component. This will be calculated relative to the host name/IP and port number provided in the Managed Connection Factory.

Cookie

The HTTP cookies provide the server with a mechanism to store and retrieve state information on the client application's system. This mechanism allows Web-based applications the ability to store information about selected items, user preferences, registration information, and other information that can be retrieved later.

The cookie can be specified in the form of name value pairs separated by semicolon. For example, name1=value1;name2=value2. The input schema will have an element Cookie corresponding to this property.

Note: Cookies set by the server will be discarded after servicing the client request If Connection pooling is disabled.

Pragma

The Pragma general-header field is used to include implementation- specific directives that might apply to any recipient along the request/response chain. For more information refer section 14.32 Pragma at http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html.

The input schema will have an element Pragma corresponding to this property.

Custom Headers

If there are any custom headers that have to be sent along with the request, then select the checkbox against this property and click the **Headers** button. Headers can be added using the editor as shown in the Figure 4.

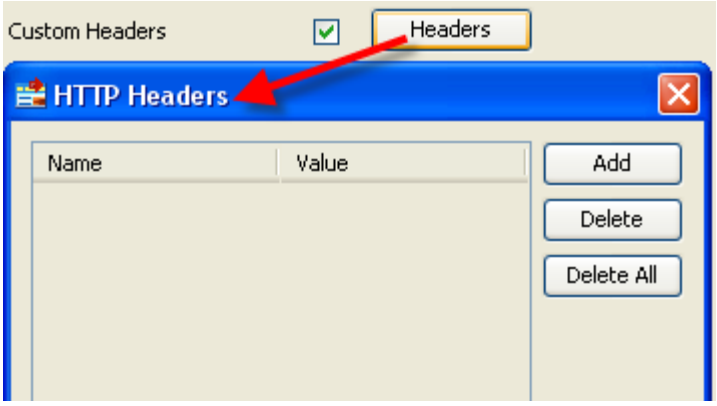


Figure 4: Custom Headers configuration

If custom headers are enabled, then the Header element of the input schema will have a child element CustomHeaders and any custom headers specified in the table are added as child elements of this CustomHeaders element.

Parameter/PostData Details

If there are any parameters to be sent along with the request, then the select the checkbox against this property and click the **Parameters** button. Parameters can be added using the editor as shown in the Figure 5.

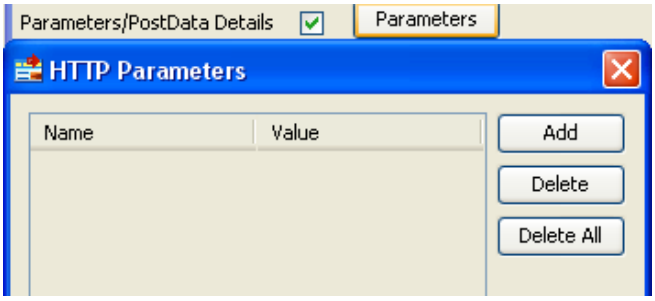


Figure 5: Parameters/PostData configuration

The schema of the component has an element entity which accepts parameters as name value pairs.

Apply URL Encoding on Parameters

If this property is selected, parameters are converted to the application/x-www-form-urlencoded MIME format (HTML form encoding) using the platform's default encoding scheme.

When **HTTP** method is set as POST, request properties appear as shown in Figure 6.

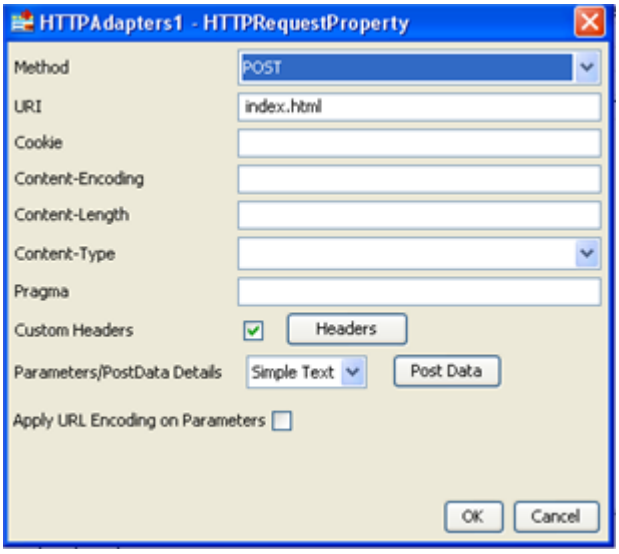


Figure 6: Request properties when HTTP method is set POST

Refer to previous section for description of properties URI, Cookie, Pragma, Custom Headers, and Apply URL Encoding on Parameters. Additional properties are described below.

Content-Encoding

The Content-Encoding header field is used as a modifier to the media-type. When present, its value indicates what additional content coding have been applied to the body, and thus what decoding mechanisms must be applied in order to obtain the media-type referenced by the Content-Type header field.

Content-Encoding is primarily used to allow a document to be compressed without losing the identity of its underlying media type.

An element Content-Encoding will be added corresponding to this property in the input schema of the component.

Content-Length

The Content-Length header field indicates the size of the body of the posted content. An element Content-Length will be added corresponding to this property in the input schema of the component.

Content-Type

The Content-Type header field indicates the media type of the body sent to the recipient. Component supports the following values:

- text/html
- image/jpeg
- model/vrml
- video/quickline
- application/java
- text/css
- text/javascript
- multipart/form-data

An element Content-Type will be added corresponding to this property in the input schema of the component.

To upload files Content-Type has to be set to multipart/form-data. When the Content-Type is set to multipart/form-data, then the generated input schema doesn't contain Content-Type element and the value specified in the CPS is used internally.

Parameters/PostData Details

When **HTTP** method is set to POST, then the post data type can be configured as shown in Figure 7.

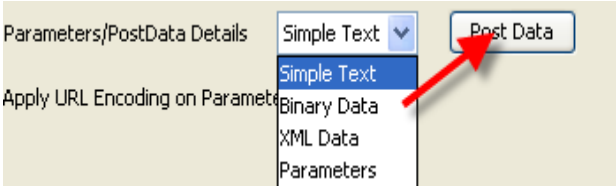


Figure 7: PostData configuration when HTTP method is set to POST

- **Simple Text**

  This is used when data to be posted is simple text. The text to be posted can be specified by using the text area that opens by clicking the button Post Data. In this case, an element text is added as a child to entity element in the schema of the input port corresponding to the text that is being posted.

- **Binary Data**

  This is used when data to be posted is in binary format. The content can be specified by clicking on the Load from File button in the UI that opens up when the Post Data button is clicked. In this case, an element BinaryData is added as child to entity element in the schema of the input port corresponding to the data that is being posted.

- **XML Data**

  This is used when data to be posted is in the form of XML conforming to a specific schema. The content can be specified in the schema editor that opens up when the Post **D**ata button is clicked. In this case, an element XMLData is added as child to entity element in the schema of the input port, whose type is same as the schema provided.

- **Parameters**

  This is used when parameters are being posted and these can be configured as shown in Figure 5.

Files

When HTTP method is set to POST and the Content-Type is set to multipart/form-data, then the files to be posted can be specified by clicking Files button. This button will be enabled when the Content-Type is set to multipart/form-data. The file parameters will be set as name-value pairs where the name is the parameter name and the value is the complete path of the file which has to be uploaded.

Figure 8: Files Button

Example: To upload a file on http://www.filefactory.com the following parameters has to be set

- redirect to 1
- enabled to 1
- file to the path of the file which has to be uploaded
- uploadStart to true
- uploadOne to true

These parameters can be determined by looking at the HTML page source.

If the input type of the parameter is file, then it has to be provided under Files and all the other parameters have to be provided under Parameters.

In the above example, parameters redirect, enabled, uploadStart, and uploadOne should be provided under Parameters and the parameter file should set under Files. These parameters can also be provided in the input message. If a parameter is specified in both the CPS and in the input, then the value provided in CPS will be overridden by the value provided in input.

The sample input is shown in Figure 24.

Refer to section Input and Output for details about the effects of these configurations on input and output structures.

4.13.1.2.1.2 HTTPResponseProperty

When a request is sent to HTTP Server, a HTTP response is obtained. Output message is created from this HTTP response based on the configuration details of this HTTPResponse property.

Click the ellipsis button to launch HTTPresponseProperty dialog box as shown in Figure 9.

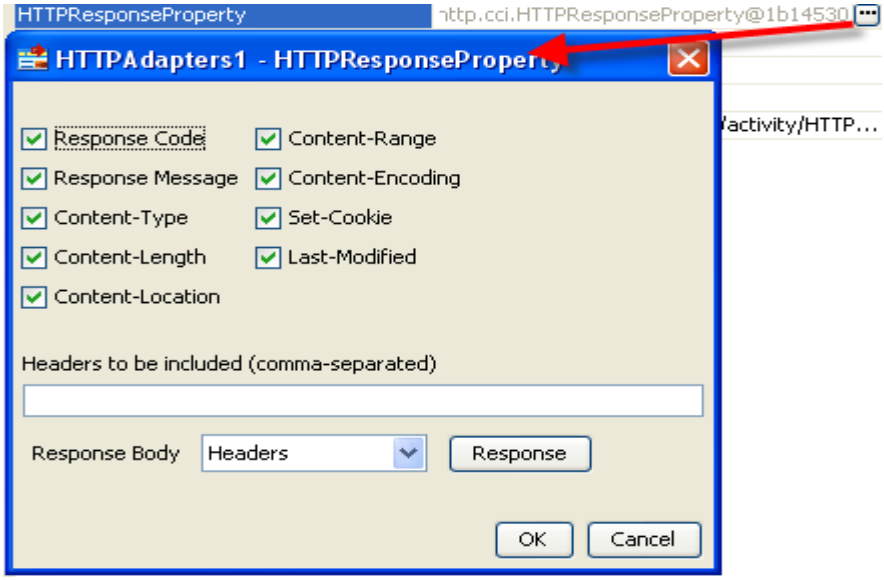


Figure 9: HTTPResponse Property configuration

Response Code

The HTTP response code that is sent from the server. If the corresponding checkbox is selected, this is included in the output message. If the response type property is chosen as XML, an element Response-Code is added to the schema of the output port. Other wise a message Header with name HTTP_Response-Code is set as message property on output message.

- **Response Message**

  The message corresponding to the response code returned. If corresponding checkbox is selected, it is included in the output message. If the response type property is chosen as XML, an element Response-Message is added to the schema of the output port. Other wise a message Header with name HTTP_Response-Message is set as message property on output message.

Headers form part of the HTTP response stream which carries information about the response stream that is sent. The headers below are standard HTTP response headers. If corresponding checkbox is selected, it is included in the output message. An element with same name as the header is added as child element of Header element in the schema of the output port if the response type property is chosen as XML. Other wise a message header with name prefixed with HTTP_ is set as message property on output message.

Content-Type

The Content-Type header field indicates the media type of the body sent to the recipient.

- **Content-Length**

  The Content-Length header field indicates the size of the body sent to the server. Any Content-Length greater than or equal to zero is a valid value.

- **Content-Location**

The Content-Location header field is used to supply the resource location for the entity enclosed in the message when that entity is accessible from a location separate from the requested resource's URI.

- **Content-Range**

  The Content-Range header is sent with a partial body to specify where in the full body the partial body should be applied.

- **Content-Encoding**

  The Content-Encoding header field is used as a modifier to the media-type. When present, its value indicates what additional content coding have been applied to the body, and thus what decoding mechanisms must be applied in order to obtain the media-type referenced by the Content-Type header field.

- **Set-Cookie**

  The Set-Cookie header is sent by the server in response to an HTTP request, which is used to create a cookie on the user's system. The Cookie header is included by the client application with an HTTP request sent to a server, if there is a cookie that has a matching domain and path.

  The Set-Cookie value can be specified in the following format name1=value1;name2=value2.

- **Last-Modified**

  The Last-Modified header field indicates the date and time at which the origin server believes the variant was last modified. The exact meaning of this header field depends on the implementation of the origin server and the nature of the original resource. For files, it may be just the file system last-modified time. For entities with dynamically included parts, it may be the most recent of the set of last-modify times for its component parts. For database gateways, it may be the last-update time stamp of the record. For virtual objects, it may be the last time the internal state changed.

  For detailed information on standard HTTP headers, please refer to http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html .

Headers to be included (comma-separated)

If there are any custom headers apart from the standard HTTP headers, those headers can be specified here in a comma-separated list.

If the custom headers are added and the response type is chosen as XML, the specified properties appear as the child elements of Custom-Headers element in the output message. If response type is chosen as Headers, then a message header with name prefixed with HTTP_ will be sent in the output message.

- **Response Body**

  This property defines the format of output message that is sent to output port. At runtime, the component converts the HTTP response obtained from HTTP Server into output message based on this property.

  - **Headers**

    This property is used to set the selected headers (both standard HTTP and custom headers) as message properties on output message. All the selected headers are read from HTTP response obtained and set on output message. The response message from HTTP response is set body of the output message.

  - **XML**

    When this property is used, all the selected headers are set as elements in output schema created by the component.

- **Response**

  The message body of the output message that is generated from HTTP response can be specified in schema editor that opens on clicking this button. This property is optional. If specified, output schema depends on the response type (Headers or XML). If response type is Headers, the schema specified here is set output schema. If response type is XML, a response message is created in the form of XML data conforming to the schema specified here and set as CDATA in the body of the output message.

  Please refer to section Input and Output for details about the effects of these configurations on input and output structures.

Treat Null Content-Type As

When the Content-Type header in HTTP response stream is null, the action to be taken is specified here as shown in Figure 10.

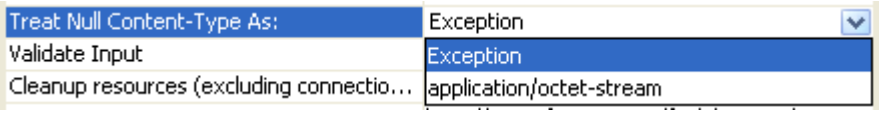


Figure 10: Configuration of null 'Content-Type'

- **Exception**

  If this property is set, an exception is sent to error port.

- **application/octet-stream**

If this property is set, then Content-Type header of the response is set to Application/octet-stream.

4.13.1.3 Input and Output

4.13.1.3.1 Input

The input schema for the component is defined based on the configuration of **HTTPRequestProperty**.

When the **HTTP method** is set to GET, the input schema varies based on the configuration of headers. Configuration of parameters does not affect the input schema generated.

- When no headers are selected, input schema is defined as shown in Figure 11 and a sample input is shown in Figure 12.

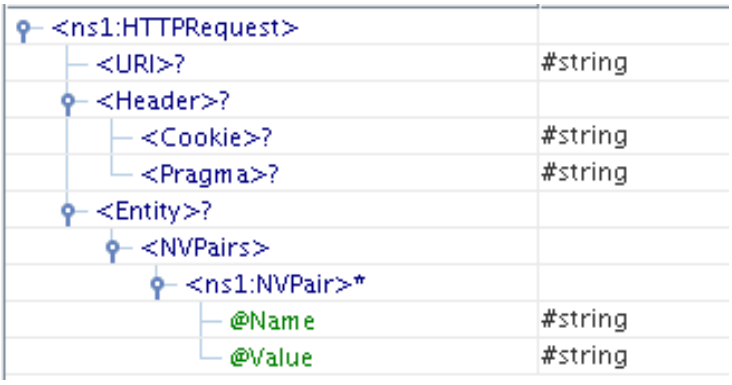


Figure 11: Schema when there are no headers

```
<ns1:HTTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/HTTPAdapters1/HT
   <URI>URI</URI>
   <Header>
      <Cookie>Cookie</Cookie>
      <Pragma>Pragma</Pragma>
   </Header>
   <Entity>
      <NVPairs>
         <ns1:NVPair Name="Name" Value="Value"/>
      </NVPairs>
   </Entity>
</ns1:HTTPRequest>
```

Figure 12: Sample input XML for schema in Figure 11

- When custom headers are added as shown in Figure 13

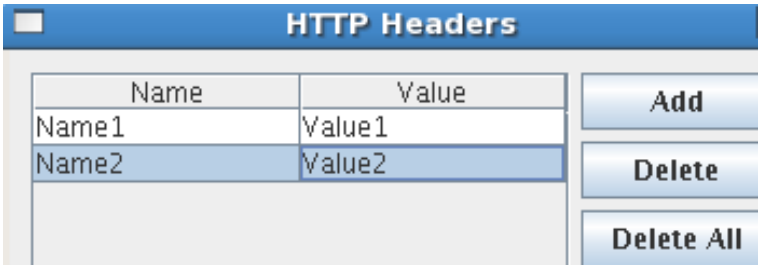


Figure 13: Sample Headers configuration

The input schema is defined as shown in Figure 14 and a sample input is shown in Figure 15.

Figure 14: Schema when custom headers are selected

```
<ns1:HTTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/HTTPAdapters1/HT
   <URI>URI</URI>
   <Header>
      <Cookie>Cookie</Cookie>
      <Pragma>Pragma</Pragma>
      <CustomHeaders>
         <Name2>Name2</Name2>
         <Name1>Name1</Name1>
      </CustomHeaders>
   </Header>
   <Entity>
      <NVPairs>
         <ns1:NVPair Name="Name" Value="Value"/>
      </NVPairs>
   </Entity>
</ns1:HTTPRequest>
```

Figure 15: Sample input XML for schema in Figure 14

Note: Headers and parameter values defined in input XML override the values set in CPS. If the corresponding header or parameters elements are not present in the input XML values from CPS are used.

When the **HTTP method** is set to Post, the input schema varies based on the configuration of headers and parameters/post data details.

- When no custom headers are added and **Parameters/PostData Details** is set to Simple Text, input schema is defined as shown in Figure 16 and a sample input is shown in Figure 17.

<ns1:HTTPRequest>	
<URI>?	#string
<Header>?	
<Cookie>?	#string
<Content-Encoding>?	#string
<Content-Length>?	#string
<Content-Type>?	#string
<Pragma>?	#string
<Entity>?	
<Text>	#string

Figure 16: Schema when parameters/postdata set to SimpleText

```
<ns1:HTTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/HTTPAdapters1/H
   <URI>URI</URI>
   <Header>
      <Cookie>Cookie</Cookie>
      <Content-Encoding>Content-Encoding</Content-Encoding>
      <Content-Length>Content-Length</Content-Length>
      <Content-Type>Content-Type</Content-Type>
      <Pragma>Pragma</Pragma>
   </Header>
   <Entity>
      <Text>Text</Text>
   </Entity>
</ns1:HTTPRequest>
```

Figure 17: Sample input XML for schema in Figure 16

- When custom headers are added as shown in Figure 12 and **Parameters/PostData Details** is set to Binary Data**,** input schema is defined as shown in Figure 18 and a sample input is shown in Figure 19. Content of element **BinaryData** in input XML should be a **base64** encoded value of actual binary input.

<ns1:HTTPRequest>	
<URI>?	#string
<Header>?	
<Cookie>?	#string
<Content-Encoding>?	#string
<Content-Length>?	#string
<Content-Type>?	#string
<Pragma>?	#string
<CustomHeaders>?	
<Name2>	#string
<Name1>	#string
<Entity>?	
<BinaryData>	#string

Figure18: Schema when parameters/postdata set to 'BinaryData'

```
<ns1:HTTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/HTTPAdapter
   <URI>URI</URI>
   <Header>
      <Cookie>Cookie</Cookie>
      <Content-Encoding>Content-Encoding</Content-Encoding>
      <Content-Length>Content-Length</Content-Length>
      <Content-Type>Content-Type</Content-Type>
      <Pragma>Pragma</Pragma>
      <CustomHeaders>
         <Name2>Name2</Name2>
         <Name1>Name1</Name1>
      </CustomHeaders>
   </Header>
   <Entity>
      <BinaryData>VGhpcyBpcyBiaW5hcnkgZGF0YQ</BinaryData>
   </Entity>
</ns1:HTTPRequest>
```

Figure 19: Sample input XML for schema in Figure 18

- When custom headers are added as shown in Figure 12 and **Parameters/PostData Details** is set to XML, input schema is defined as shown in Figure 20 and a sample input is shown in Figure 21.

<ns1:HTTPRequest>	
<URI>?	#string
<Header>?	
<Cookie>?	#string
<Content-Encoding>?	#string
<Content-Length>?	#string
<Content-Type>?	#string
<Pragma>?	#string
<CustomHeaders>?	
<Name2>	#string
<Name1>	#string
<Entity>?	
<XMLData>	
<ns2:Title>	#string

Figure 20: Schema when parameters/postdata set to 'XML'

```
<ns1:HTTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/HTTPAdapter
   <URI>URI</URI>
   <Header>
      <Cookie>Cookie</Cookie>
      <Content-Encoding>Content-Encoding</Content-Encoding>
      <Content-Length>Content-Length</Content-Length>
      <Content-Type>Content-Type</Content-Type>
      <Pragma>Pragma</Pragma>
      <CustomHeaders>
         <Name2>Name2</Name2>
         <Name1>Name1</Name1>
      </CustomHeaders>
   </Header>
   <Entity>
      <XMLData>
         <ns2:Title xmlns:ns2="http://www.books.org">Title</ns2:Title>
      </XMLData>
   </Entity>
</ns1:HTTPRequest>
```

Figure 21: Sample input XML for schema in Figure 20

A sample schema is provided by clicking PostData button. This schema is set as child element under **XMLData** element. In the input message, xml data conforming to the schema specified should be set else an error can be thrown as this is a required field.

- When Content-Type is set as multipart/form-data and the custom headers are added then the input schema is defined as shown in Figure 22 and a sample input is shown in Figure 23. The Schema doesn't contain the Content-Type element in this case.

<ns1:HTTPRequest>	
<URI>?	#string
<Header>?	
<Cookie>?	#string
<Content-Encoding>?	#string
<Content-Length>?	#string
<Content-Type>?	#string
<Pragma>?	#string
<CustomHeaders>?	
<Name2>	#string
<Name1>	#string
<Entity>?	
<NVPairs>	
<ns1:NVPair>*	
@Name	#string
@Value	#string

Figure 23: Schema when parameters/postdata set to Parameters

```
<ns1:HTTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/HTTPAdapters1/HT
   <URI>URI</URI>
   <Header>
      <Cookie>Cookie</Cookie>
      <Content-Encoding>Content-Encoding</Content-Encoding>
      <Content-Length>Content-Length</Content-Length>
      <Content-Type>Content-Type</Content-Type>
      <Pragma>Pragma</Pragma>
      <CustomHeaders>
         <Name2>Name2</Name2>
         <Name1>Name1</Name1>
      </CustomHeaders>
   </Header>
   <Entity>
      <NVPairs>
         <ns1:NVPair Name="Name" Value="Value"/>
      </NVPairs>
   </Entity>
</ns1:HTTPRequest>
```

Figure 23: Sample input XML for schema in Figure 21

The files to be uploaded can be provided under Files element as shown in Figure 25. In the element File the Name attribute specifies the name of the parameter and the Value specifies the path of the file to be uploaded. If the **custom headers** are not configured, then the element CustomHeaders will not be present in the schema.

When custom headers are added as shown in Figure 13 and **Parameters/PostData Details** is set to parameters, input schema is defined as shown in Figure 24 and a sample input is shown in Figure 25.

Figure 24: Schema when parameters/postdata set to Parameters

```
<ns1:HTTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/HTTPAdapters1/HT
   <URI>URI</URI>
   <Header>
      <Cookie>Cookie</Cookie>
      <Content-Encoding>Content-Encoding</Content-Encoding>
      <Content-Length>Content-Length</Content-Length>
      <Content-Type>Content-Type</Content-Type>
      <Pragma>Pragma</Pragma>
      <CustomHeaders>
         <Name2>Name2</Name2>
         <Name1>Name1</Name1>
      </CustomHeaders>
   </Header>
   <Entity>
      <NVPairs>
         <ns1:NVPair Name="Name" Value="Value"/>
      </NVPairs>
   </Entity>
</ns1:HTTPRequest>
```

Figure 25: Sample input XML for schema in Figure 24

When custom headers are added, they are added as child elements under **CustomHeaders** element. The addition of parameter does not affect the input schema generated.

In all the above cases except when **PostData** is set to XML, all the elements present in the Input message are optional. A Sample input message with no fields can be used as shown in Figure 26.

```
<ns1:HTTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/HTTPAdapters3/HTTP/

</ns1:HTTPRequest>
```

Figure 26: Sample input XML

If there are no fields as shown in above Figure 24, the default values which are configured in CPS are set on the request message. Else the value of the corresponding field is taken from input message.

When the property Use Connection details from input is chosen, an additional element ConnectionFactorySettings is added to the input schema, as shown in the figure. Properties that are used to create the connection are present under this element.

XML Item	Value Type
<ns1:HTTPRequest>	
<ConnectionFactorySettings>?	
<ConnectionProperties>?	
<HTTPHost>?	#string
<HTTPPort>?	#int
<HTTPRequestTimeout>?	#int
<AllowAutoRedirection>?	true | false
<ForceHTTP10>?	true | false
<AuthenticationMode>?	None | Basic | Digest
<BasicOrRealmProperties>?	
<UserName>?	#string
<Password>?	#string
<RealmName>?	#string
<ByPassProxyForHosts>?	#string
<SSLSettings>?	
<TrustStoreLocation>?	#string
<TrustStorePassword>?	#string

Figure 27: Input schema with ConnectionFactorySettings

4.13.1.3.2 Output

The output schema for the component is based on the configuration details provided for **HTTPResponseProperty**.

When the **Response Body** for output message is set as **Headers** and no schema is provided for the Response, then no schema is set on output port. If a schema is provided by clicking the Response button, it is set as the output schema. If any headers are selected or if any custom headers are configured, then they are set as message properties on the output message.

When the **Response Body** for output message is set as XML, output schema generated depends on the schema provided by clicking the **Response** button.

- When no schema is provided in Response, output schema is defined as shown in Figure 28 and a sample output XML is shown in Figure 28.

```
<ns1:HTTP-Response>
    [0,1]
        <ns1:Response-Code>?            #string
        <ns1:Response-Message>?         #string
        <ns1:Header>
            [0,1]
                <ns1:Content-Encoding>  #string
                <ns1:Content-Length>    #string
                <ns1:Content-Type>      #string
                <ns1:Content-Range>     #string
                <ns1:Content-Location>  #string
                <ns1:Last-Modified>     #string
                <ns1:Set-Cookie>        #string
        <ns1:Body>
            <ns1:Text>*                 #string
```

Figure 28: Output Schema when no schema is set in Response

```
<ns1:HTTP-Response xmlns:ns1="http://www.fiorano.com/fesb/activity/HTTPAdapters1/
    <ns1:Response-Code>Response-Code</ns1:Response-Code>
    <ns1:Response-Message>Response-Message</ns1:Response-Message>
    <ns1:Header>
        <ns1:Content-Encoding>Content-Encoding</ns1:Content-Encoding>
        <ns1:Content-Length>Content-Length</ns1:Content-Length>
        <ns1:Content-Type>Content-Type</ns1:Content-Type>
        <ns1:Content-Range>Content-Range</ns1:Content-Range>
        <ns1:Content-Location>Content-Location</ns1:Content-Location>
        <ns1:Last-Modified>Last-Modified</ns1:Last-Modified>
        <ns1:Set-Cookie>Set-Cookie</ns1:Set-Cookie>
    </ns1:Header>
    <ns1:Body>
        <ns1:Text>Text</ns1:Text>
    </ns1:Body>
</ns1:HTTP-Response>
```

Figure 29: Sample XML for schema in Figure 28

If any custom headers are configured, then they are added as child elements under **CustomHeaders** element which is set as child element under **Header** element.

- When schema is provided in Response, output schema is defined as shown in Figure 30 and a sample output XML is shown in Figure 31.

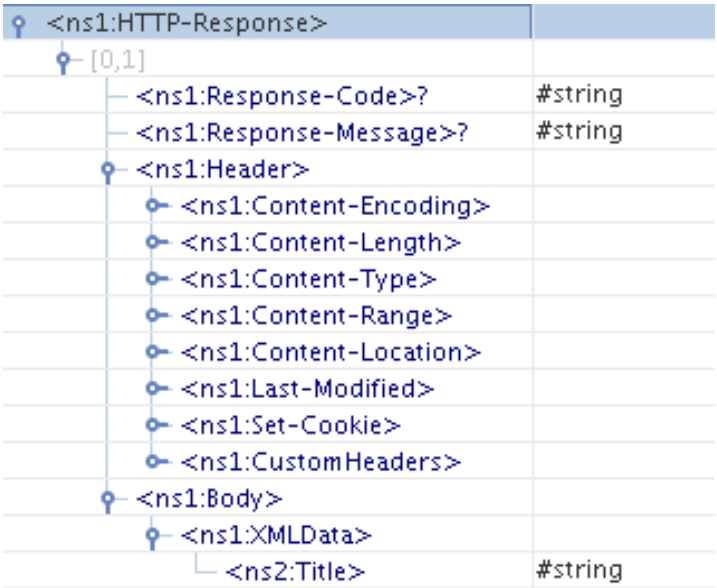


Figure 30: Output Schema when sample schema is set in Response

```
<ns1:HTTP-Response xmlns:ns1="http://www.fiorano.com/fesb/activity/HTTPAdapters
   <ns1:Response-Code>Response-Code</ns1:Response-Code>
   <ns1:Response-Message>Response-Message</ns1:Response-Message>
   <ns1:Header>
      <ns1:Content-Encoding/>
      <ns1:Content-Length/>
      <ns1:Content-Type/>
      <ns1:Content-Range/>
      <ns1:Content-Location/>
      <ns1:Last-Modified/>
      <ns1:Set-Cookie/>
      <ns1:CustomHeaders>
         <ns1:Name1/>
      </ns1:CustomHeaders>
   </ns1:Header>
   <ns1:Body>
      <ns1:XMLData>
         <ns2:Title xmlns:ns2="http://www.books.org">Title</ns2:Title>
      </ns1:XMLData>
   </ns1:Body>
</ns1:HTTP-Response>
```

Figure 31: Sample XML for schema in Figure 30

4.13.1.4 SSL Setup

To communicate with https websites, SSL has to be enabled in HTTPAdapter and the values for KeyStore, TrustStore, KeyStore Password, TrustStore password and KeyStore Client Key have to be specified in the SSL settings dialog in the CPS.

Client keystore and truststore are required for SSL communication. The procedure to obtain the certificate from the https website and to generate the client keystore and truststore is explained in sections 3.11.2.1, 3.11.2.2 and 3.11.2.3 of Fiorano SOA User Guide.

Section 3.11.2.4 (Using the Keystore and Truststore in an SSL Application) explains the procedure to use the generated Client keystore and truststore in HTTPAdapters for SSL configuration.

The default HTTP Port to be used for SSL communication is 443. If the http server uses a different port then it has to be mentioned.

4.13.1.5 Using Named Configurations

Changes are done in eStudio wrt to Named Configurations. With the implementation of Named Configurations MCF panel is changed as shown in Figure below.

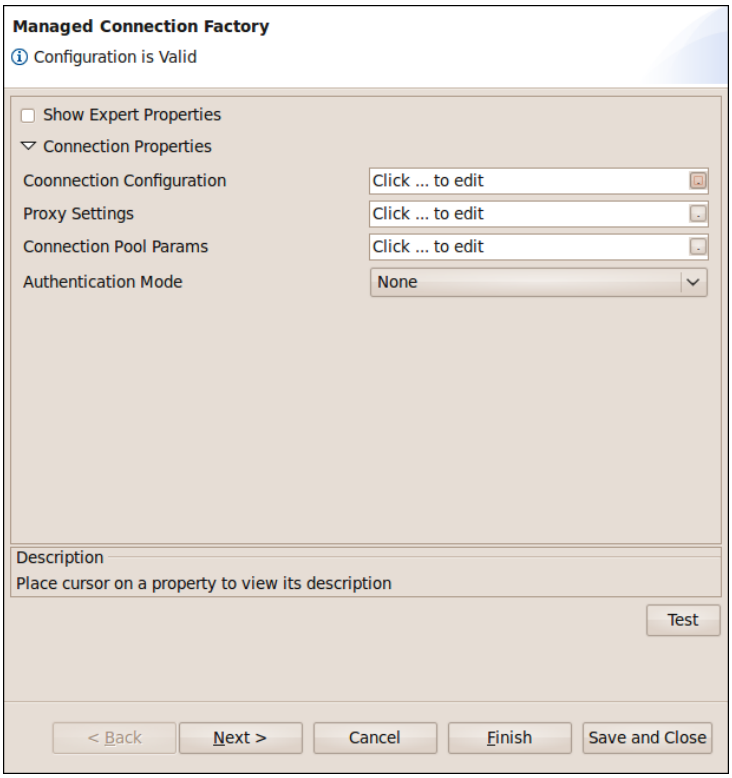


Figure: Modified Managed Connection Factory

Some properties are grouped as one property in the modified configuration panel. HTTP Host Name/IP, HTTP Port, HTTP Request Timeout, Allow Auto Redirection and Force HTTP which were present previously in the MCF panel are grouped under the name Connection Configuration.

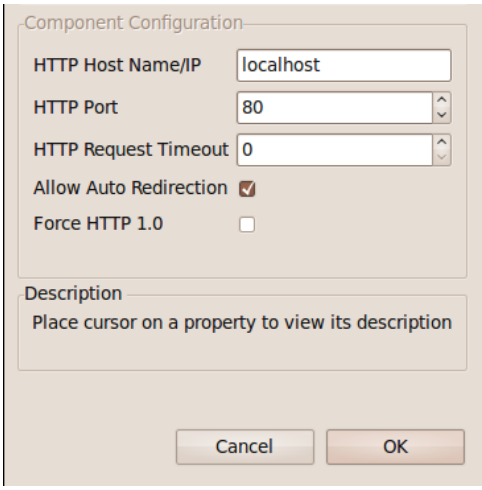


Figure: Connection Configuration

Similarly SSL Security details and Proxy Settings Panels are modified to use Named Configurations.

4.13.1.6 Functional Demonstration

4.13.1.6.1 Scenario 1

Send a request to Web Server and display the response.

Configure the HTTPAdapters as described in *Configuration and Testing* section and use feeder and display component to send sample input and check the response respectively.

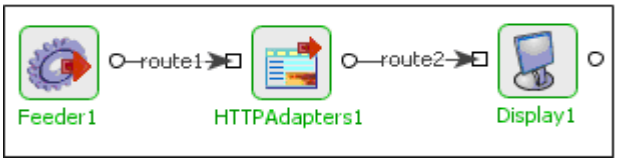


Figure 32: Demonstrating Scenario 1 with sample input and output

Input Message

```
<ns1:HTTPRequest xmlns:ns1="http://www.fiorano. /fesb/activity/HTTPAdapters1/HTTP/In">
 <URI>www.google.com</URI>
 <Header>
<Cookie>Cookie</Cookie>
<Pragma>Pragma</Pragma>
 </Header>
 <Entity>
<NVPairs>
 <ns1:NVPair Name="Name" Value="Value"/>
</NVPairs>
 </Entity>
</ns1:HTTPRequest>
```

Output Message

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD>
<META http-equiv=Content-Type content="text/html; charset=iso-8859-1">
<META content="MSHTML 6.00.2600.0" name=GENERATOR>
<script language="JavaScript" type="text/JavaScript">
<!--

function MM_preloadImages() { //v3.0
var d=document; if(d.images){ if(!d.MM_p) d.MM_p=new Array();
var i,j=d.MM_p.length,a=MM_preloadImages.arguments; for(i=0; i<a.length; i++)
if (a[i].indexOf("#")!=0){ d.MM_p[j]=new Image; d.MM_p[j++].src=a[i];}}
}
//-->
</script>
<link href="css/fiorano.css" rel="stylesheet" type="text/css">
</HEAD>
<BODY bgcolor="#FFFFFF" leftmargin="0" topmargin="0" marginwidth="0" marginheight="0">

<Script Language="JavaScript" >
<!--
function search_check()
{
                        if(document.search.query.value == "")
                        {
                            alert("Query cannot be blank")
                        }

                        else
                        {
                            document.search.submit()
                        }
}
-->
</Script>
<html>
<head>
<link rel="stylesheet" href="/css/fiorano.css" type="text/css">
……................
</body>
</html>
```

4.13.1.7 Use Case Scenario

In Order Entry sample, a user is provided a web based interface to send a purchase order to a company. In case the order is accepted, HTTPAdapters is used to POST the order delivery request to a third party vendor.

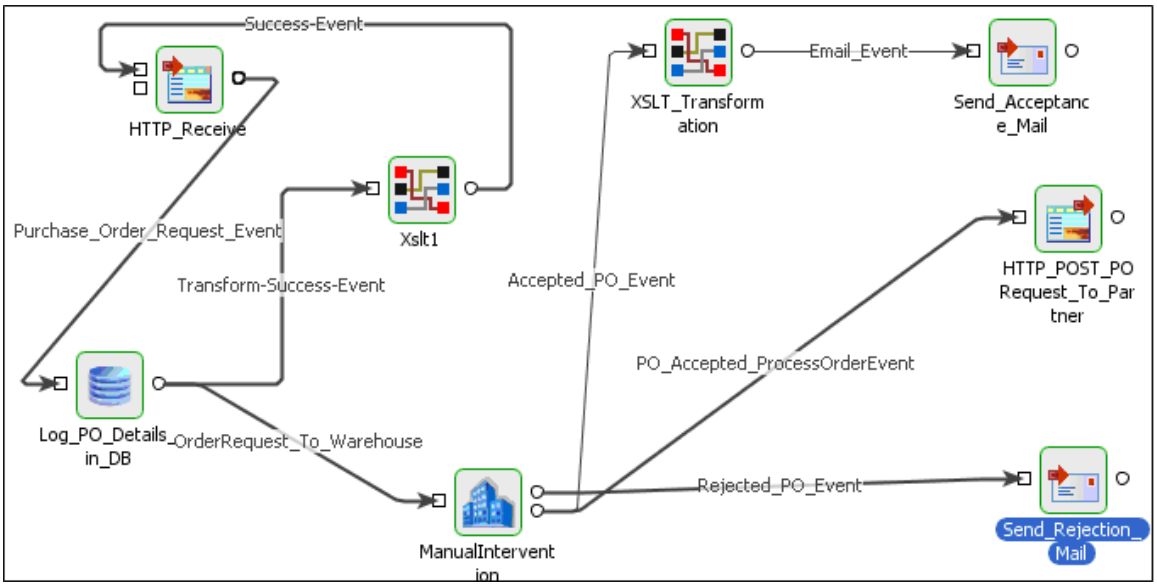


Figure 33: Order Entry

4.13.1.8 Useful Tips

When connecting to secured sites, the component accesses truststore from different locations in different launch modes -

- Separate Process - `%JAVA_HOME%\jre\lib\security\cacerts`
- In-memory - `%FIORANO_HOME%\esb\server\profiles\certs\jssecacerts`

'jssecacerts' in %FIORANO_HOME% does not have trusted certificates required by most of the secured sites. Hence, to connect to secured sites when component is launched in-memory, either the trusted certificates should be added to 'jssecacerts' or the property 'javax.net.ssl.trustStore' (in %FIORANO_HOME%\esb\server\bin\server.conf, if FPS is launched using server.bat/server.sh and %FIORANO_HOME%\esb\fps\bin\fps.conf, if FPS is launched using fps.bat/fps.sh) should be set to %JAVA_HOME%\jre\lib\security\cacerts

4.13.2 HTTPReceive

The HTTPReceive component acts as an interface between an HTTP client and an Event Process and receives HTTP requests using the Hyper Text Transfer Protocol (HTTP). The request is then converted into an XML Message by the HTTPReceive component and is sent to the Event Process which in turn processes it and sends the result to the HTTPReceive component. The HTTPReceive component processes the input in either of the following ways.

The Event Process processes the input and sends the output to the HTTPReceive component. In this case, the input port of the HTTPReceive component receives the output, converts it into HTTP response and sends it back to the HTTP client through HTTP protocol.

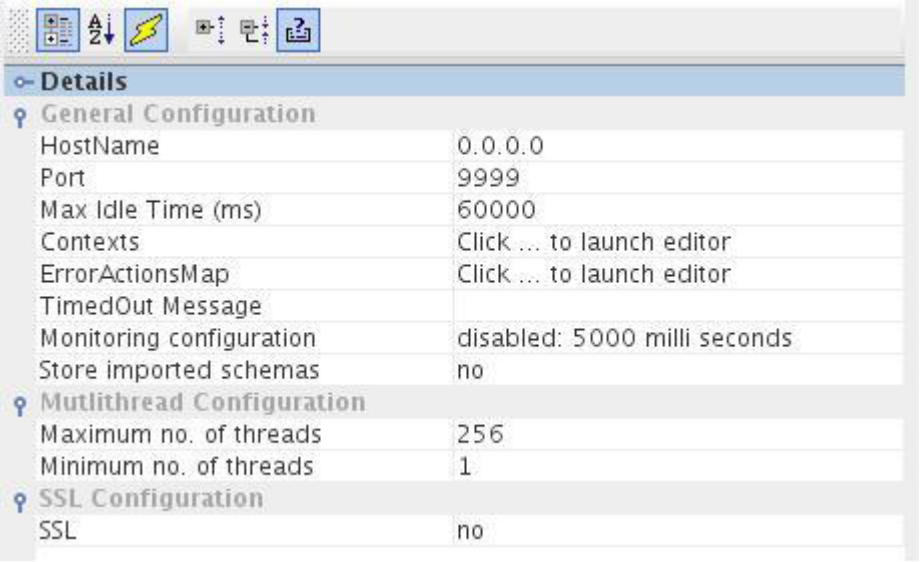
The Event Process sends a response error message to the HTTPReceive component. In this case, the error event port of the HTTPReceive component receives the response error through its input event port, converts it into a HTTP response error and then sends it to the HTTP client using HTTP.

The Event Process need not return response. In this case, the HTTPReceive component is preconfigured to the
one-way-send mode wherein the HTTPReceive component does not expect and wait for any response. By default, HTTP(S) default response or OK would be returned to the HTTP client for each HTTP request.

Note: The Stream Read Buffer size field should be set only by advanced users as the optimum value depends on specific scenarios in which the component is used.

4.13.2.1 Configuration and Testing

The configuration of HTTPReceive is defined as shown in Figure 1.



Details	
General Configuration	
HostName	0.0.0.0
Port	9999
Max Idle Time (ms)	60000
Contexts	Click ... to launch editor
ErrorActionsMap	Click ... to launch editor
TimedOut Message	
Monitoring configuration	disabled: 5000 milli seconds
Store imported schemas	no
Mutlithread Configuration	
Maximum no. of threads	256
Minimum no. of threads	1
SSL Configuration	
SSL	no

Figure 1: Configuration of HTTPReceive

4.13.2.1.1 General Configuration

4.13.2.1.1.1 HostName

The host name or the IP address of the system on which the web server is present.

4.13.2.1.1.2 Port

The port on which the Web Server is running.

4.13.2.1.1.3 Max Idle Time (ms)

The maximum time in milliseconds for which the component has to wait for processing the request.

For more details, refer http://jetty.codehaus.org/jetty/jetty-6/apidocs/org/mortbay/jetty/AbstractConnector.html#setMaxIdleTime(int)

4.13.2.1.1.4 Contexts

A Context will encapsulate details of a single HTTP service. Multiple contexts can be configured as shown in Figure 2.

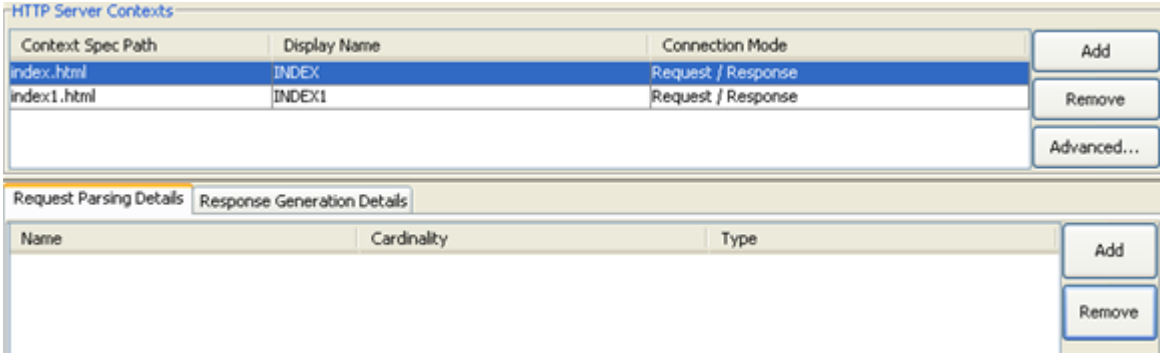


Figure 2: Configuration of Multiple Contexts

A Context can be added by clicking on the Add button. For each context added, ports will be added to the component based on the value given in the editable table as explained below.

Context Specific path: The unique context path of the HTTP request corresponding to a context. The path of the context will be computed relative to the server details provided by properties Host Name and Port.

The URI of a context will be http://<Host Name>:<Port>/<Context Spec Path>.

Display Name: The unique display name for the context. The names of the ports that are generated corresponding to a context are suffixed by **__<Display Name>**.

Connection Mode: The connection mode defines the way the client interacts with the server. The client sends the request to the server. The **HTTPReceive** adapter parses the request and sends it to the Event Process through output port **HTTPRequest__<Display Name>** which will be sent to the Event Process. The behavior of the component after this will be dependant on the connection mode chosen.

- **Request/Response**

  In this mode, the **HTTPReceive** adapter waits for reply from the Event Process after sending the request. Two input ports with names **HTTPResponse__<Display Name>** and **ERROR__<Display name>** will be created for receiving response and error messages respectively. Then the component handles the response based on the response generation details provided for the context.

- **One Way Send**

  In this mode, the **HTTPReceive** adapter will send a default response, HTTP(S) default response/OK back to the HTTP client after sending the request message to the Event Process. No input ports will be created in this case for this context.

  When a context is selected the details of handling the request and response can be configured using the tabs Request Parsing Details and Response Generation Details.

Request Parsing Details: The HTTP Request stream received by the component when a client sends request will be parsed based on the details provided in this tab. The UI has three tabs at the bottom which define various types of request parsing details as shown in Figure 3.

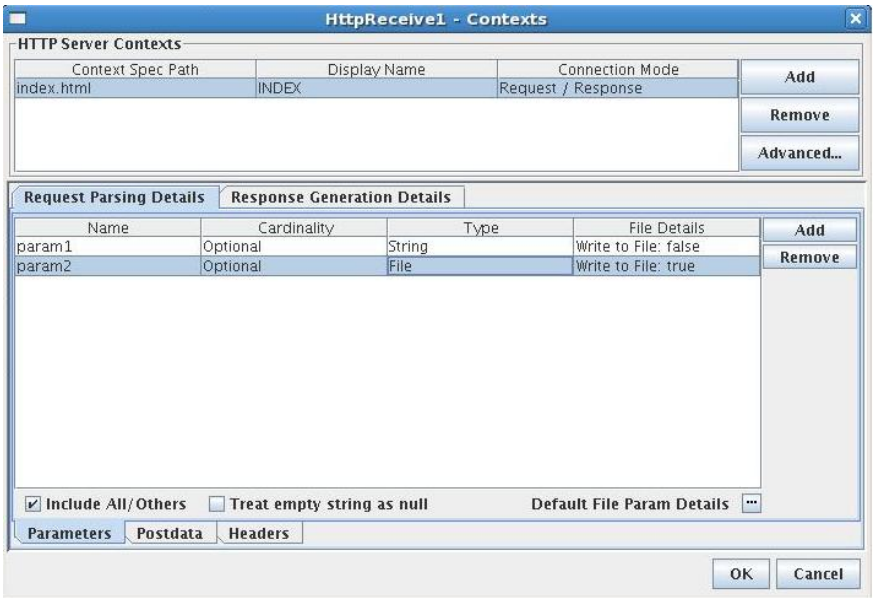


Figure 3: Configuration of Request Parsing Details

When a HTTP request is received by the component, it is transformed into a JMS message and sent to the Event Process through the output port of the component based on the details configured using this property. Schema of the output port **HTTPRequest_<Context Display Name>** will change based on the details provided for this property.

- **Parameters**

  Parameters define the characteristics of the HTTP request data stream being parsed for converting the request to message. Parameters can be added by clicking Add button in the Parameters tab in request parsing details tab. Added parameters can be removed by clicking on the Remove button. For each parameter added a new element will be added as child of Params element in the schema of the corresponding output port based on the following details.

- **Param Details**
  - **Name**

    The name of the parameter that is passed in the request. The name for corresponding element in the output schema will be set to this value.
  - **Cardinality**

    If a parameter is definitely necessary for the processing of a request then it must be marked required, otherwise, optional must be chosen. The cardinality of the corresponding element in the output schema will be the same.
  - **Type**

    The data type of he parameter can be specified as one of String, Boolean, Decimal or Integer. The XSD type of corresponding element in the output schema will be same.
  - **File Details**

    If the param is of File data type which especially used in multipart/form-data request, details regarding whether the file attachment should be saved or not are to be entered in File Details window of the param as shown in Figure 4.

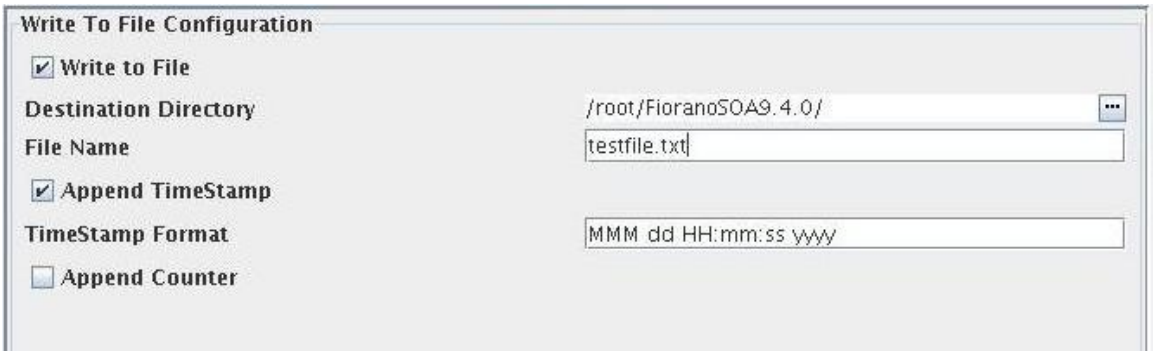


Figure: Write to File Configuration Details of Param

- **Write to File**: Select this option to write request attachments to file, if not selected attachments will be sent to output port. While sending to output port, Attachments will be base64 encoded if they does not contain text content.
- **Destination Directory**: Provide the path of the Directory in which attachments needs to be saved.
- **File Name**: Specify the file name in which corresponding file param attachment needs to be saved.
- **Append TimeStamp**: Select this option to append time stamp to the file name while writing attachments to file.
- **TimeStamp Format**: Specify the time stamp format to be appended to the file name.
- **Append Counter**: Select this option to append a counter to the file name in case a file with the same name exists. If Is Append TimeStamp is also enabled then time stamp will be appended before appending counter.

  Example: fileName_0(with out time stamp), fileName_1313406444699_1(with time stamp).

- **Include All/Others**

  If this option is selected, parameters from the request that are not defined as parsing parameters will also be added in output message. These will be added under the Params element in the output schema.

- **Treat empty string as null**

  If this option is selected, and the content of this parameter in HTTP Request stream is empty, the parameter will be considered as null otherwise it will be considered as an empty string.

- **Default File Param Details**

  Enter the default file param details for saving attachments to file. If no details are mentioned in File Details window of newly added param, the default values given here will be picked to save the attachments or to send the encoded attachment data to output port.

- **Post Data**

  If data is relatively large and is to be posted from the request, the way it has to be parsed must be specified by selecting the **Post Data** tab in the request parsing details tab.

  The data to be posted by the client can be one of the following types.

  - **----------**

    This option must be specified when data is not posted as part of the request. If chosen, post data, if passed as part of the HTTP request will not be present in the request message. In this case parameters or headers must be compulsorily included.

  - **XML Text [Provide an XSD for the XML text]**

    This must be chosen if the data posted is XML that conforms to a schema. The schema can be provided using the XSD editor as shown in Figure 4. If parameters or headers are specified then an element XMLData will appear in the schema of output port whose type is same as that of the provided schema. Otherwise, this schema will be set as output port schema.

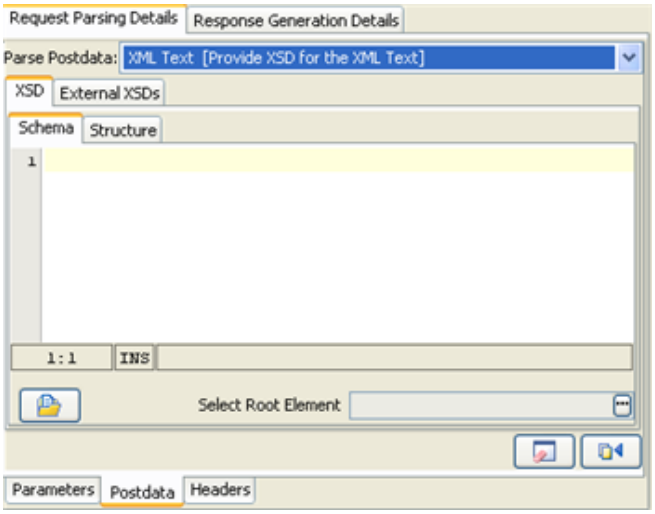


Figure 4: Configuration of Post Data in Request Parsing Details

- **Simple Text**

  If data from the request stream is text not conforming to any schema. This option can be selected if the data need not be transformed using Fiorano Mapper and needs to be transferred as is. If parameters or headers are specified, an element Data of string type will appear in the schema of output port and the text will be inserted as CDATA. Otherwise, it will be set as message text.

- **Bytes**

  Data from the request stream will be filled as bytes in the JMS Message. Use this option in case you need to send media files as binary data via HTTP.

- **Headers**

  A Header is a part of a data stream which specifies information about the data stream. This option is used to specify the type of information being sent across the data stream. Headers can be added by clicking Add button in the Headers tab in request parsing details tab. Added Headers can be removed by clicking on the Remove button. For each Header added a new element will be added as child of Headers element with attributes name, cardinality and type similar to Parameters.

  - **Fill Defaults**

    This button can be used to include the default headers as shown in the Table1.

Scheme	Optional	String
Version	Optional	String
IsCommitted	Optional	String
Host	Optional	String
Port	Optional	String
EncodedPath	Optional	String
Method	Optional	String
Connection	Optional	String
Pragma	Optional	String
Content-Length	Optional	String
Content-Type	Optional	String
Cookie	Optional	String

Table 1: Default headers and their data types

  - **Include All/Others**

    If this option is selected all headers from the request stream are included in the message from the output port. The headers will be added under Headers element.

Response Generation details: If the context connection mode is chosen as Request/Reply, the component sends the request message and waits for response from the Event Process. The response message that is received on input port **HTTPResponse__<Context Display Name>** is converted to HTTP Response stream suitable for the invoking client based on the details provided here.

The selection can be made in the Response Generation Details tab and selecting the Response Data tab at the bottom, as shown in the Figure 5.

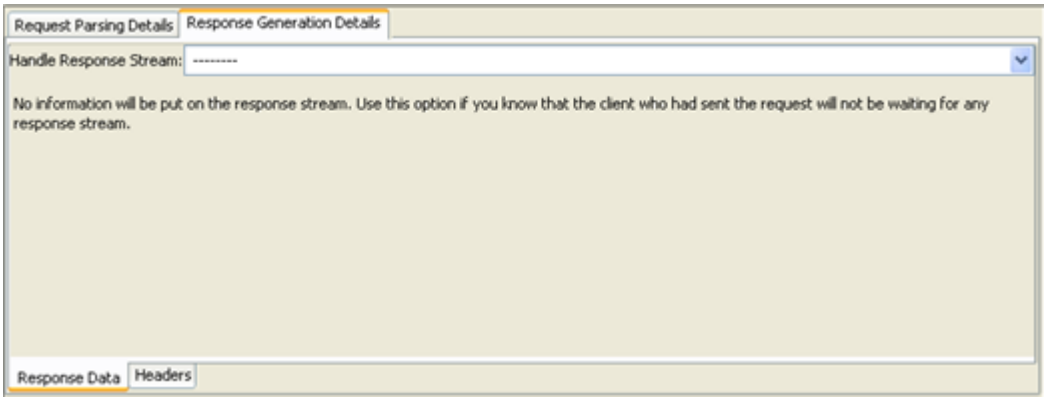


Figure 5: Response Generation Details – Response Data

- **Response Data**
  - **-------**

    When this option is used, no information is put on the response stream.
  - **Simple Text**

    Text portion of the JMS Message will be read and put in the response stream. This option can be chosen when the invoking client does not expect a response that conforms to a specific schema. No schema is set on the input port **RESPONSE**.
  - **XML Text [Provide an XSD for the XML Text]**

    If the client expects the response to be compliant to a particular schema, then the schema must be provided using the schema editor. The schema will be set as schema of the input port **RESPONSE**.
  - **Bytes**

    If the client expects the response to be compliant to a particular schema, then the schema must be provided using the schema editor. The schema will be set as schema of the input port **RESPONSE**.
- **Headers**

  This option is used to specify the type of information being sent across the response stream. Headers can be added by clicking Add button in the Headers tab in Request Parsing Details tab. Added Headers can be removed by clicking on the Remove button. For each header added, a new element will be added as child of Headers element in the schema of the input port with attributes name, cardinality and type similar to **Parameters** in Request Parsing Details.

- **Fill Defaults**

  This is used to include all default headers in the response message. Please refer to Table 1 for default headers.

- **Handle All**

  If this option is selected all headers that are present in the response received from the Event Process will be added as headers in the response stream.

Advanced Properties:

The advanced properties for a context can be configured by clicking on the **Advanced Properties** button in **Context Details** panel. These properties depend on the connection mode between the adapter and the client.

- If the **Connection Mode** of the adapter is Request/Response, then clicking this button pops up the following details as shown in Figure 6.

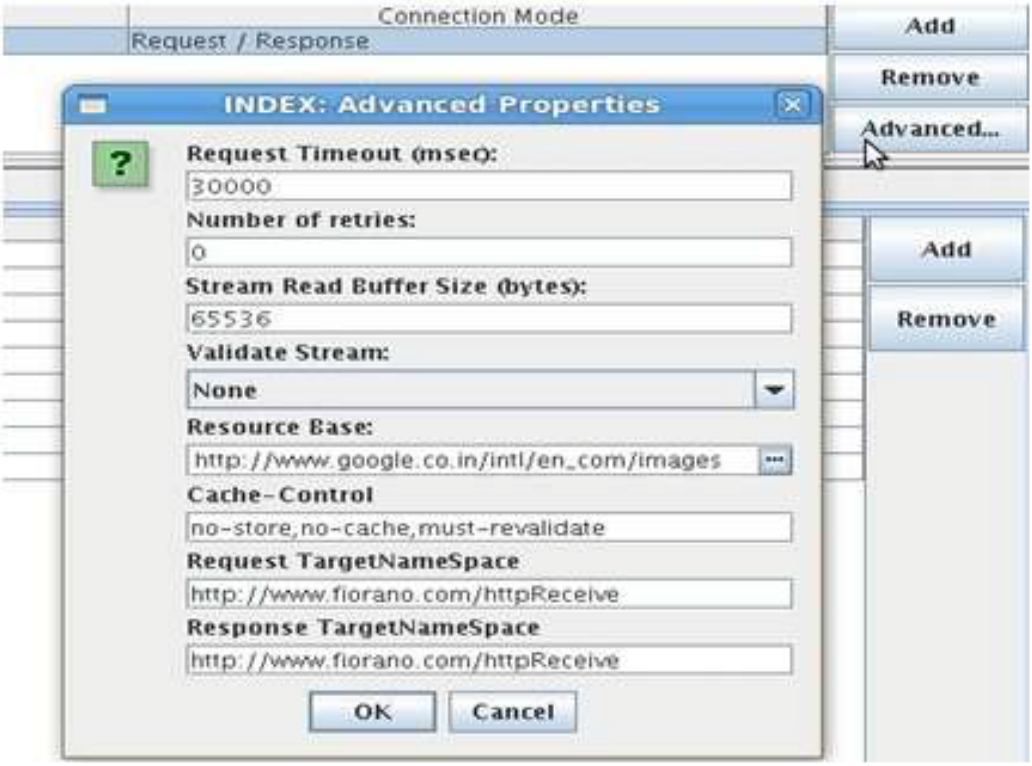


Figure 6: Advanced Properties - Request/Response

- **Request TimeOut (msec):** It is the length of time in milliseconds which the HTTP Receive adapter will wait for the response message.
- **Number of Retries:** It is the number of times adapter will retry in case there is no response.
- **Stream Read Buffer Size (Bytes):** It is the size of the buffer that will be used to read a HTTP request stream.
- **Validate Stream:** Validates the request and response stream based on the schema generated. It has the following options:
  - Both Request and Response Stream: validates both the request and response streams.
  - Only Request Stream: Validates only request stream
  - Only Response Stream: Validates only response stream
  - None: No validation is done for request and response streams

- **Resource Base:** Location of the resource that contains the static content like images, css and so on. This can be any valid path in the file system (may also be relative to %FIORANO_HOME%\runtimedata\PeerServers\<profile-name>\FPS\run\components\HttpReceive\4.0) or a web URL. The resources present in resource base will then be hosted in URLs of the form http://<hostname>:<port>/<contextpath>/<resource>
- **Cache Control:** The cache control mechanism that is to be used to cache the static content. For example, cache control value "max-age=3600,public" states that static content would be cached for up to an hour and shared between all users, without checking the server. For valid Cache-Control values, please refer to section 14.9 in the following link http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html
- **Request TargetNameSpace:** The target name space for the schema generated for the output port **HTTPRequest_<Context Display Name>**.
- **Response TargetNameSpace:** The target name space for the schema generated for the input port **HTTPResponse_<Context Display Name>**.

- If the **Connection Mode** is One Way Send, the following properties as shown in Figure 7 can be configured.

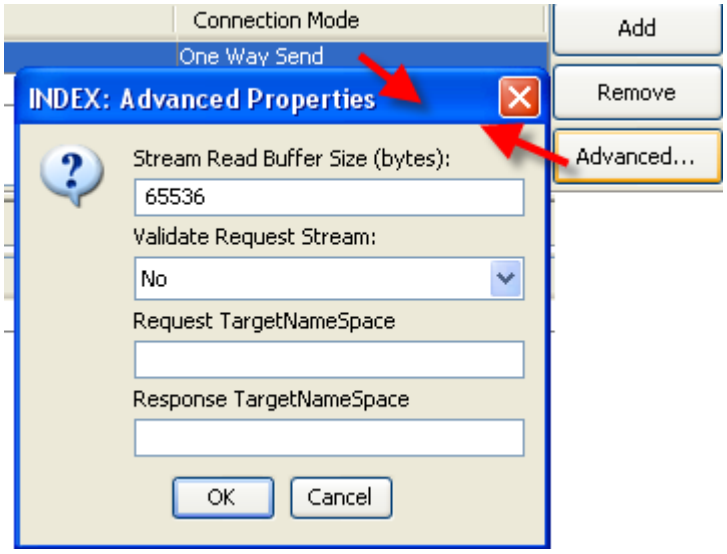


Figure 7: Advanced Properties - One Way Send

For description of these properties, refer to the section of **Advanced** properties for Request/Response connection mode.

4.13.2.1.1.5 Error Actions Map

The suitable actions that have to be taken for possible errors that might happen while the component is running can be specified using the Error Actions Map editor, by clicking on the ellipsis button against this property.

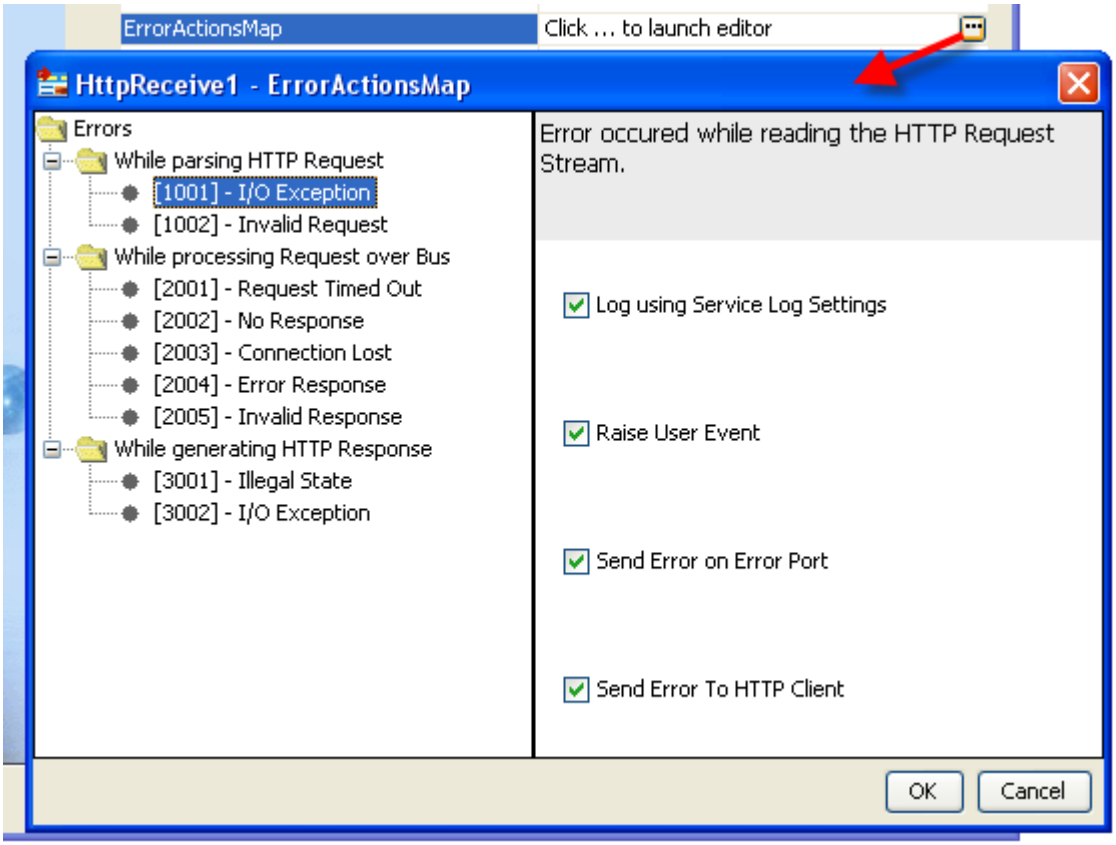


Figure 8: Configuration of Error Actions Map

The various options for handling errors that might occur while sending or receiving data streams can be configured. The various instances when an error might occur are as follows:

- While parsing HTTP Request

  These errors may occur while parsing the HTTP request.

  - [1001] - I/O Exception: This error may occur while reading the HTTP Request Stream
  - [1002] - Invalid Request: This error may occur when HTTP Request does not match with the configured Format.

- While processing Request over Bus

  These errors may occur while processing the request over the bus.

  - [2001] - Request Timed Out: This error may occur when the request is timed out.
  - [2002] - No Response: This error may occur when there is no response.
  - [2003] - Connection Lost: This error may occur when the Connection to Peer Server is lost while sending Request Document over the Bus.
  - [2004] - Error Response: This error may occur when the Response received from the Bus contains one or more errors.
  - [2005] - Invalid Response: This error may occur when the Response does not match with the configured Format.
- While Generating HTTP Response

  These errors may occur when the HTTPReceive adapter is generating the HTTP Response.

  - [3001] - Illegal State: This error may occur while setting HTTP Response Headers.
  - [3002] - I/O exception: This error may occur while sending data over the HTTP Response Stream.

Various remedial actions can be specified for each of the above mentioned errors to handle the errors effectively. The remedial actions that are displayed in the Error Handling panel are as follows:

- Log using service log settings

  The error is logged using the settings for the business service. These settings can be changed by using the properties view of the component.
- Raise User Event

  The error is logged as a user event.
- Send Error on error port

  The error is sent to the error port of the component.
- Send Error to HTTP client

  The error is sent to the HTTP client invoking the service.

4.13.2.1.1.6 Timed out Message

This is an expert property which will be displayed by clicking the button show expert properties in the CPS. The message that has to be sent to the client invoking a context when the request sent gets timed out.

4.13.2.1.2 Multithread Configuration

4.13.2.1.2.1 Maximum No of Threads

A thread pool ensures threads are used efficiently within the container. A number of threads are created at initialization and placed in the pool. When there is work to be done, for example, to service a request, a free thread from the pool is allocated and then returned to the pool when the work has been completed.

If the maximum pool size is reached, jobs wait for a free thread.

4.13.2.1.2.2 Minimum No of Threads

The minimum number of threads that have to be present in the thread pool. Idle threads in the pool will timeout and terminate until the minimum number of threads are running.

4.13.2.1.3 Using Named Configurations

Changes are done in eStudio wrt to Named Configurations. With the implementation of Named Configurations, Component Configuration panel is changed as shown in Figure below.

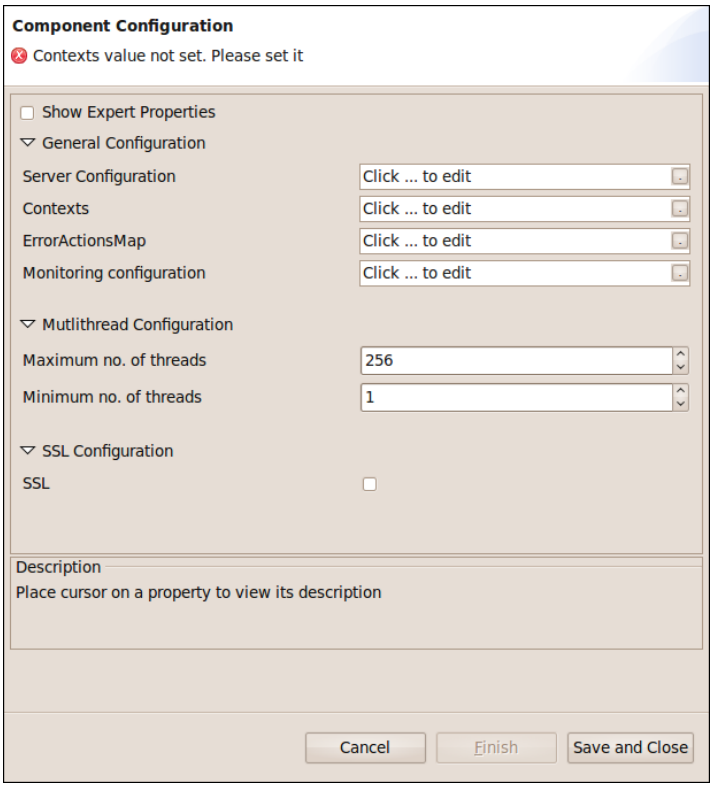


Figure: Modified Component Configuration Panel

Some properties are grouped as one property in the modified configuration panel. HostName, Port and Max Idle Time (ms) which were present previously in the Component Configuration panel are grouped under the name Server Configuration.

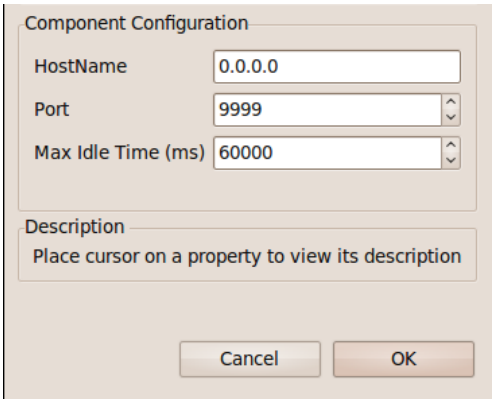


Figure: Connection Configuration

Similarly SSL Security details and Proxy Settings Panels are modified to use Named Configurations.

4.13.2.2 Input and Output

4.13.2.2.1 Input

The input schema is auto generated based on the configuration of **Response Generation Details** provided for the context as described under Response Generation Details.

- When Handle Response Stream in **Response Data** is set to option ------ or Bytes, and no headers are configured, then no schema is set on the input port **RESPONSE**.

  If headers are configured as shown in Figure 9, input schema generated is shown in Figure 10 and sample input XML is shown in Figure 11.

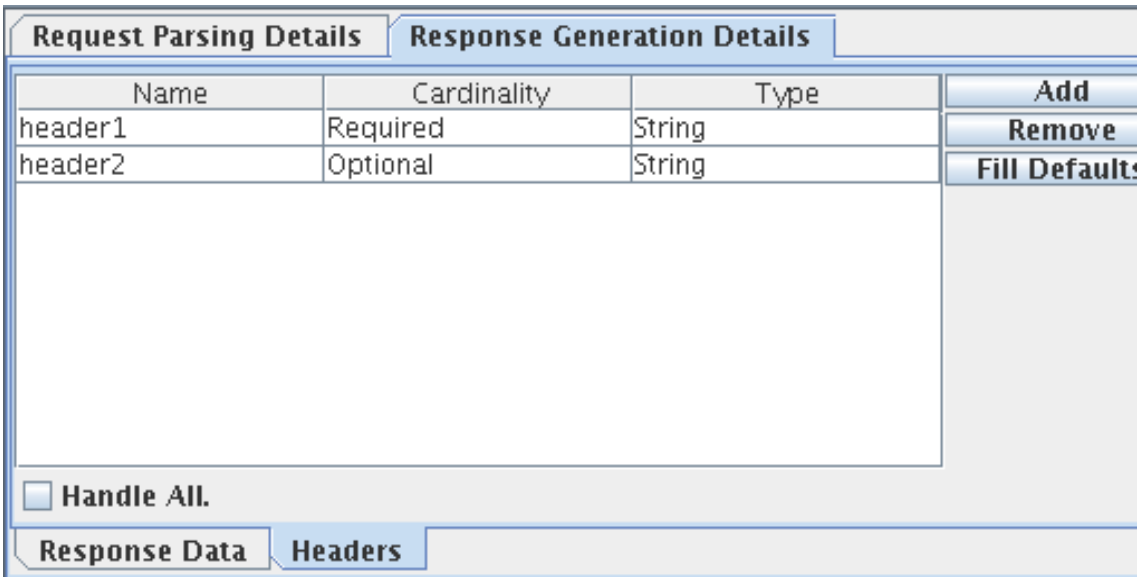


Figure 9: Sample Configuration of Response Generation Details – Headers

<ns1:HTTPResponse>	
<Headers>?	
<header1>	#string
<header2>?	#string

Figure 10: Sample Input Schema when headers are configured

```
<ns1:HTTPResponse xmlns:ns1="http://www.fiorano.com/httpReceive">
   <Headers>
      <header1>header1</header1>
      <header2>header2</header2>
   </Headers>
</ns1:HTTPResponse>
```

Figure 11: Sample Input XML for schema shown in Figure 10

<ns1:HTTPResponse>	
<Headers>?	
<header1>	#string
<header2>?	#string
<XMLData>	
<ns1:Title>	#string

Figure 12: Input port Schema when Response data is XML

- When Handle Response Stream in **Response Data** is set to XML, and headers are configured as shown in Figure 9, input schema generated is shown in Figure 12 and sample input XML in Figure 13.

```
<ns1:HTTPResponse xmlns:ns1="http://www.books.org">
   <Headers>
      <header1>header1</header1>
      <header2>header2</header2>
   </Headers>
   <XMLData>
      <ns1:Title>Title</ns1:Title>
   </XMLData>
</ns1:HTTPResponse>
```

Figure 13: Sample Input XML for schema shown in Figure 12

The schema provided in Response data is set under XMLData element in input port schema. If headers are not configured, then this schema is directly set on the input port.

- When Handle Response Stream in **Response Data** is set to Simple Text, and headers are configured as shown in Figure 9, input schema generated is shown in Figure 14 and sample input XML in Figure 15.

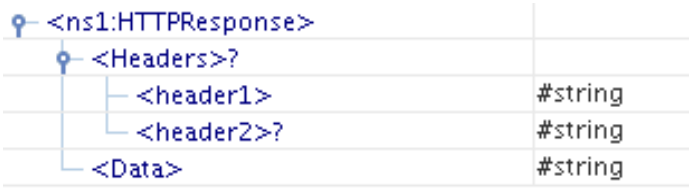


Figure 14: Input port Schema when Response data is Simple Text

```
<ns1:HTTPResponse xmlns:ns1="http://www.books.org">
   <Headers>
      <header1>header1</header1>
      <header2>header2</header2>
   </Headers>
   <Data>Data</Data>
</ns1:HTTPResponse>
```

Figure 15: Sample Input XML for schema shown in Figure 14

4.13.2.2.2 Output

The output schema is auto generated based on the configuration of request parsing details provided. If parameters are also provided, then the schema is a concatenation of the parameters and the schema provided.

- When Post Data in **Request Parsing details** is set to a option ------ or Bytes.

  Configure headers shown in Figure 9 in Request details and parameters as shown in Figure 16.

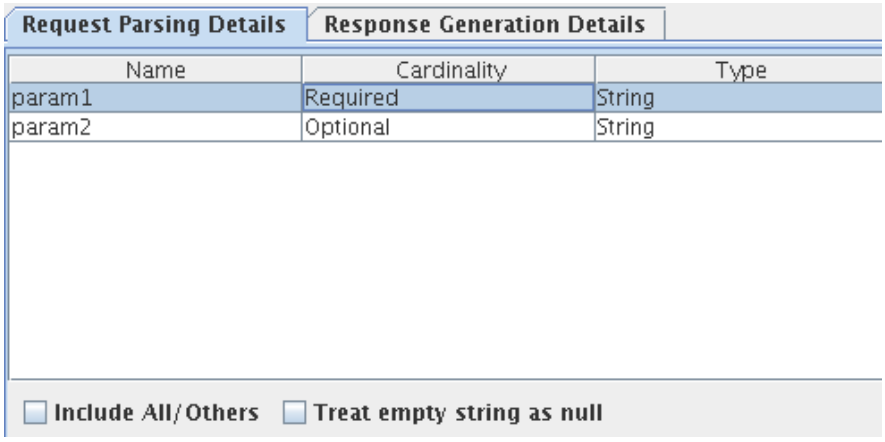


Figure 16: Sample Configuration of Request details – Parameters

For the above configuration, schema set on output port RESPONSE is shown in Figure 17 and sample XML in Figure 18.

<ns1:HTTPRequest>	
<Headers>?	
<header1>	#string
<header2>?	#string
<Params>?	
<param1>	#string
<param2>?	#string

Figure 17: Output Schema with parameters and headers.

```
<ns1:HTTPRequest xmlns:ns1="http://www.fiorano.com/httpReceive">
   <Headers>
      <header1>header1</header1>
      <header2>header2</header2>
   </Headers>
   <Params>
      <param1>param1</param1>
      <param2>param2</param2>
   </Params>
</ns1:HTTPRequest>
```

Figure 18: Output XML for schema shown in Figure 17

When Post Data is set to Bytes and if no headers and parameters are configured then no schema is set on the output port. When Post Data is set to -----, configuration of parameters is mandatory, at least one parameter should be configured.

- When Post Data in **Request Parsing details** is set to a option either XML, output schema is defined as shown in Figure 19 and sample XML shown in Figure 20.

<ns1:HTTPRequest>	
<Headers>?	
<header1>	#string
<header2>?	#string
<Params>?	
<param1>	#string
<param2>?	#string
<XMLData>	
<ns1:Title>	#string

Figure 19: Output schema when Post Data set to 'XML'

```
<ns1:HTTPRequest xmlns:ns1="http://www.books.org">
   <Headers>
      <header1>header1</header1>
      <header2>header2</header2>
   </Headers>
   <Params>
      <param1>param1</param1>
      <param2>param2</param2>
   </Params>
   <XMLData>
      <ns1:Title>Title</ns1:Title>
   </XMLData>
</ns1:HTTPRequest>
```

Figure 20: Output XML for schema shown in Figure 19

Schema which is provided in Post Data is set under XMLData of output schema. If no headers and parameters are provided, the schema provided is directly set on the output port.

- When Post Data in **Request Parsing details** is set to a option Simple Text, parameters defined as shown in Figure 16 and headers configured as shown in Figure 9, then output schema is defined as shown in Figure 21 and sample XML shown in Figure 22.

<ns1:HTTPRequest>	
<Headers>?	
<header1>	#string
<header2>?	#string
<Params>?	
<param1>	#string
<param2>?	#string
<Data>	#string

Figure 21: Output schema when Post Data set to 'Simple Text'

```
<ns1:HTTPRequest xmlns:ns1="http://www.books.org">
   <Headers>
      <header1>header1</header1>
      <header2>header2</header2>
   </Headers>
   <Params>
      <param1>param1</param1>
      <param2>param2</param2>
   </Params>
   <Data>Data</Data>
</ns1:HTTPRequest>
```

Figure 22: Output XML when Post Data set to Simple Text.

When **Post Data** is set to **Simple Text** and if no headers and parameters are configured, then no schema is set on the output port.

4.13.2.4 Functional Demonstration

4.13.2.4.1 Scenario 1

Receiving a HTTP request and returning the parsed request as response.

Configure the HTTP Receive adapter as described in *Configuration* section. Add two parameters REQUEST and CREDENTIALS in **Request Parsing Details**.

Connect a Display component to its output port and also the output port of Display to the response port of HTTP Receive as shown in Figure 23.

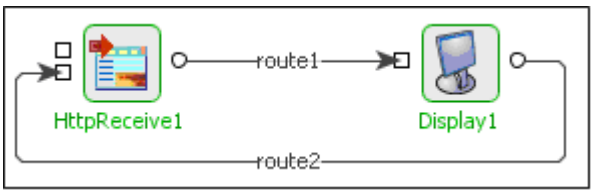


Figure 23: Sample Event Process demonstrating Scenario 1

Open %FIORANO_HOME%\esb\samples\EventProcesses\PurchasingSystem\resources\PurchasingSystem_Input.html in web browser. The web page looks like Figure 24. Click the Submit button to send the HTTP request. Check the port number by editing the html page, it should be same as provided in the component CPS.

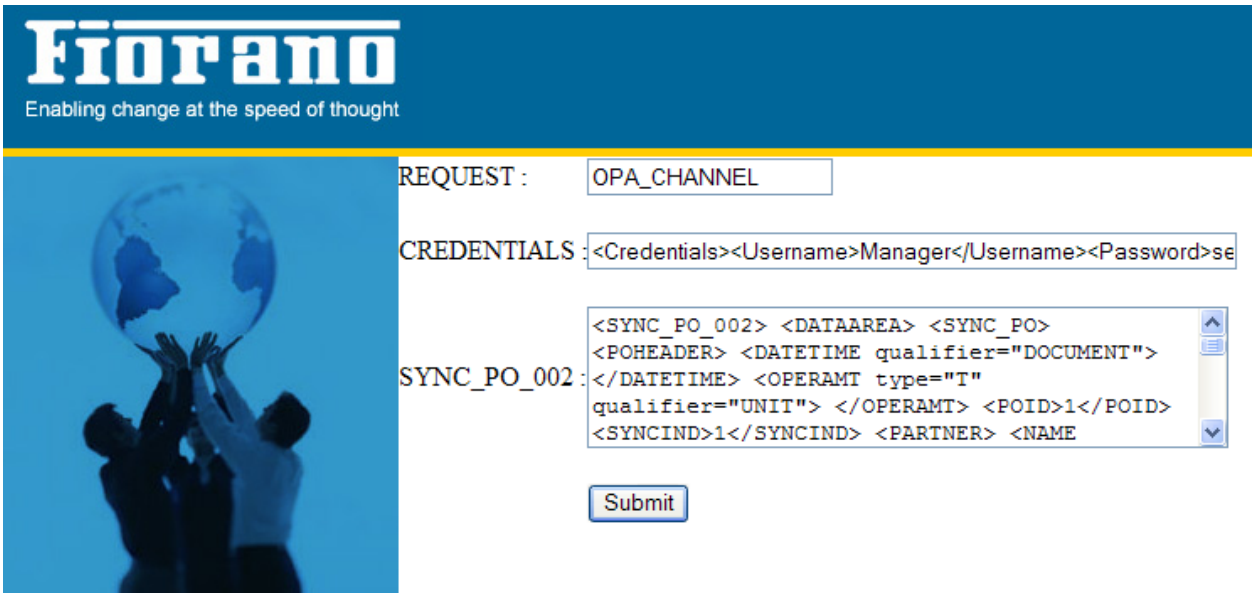


Figure 24: Sample HTTP Request

Now, HTTPReceive picks the parameters REQUEST and CREDENTIALS as configured in the Context details and sends the same XML back to the web page. Figure 25 shows the response sent by HTTP Receive.

Figure 25: Response sent by HTTPReceive adapter

4.13.2.5 Use Case Scenario

In the Purchasing System sample Event Process, the purchase details submitted from the web page are received using the HTTP Receive component.

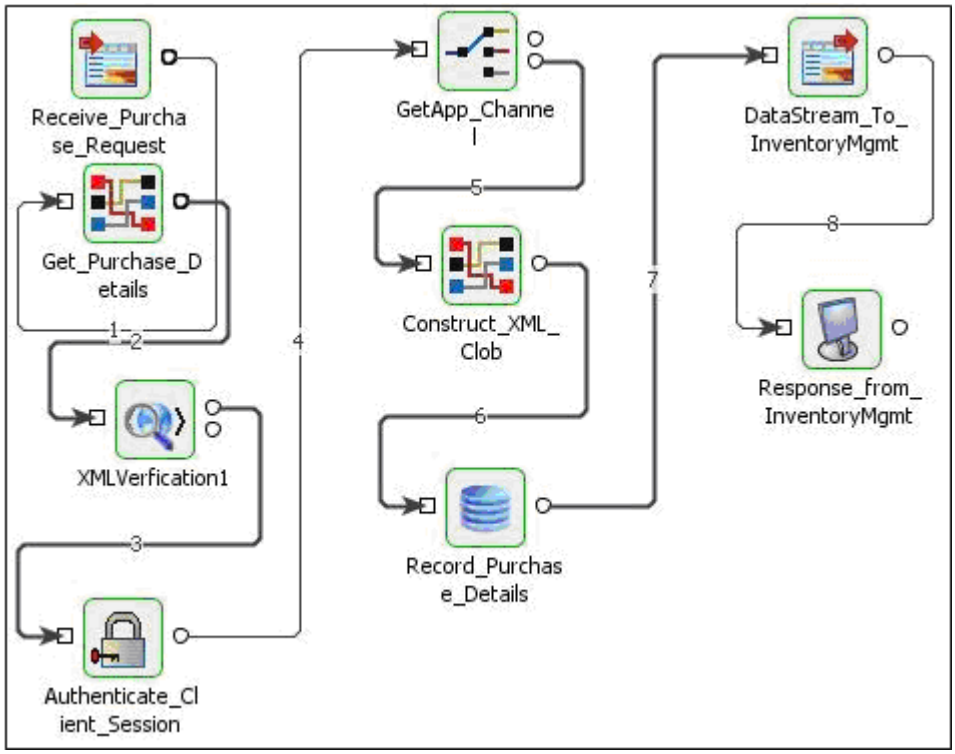


Figure 26: Sample use case scenario

4.13.2.6Useful Tips

The error ports of the components that receive the HTTP request through HTTP Receive could be connected to the Error port of HTTP Receive to report errors.

4.13.3 HTTP Stub

The HTTPStub component acts as an interface between an HTTP client and an Event Process and receives HTTP requests using the Hyper Text Transfer Protocol (HTTP). This component creates a context in the HTTP gateway hosted on the **peer server** based on the configuration provided.

The component receives client request on its output port and passes the request message to the connected components in the Event Process.

4.13.3.1 Configuration and Testing

The component has the following attributes which can be configured from its Configuration Property sheet. Figure 1 illustrates the panel with expert properties view enabled.

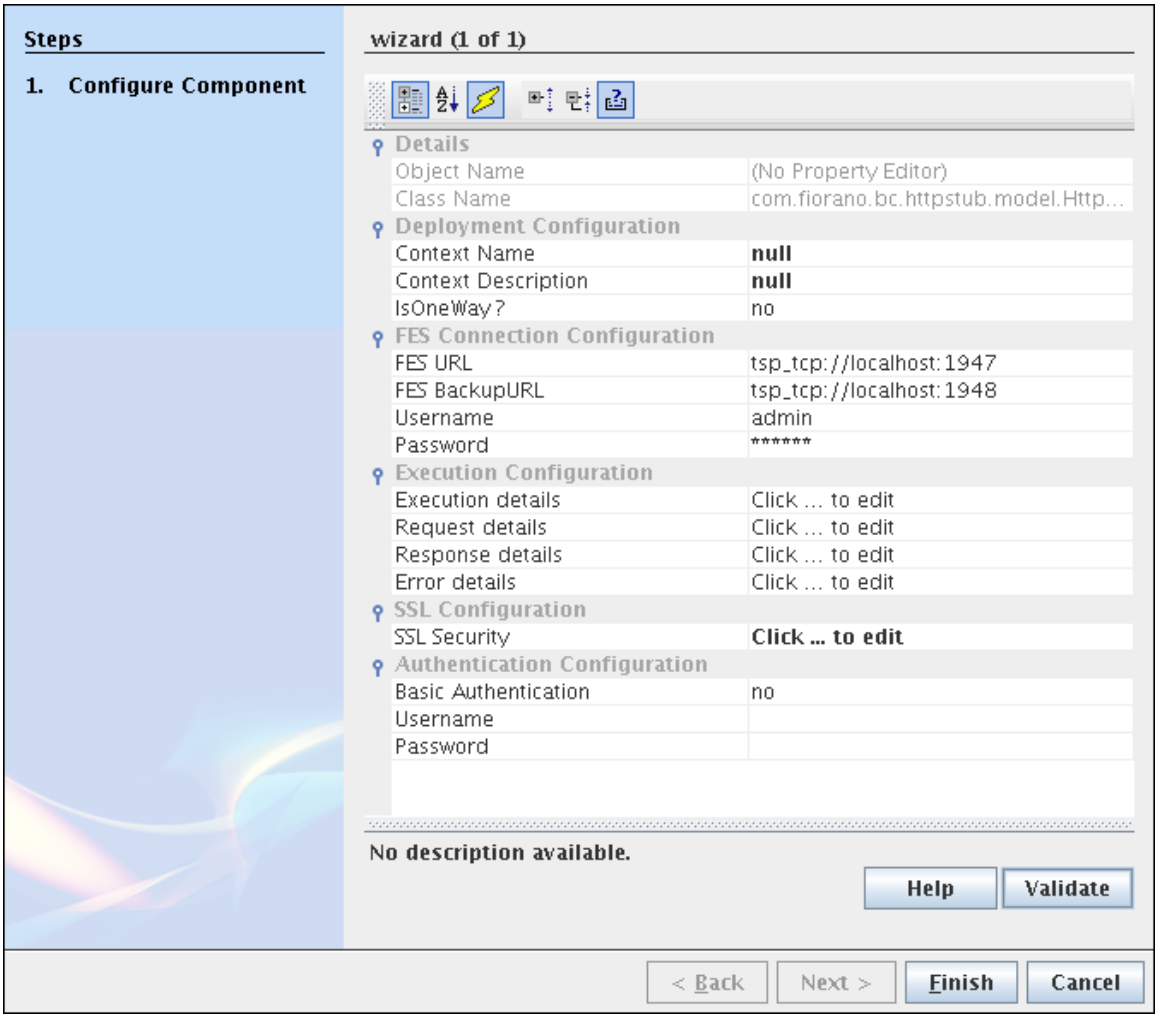


Figure 1: Configurable properties for HTTPStub component

4.13.3.1.1 Deployment Configuration

4.13.3.1.1.1 Context Name

The name of the context which will be created for this component. The Effective End Point URL for this context will be computed based on the context name as

```
http://<PEER_SERVER_IP>:<PEER_SERVER_HTTP_PORT>/<CONTEXT_ROOT>/<CONTEXT_NAME>
```

Peer server PEER_SERVER_HTTP_PORT is 1880 by default.

4.13.3.1.1.2 Context Description

The description of the context which will be displayed in HTTP gateway.

4.13.3.1.1.3 Is One Way

Determines whether a response has to be sent back to client invoking the service.

- **Yes:** No response will be sent back to the client. Only an output port REQUEST will be present to send the request to the Event Process.
- **No:** Response is sent after processing the request. Two input ports RESPONSE and FAILURE will be added in addition to the output port to send response, to receive response and error details from the Event Process.

Note: The properties **Response** details and **Error** details will not be present if this property is set to yes.

4.13.3.1.1.4 Admin End Point URI

URI of the admin context. This is not an editable property. This is used to deploy/undeploy HTTP contexts.

4.13.3.1.2 FES Connection Configuration

4.13.3.1.2.1 FES URL

The URL of Enterprise Server to which the Peer Server on which the component is running is connected.

4.13.3.1.2.2 Backup FES URL

The alternate URL that should be tried for connecting to the Enterprise Server if the Enterprise Server cannot be connected to using the URL mentioned against property **FES URL**.

Note: In case of Enterprise Servers in HA mode, this should point to Secondary Server URL if the primary is set against Server URL property and vice-versa.

4.13.3.1.2.3 Username

User name that should be used to connect to the Enterprise Server.

4.13.3.1.2.4 Password

Password that should be used to connect to the Enterprise Server.

4.13.3.1.3 Execution Configuration

4.13.3.1.3.1 Execution Details

The details necessary for execution of a request that is sent to the component can be configured using the UI shown in Figure 2.

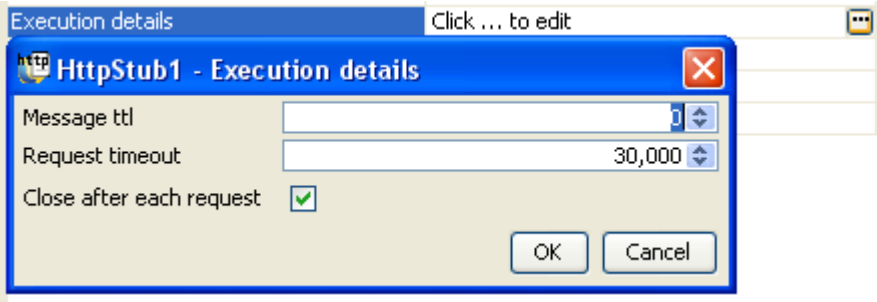


Figure 2: Configuration of Execution details

- **Message ttl:** The request received by the component is parsed and converted into a JMS message. This property indicates the time in milliseconds for which the JMS messages will be stored on the Peer Server. The default value 0 (zero) indicates that the messages will be stored on the server without any timeout.
- **Request Timeout:** If the property IsOneWaySend, the component accepts the response it reaches before the timeout period. If no response is received, on timeout Error message will be sent to the client.

  Note: Request will be processed by the connected components in the Event Process even after the timeout, but the response is not sent back to the client by HTTPStub. 0 (zero) indicates infinite timeout.
- **Close After each request:** If this option is chosen, all the resources (excluding connection) used by the component will be cleaned up after processing the request and recreated for the next request.

4.13.3.1.3.2 Request Details

When a HTTP request is received by the component, it is transformed into a JMS message and sent to the Event Process through the output port of the component based on the details configured using this property. These properties can be configured by clicking on the ellipsis against this property which opens request details editor as shown in the Figure 3.

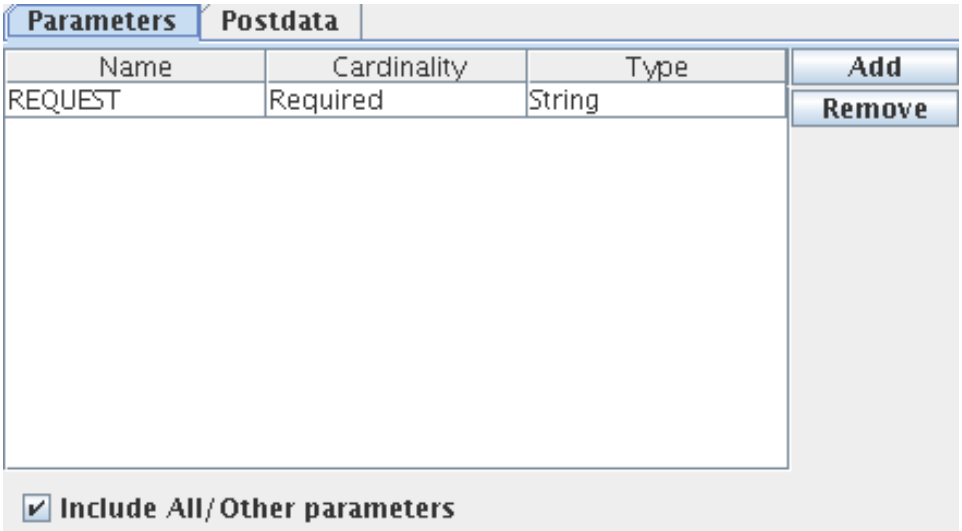


Figure 3: Request Details – Parameters

- **Parameters:** Parameters define the characteristics of the HTTP request data stream being parsed for converting the request to message. Parameters can be added by clicking `Add` button in the **Parameters** tab of the request details editor shown in Figure 3. Added parameters can be removed by clicking on the `Remove` button.
    - **Name:** The name of the parameter that is passed in the request. The name for corresponding element in the output schema will be set to this value.
    - **Cardinality:** If a parameter is definitely necessary for the processing of a request then it must be marked required, otherwise optional must be chosen. The cardinality of the corresponding element in the schema set on output port will be the same.
    - **Type:** The data type of he parameter can be specified as one of String, Boolean, Decimal or Integer. The XSD type of corresponding element will be same.
- **Include All/Other Parameters:** This option is used if all the parameters that are present in request stream have to be included, not only the parameters configured, but also the other parameters (if any) which are not configured are parsed from the request stream and set on the response stream.

  For each added parameter, a new element will be added as child of Params element in the schema of the output port REQUEST.
- **Post Data**

  If data is relatively large and is to be posted from the request, the way it has to be parsed must be specified by selecting the Post Data tab and providing details as shown in the Figure 4.

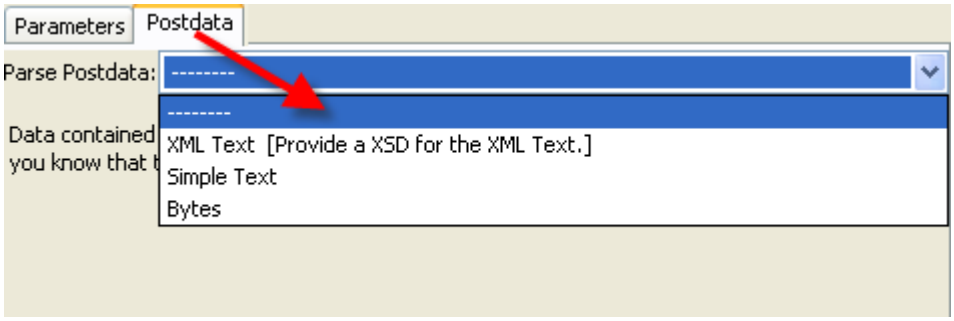


Figure 4: Request Details - PostData

- **---------- :** This option must be specified when the data is not required to be posted as part of the request. If chosen, then post data if passed as part of the HTTP request, will not be present in the request message and parameters must be added.
- **XML Text [Provide a XSD for the XML Text]:** This must be chosen if the data posted is XML that conforms to a schema. The schema can be provided using the schema editor. If chosen, an element XMLData will appear in the schema of output port which is of the same schema type.
- **Simple Text:** Data from the request stream will be considered as text not conforming to any schema. Hence, it will be added as CDATA. This option can be selected if the data is not required to be transformed using the Fiorano Mapper and needs to be transferred as is. If chosen, an element Data of string type will appear in the schema of output port.
- **Bytes:** Data from the request stream will be filled as bytes in the JMS Message. Use this option in case you need to send media files as binary data via HTTP.

  HTTP headers are received from the gateway as message properties with the header name prefixed with http_. Example http_Content-Type. For more information about HTTP Headers refer http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html.

4.13.3.1.3.3 Response Details

If the property Is One Way is set to no, the component sends the request message and waits for response from the Event Process. The response message that is received is converted to HTTP Response stream suitable for the invoking client based on the details provided here.

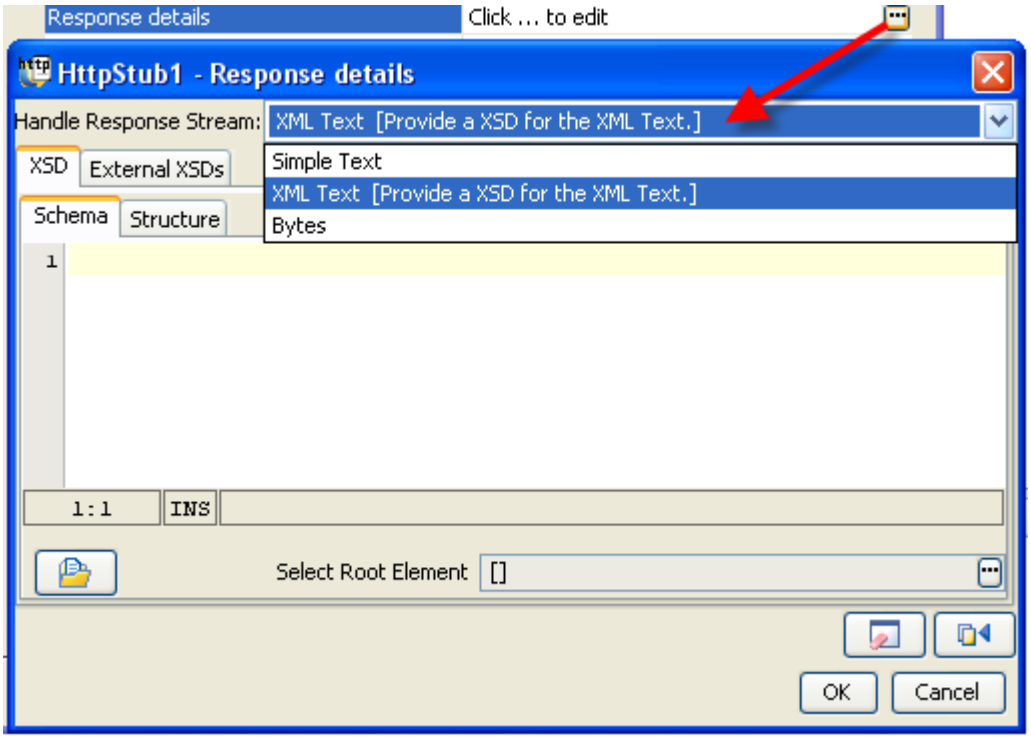


Figure 5: Response Details

- **Simple Text:** Text portion of the JMS Message will be read and put in the response stream.

  This option can be chosen when the invoking client does not expect a response that confirms to a specific schema. No schema is set on the input port RESPONSE.

- **XML Text [Provide an XSD for the XML Text.]**

  If the client expects the response to be compliant to a particular schema, then the schema is provided using the schema editor as shown in Figure 5. The schema is set as schema of the input port RESPONSE.

- **Bytes**

  Bytes portion of the JMS Message will be read and put in the response stream. This option can be chosen when the client expects the response message in the form raw bytes.

  Refer to section Input and Output for details about the effects of these configurations on input and output structures.

4.13.3.1.3.4 Error Details

Click the ellipsis button to launch an editor for providing these configurations.

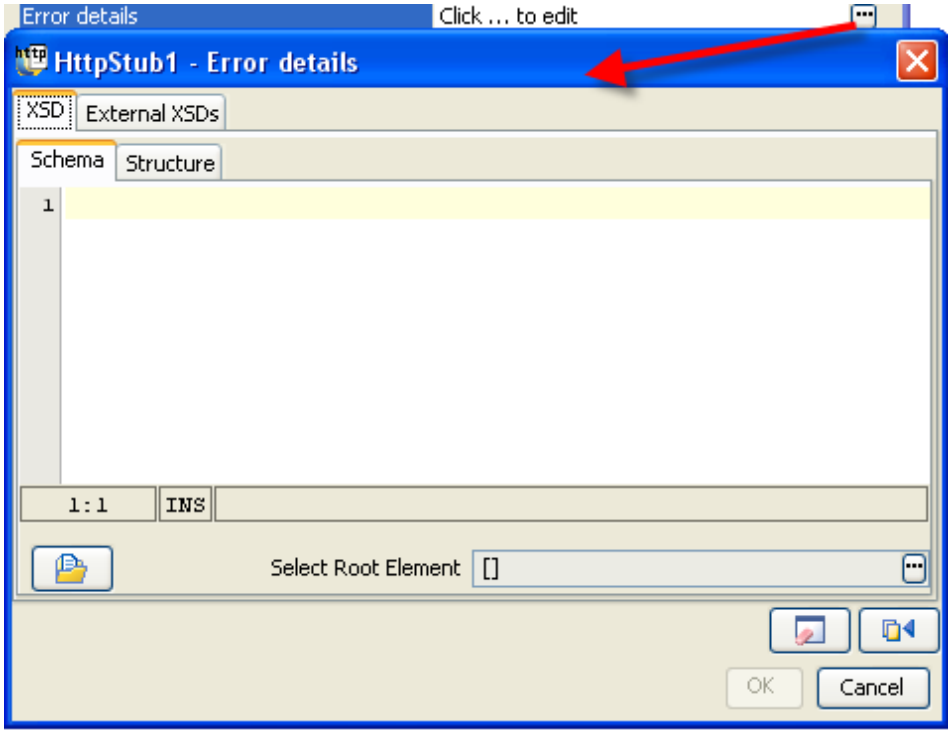


Figure 6: Error Details

If the client expects the error to be compliant to a particular schema, then the schema is provided using the schema editor as shown in Figure 6. The schema is set as schema of the input port FAILURE.

4.13.3.2 Input and output

4.13.3.2.1 Input

The input schema for the component is defined based on the configuration of **Response Details** property.

When Response is set to XML Text, then the schema provided is set as schema on input port RESPONSE. Else, if response is either set to Simple Text or Bytes, then no schema is set on the RESPONSE port.

4.13.3.2.2 Output

The output schema for the component is based on the configuration details provided for **Request Details**.

- When parameters are added (param1) and Post Data is set to -------- or when Post Data is set to Bytes, then schema set on port REQUEST is defined as shown in Figure 7 and a sample XML in Figure 8.

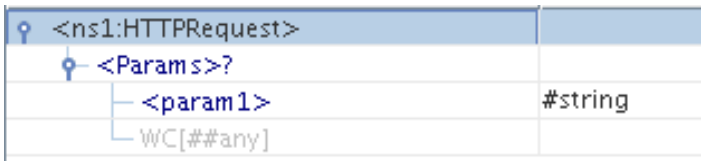


Figure 7: Output Schema with parameters and no post data

```
<ns1:HTTPRequest xmlns:ns1="http://www.fiorano.com/httpGateway">
   <Params>
      <param1>param1</param1>
   </Params>
</ns1:HTTPRequest>
```

Figure 8: Output XML with parameters and no post data

- When Post Data is set to Simple Text, then schema set on port REQUEST is defined as shown in Figure 9 and a sample XML in Figure 10.

Figure 9: Output Schema with Post Data set to Simple Text

```
<ns1:HTTPRequest xmlns:ns1="http://www.fiorano.com/httpGateway">
   <Params/>
   <Data>Data</Data>
</ns1:HTTPRequest>
```

Figure 10: Output XML with Post Data set to Simple Text

- When Post Data is set to XML Text, schema provided in Request details is set under XMLData element in output schema on port **REQUEST.** The schema set on port REQUEST is defined as shown in Figure 11 and a sample XML in Figure 12.

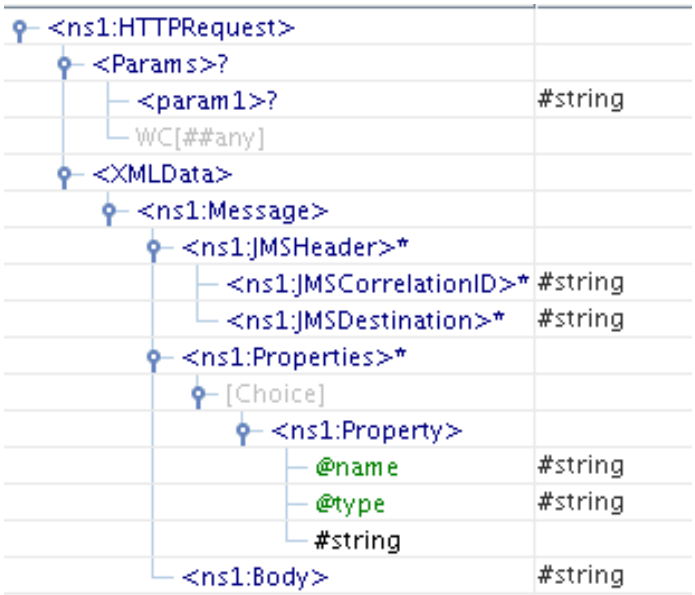


Figure 11: Output Schema with Post Data set to XML

```
<ns1:HTTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/JMSIn1/
   <Params>
      <param1>param1</param1>
   </Params>
   <XMLData>
      <ns1:Message>
         <ns1:JMSHeader>
            <ns1:JMSCorrelationID>JMSCorrelationID</ns1:JMSCorrelationID
            <ns1:JMSDestination>JMSDestination</ns1:JMSDestination>
         </ns1:JMSHeader>
         <ns1:Properties>
            <ns1:Property name="name" type="type">string</ns1:Property>
         </ns1:Properties>
         <ns1:Body>Body</ns1:Body>
      </ns1:Message>
   </XMLData>
</ns1:HTTPRequest>
```

Figure 12: Output XML with Post Data set to XML

4.13.3.3 Functional Demonstration

4.13.3.3.1 Scenario 1

Configure the component with **Response Details** set to Simple Text and parameters are configured in request details as shown in Figure 13.

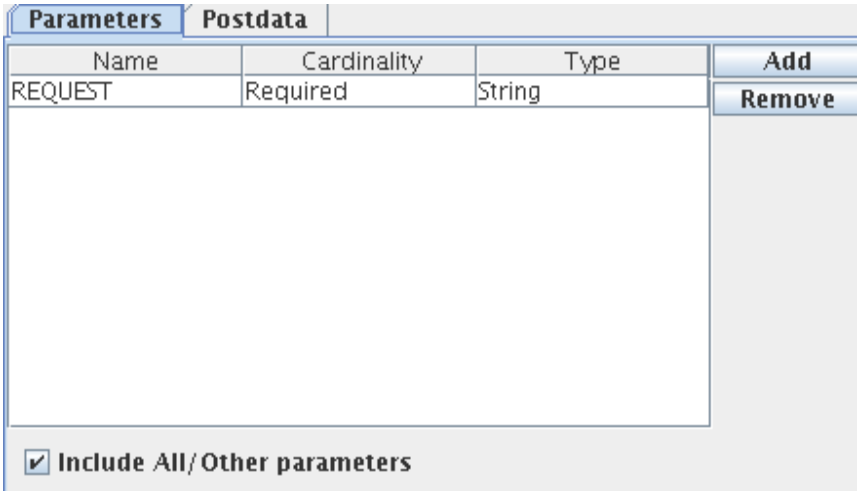


Figure 13: Sample Configuration of parameters

Figure 14 shows a sample event process with HTTPStub

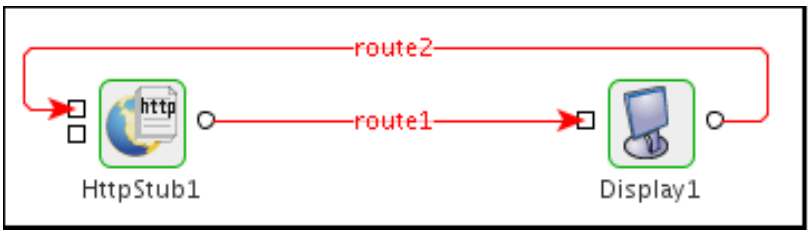


Figure 14: Demonstrating Scenario 1

When the flow is launched, HTTP context is deployed based on the configuration provided. Right-click the component and use the option **View HTTP Context** to view the deployed context. A HTTPAdapters component can be used to send in the request to the service. When a request is sent, a sample output message that is sent to output port **REQUEST** is shown below.

OutputMessage


```
<ns1:HTTPRequest
xmlns:ns1="http://www.fiorano.com/httpGateway"><Params><REQUEST>OPA_CHANNEL</
REQUEST></Params></ns1:HTTPRequest>
```

4.13.3.3.2 Scenario 2

Figure 15 demonstrates how HTTPStub can be used to deploy a mailing service as a context in HTTP Gateway.

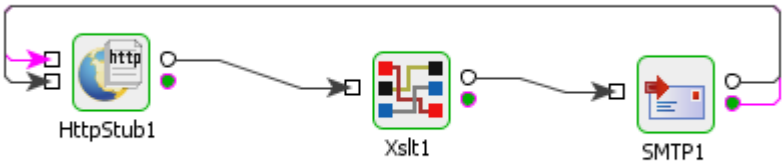


Figure 15: Deploying SMTP service using HttpStub

In the flow, HTTPStub is configured to take the Request as XML and the schema is set as same as that of input port of SMTP component. The XMLData element is mapped to the input schema of the SMTP component child to child recursively using Xslt Component. The Response is also chosen as XML and schema is set to be that of the output port of the HttpStub component. The error schema is set that of *ON_EXCEPTION port of SMTP component.

When the Event Process is launched, the context becomes active on the web server hosted on the peer on which the HTTPStub component is deployed. A HTTPAdapter can be configured to send in the request to this service.

4.13.3.4 Use Case Scenario

In Order Entry sample, a user is provided a web based interface to send a purchase order to a company. In case the order is accepted, HTTPAdapters is used to POST the order delivery request to a third party vendor. In an order entry scenario, the HTTPStub can be used for receiving orders as HTTP requests.

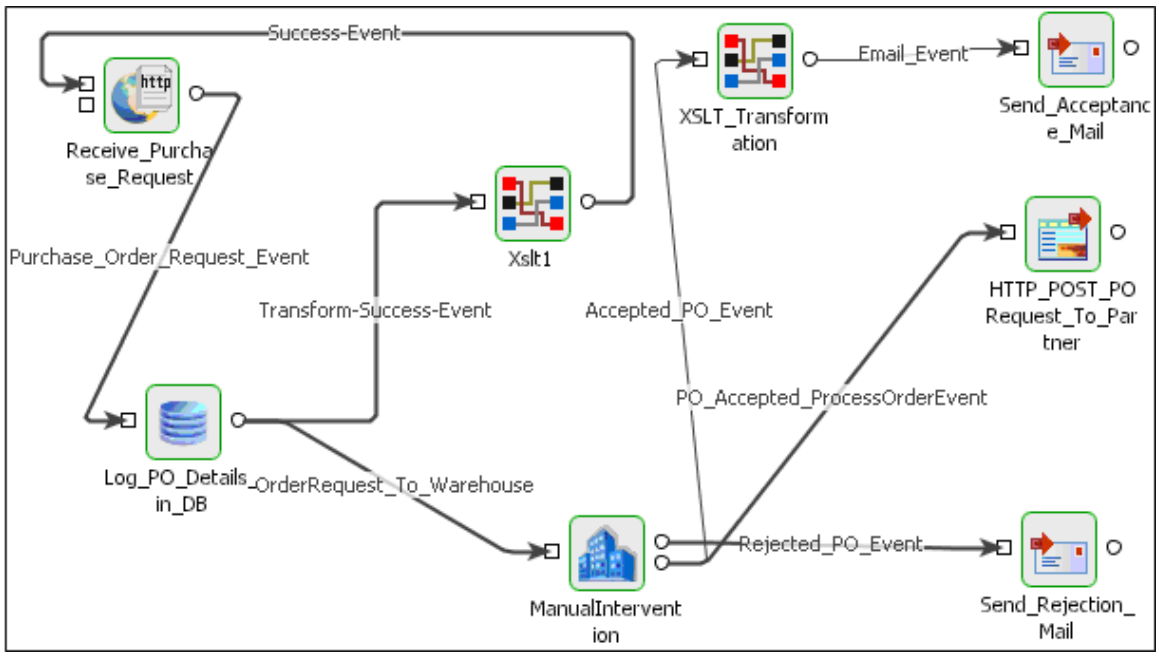


Figure 16: Order Entry

The Event Process demonstrating this scenario is bundled with the installer. The Http Stub is used instead of the HTTP Receive.

Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Fiorano Studio.

4.13.3.5 Useful Tips

- When HttpStub component is configured to launch on HA (High Availability) Peer Server

  If both Primary and Secondary servers are on the same machine

  Initially, if HttpStub is launched on the Primary Server and the generated HTTP Context contains Primary server's jetty port number. In case of failover, Primary Server shuts down and the secondary server becomes Active and relaunches the component. If the Secondary Server uses a different jetty port then the generated context URL will be changed since the jetty port is different. The clients have to be reconfigured to use new URL in this case.

  To avoid this situation, it is recommended to use same jetty ports for both primary and Secondary Peer Servers.

Jetty service will be started only after the server started successfully. In case of HA, only one server will be active at a given time and the Jetty Server will be running only in the active server and there will be no bind exceptions even if both the servers use same port number for Jetty.

If both Primary and Secondary servers are on different machines

In this case if the failover happens the hostname/IP address in the context URL will be changed. So the clients have to be reconfigured accordingly.

- HTTP headers are received from the gateway as message properties with the header name prefixed with http_. Example: http_Content-Type.
- Right-click the component and use the option **View HTTP Context** to view the deployed context.

4.13.4 SimpleHTTP

The SimpleHTTP component enables the user to get content from an external HTTP Server (Web Server). The component directly accepts the certificates. If the **Content** element is present in the input message, then Post method is used else Get method is used.

4.13.4.1 Configuration and Testing

4.13.4.1.1 Interaction Configurations

Attributes	
Accept Server Certificate	yes
Ignore Hostname Mismatch	yes
Cleanup resources after each document.	**no**
Connection Cache Threshold	**10**
Connection Removal Criteria	Least recently added
Validate Input	no
Target Namespace	http://www.fiorano.com/fesb/activity/Simpl...
Monitoring configuration	disabled: 5000 milli seconds

Figure 4.483: Sample SimpleHTTP configuration

4.13.4.1.1.1 Accept Server Certificate

When accessing https URLs, this property determines whether the server certificates should be accepted or not

- **yes** - Use this option if the server certificate should be accepted without any validation.
- **no** - Use this option if server certificate should not be accepted. An exception is sent to **ON_EXCEPTION** port.

4.13.4.1.1.2 Ignore Hostname Mismatch

- **yes** - Use this option to accept the certificate even if hostname in the certificate does not match with the hostname in the request URL.
- **no** - If hostname in the certificate does not match with the hostname in the request URL, exception is thrown.

4.13.4.1.1.3 Cleanup resources after each document

- **yes** - Use this option if the connection created for the request should be discarded after executing the request without storing it in the cache.
- **no** - Use this option to store the connection in a cache for further use without creating a new connection every time for the same URL. If this option is selected, 'Connection Cache Threshold' will be visible.

Note: If no is selected, there will be an attribute 'CacheConnection' added to the element 'URL' in the input schema. If this 'CacheConnection' value is 'false' in input XML, the connection will not be stored in the cache.

4.13.4.1.1.4 Connection Cache Threshold

This option specifies the threshold limit; that is the maximum number of connections that can be stored in the cache. '-1' is the default value, which means connections are not removed from the cache. Any other positive integer can be specified to limit the number of connections stored. In this case, Connection Removal Criteria option will be visible.

4.13.4.1.1.5 Connection Removal Criteria:

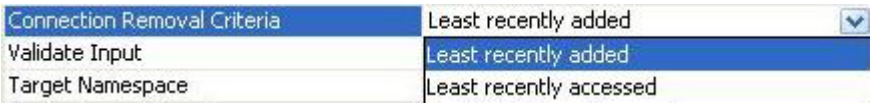


Figure: Connection Removal Criteria

This option specifies the criteria to remove connections from the cache if the threshold limit exceeds.

- **Least recently added** - The oldest entry in the cache is removed.
- **Least recently accessed -** The entry that has been in the cache for the longest time without being accessed.

Configuration shown in the **Figure 4.483** can be tested within the CPS by clicking on Test button.

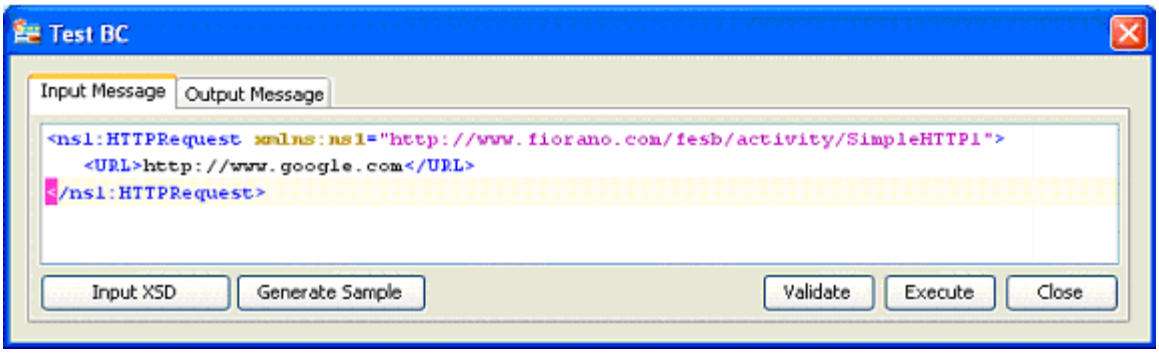


Figure 4.484: Sample SimpleHTTP input message

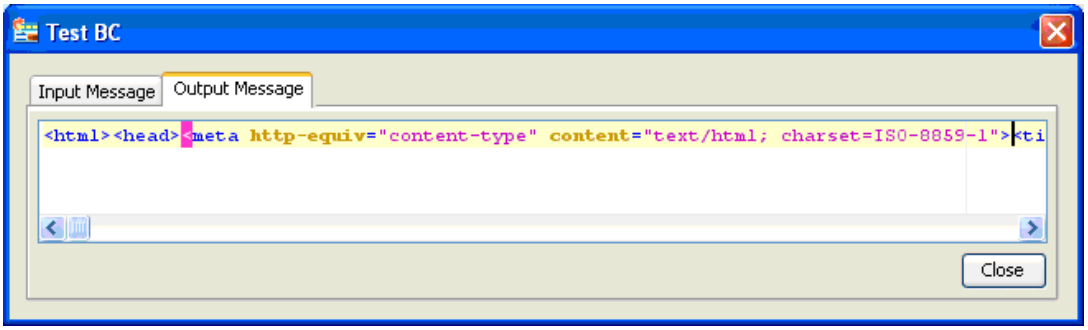


Figure 4.485: Sample SimpleHTTP output message

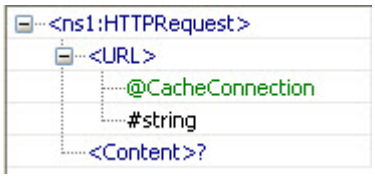


Figure: Input Port Schema Structure

Input Schema

Schema Element	Description
<URL>	URL from which content has to be fetched. If we set "Cleanup resources after each document" property to "no", then the attribute "CacheConnection" appears in this element (as shown in Figure **Input Port Schema Structure**). This attribute accepts two values "true" and "false". If "true" is set to "CacheConnection" attribute, then the connections created are cached and these cached connections will be used when required. If we set "CacheConnection" attribute to "false", then the component clears the connection after each request.
<Content>	Content to be posted (optional)

4.13.4.2 Functional Demonstration

4.13.4.2.1 Scenario 1

This scenario demonstrates a sending simple HTTP request to a server.

Configure the SimpleHTTP as described in Configuration and Testing section and use Feeder and Display component to send sample input and check the response respectively.

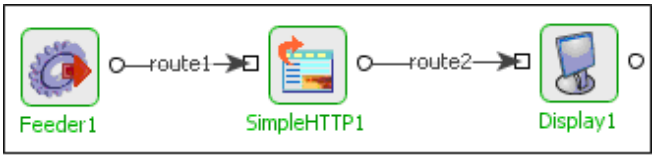


Figure 4.486: Demonstrating Scenario 1 with sample input and output

Sample Input

```
<ns1:HTTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/SimpleHTTP1">
   <URL>http://www.google.co.in/advanced_search?hl=en</URL>
</ns1:HTTPRequest>
```

Sample Output

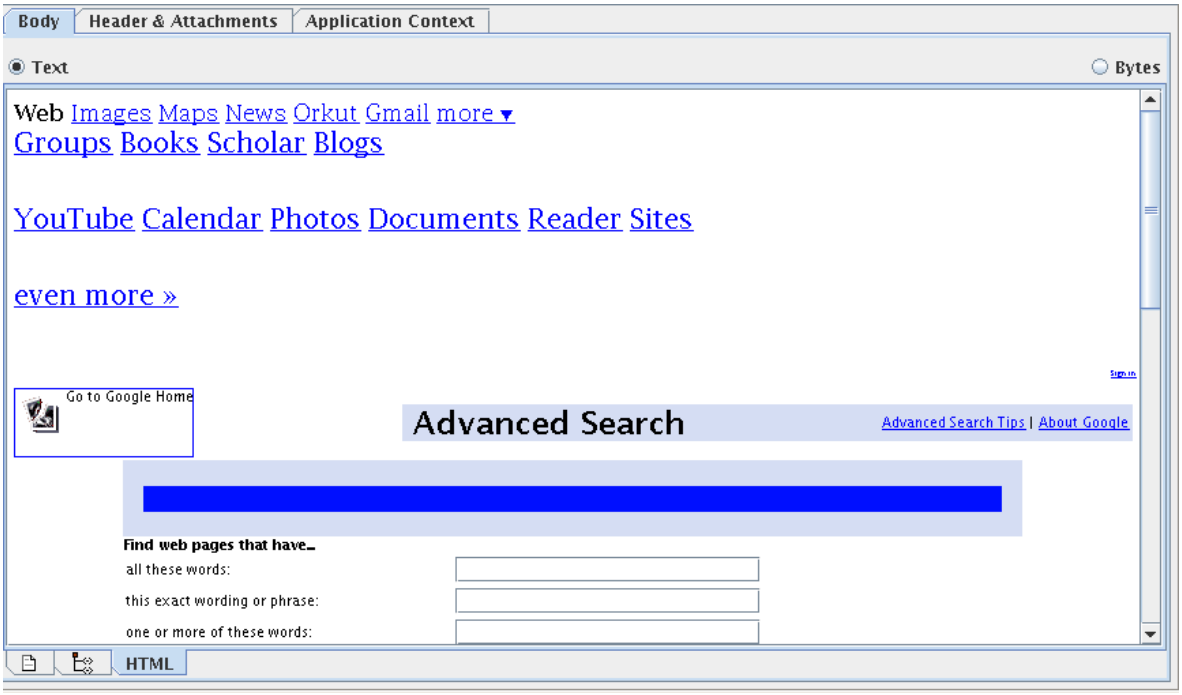


Figure 4.487: HTML display of Output in Display Component.

4.13.4.2.2 Scenario 2

The scenario demonstrates how a SimpleHTTP component can be used to post some data onto a Web Server hosted by a HTTPReceive component. The HTTPReceive component is hosting on port 9999 on the localhost machine and is configured to receive post data as simple text and send it back to the client as simple text.

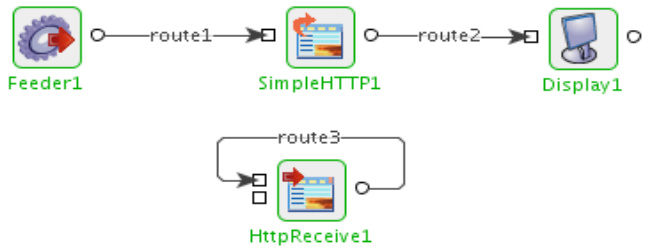


Figure 4.489: Demonstrates using SimpleHTTP component to post data

Configure the SimpleHTTP as described in Configuration and Testing section and use Feeder and Display component to send sample input given below.

Sample Input

```
<ns1:HTTPRequest xmlns:ns1="http://www.fiorano.com/fesb/activity/SimpleHTTP1">

   <URL>http://localhost:9999/index.html</URL>

   <Content>Sample Content</Content>

</ns1:HTTPRequest>
```

Sample Output

```
Sample Content
```

4.13.4.3 Useful Tips

- For providing header properties like Content-Type or Content-Length, set the required properties in the input message with a prefix http_. For example, http_Content-Type.
- For providing parameters, set the required properties in the input message as properties with a prefix param_. For example, param_myproperty.
- To Post data we need to provide the header property Content-Type by setting property http_Content-Type to appropriate value on the input message. For example if we need to post simple text content then set the property http_Content-Type to text/plain.
- When connecting to secured sites, the component accesses truststore from different locations in different launch modes -
    - Separate Process - %JAVA_HOME%\jre\lib\security\cacerts
    - In-memory - %FIORANO_HOME%\esb\server\profiles\certs\jssecacerts

  'jssecacerts' in %FIORANO_HOME% does not have trusted certificates required by most of the secured sites. Hence, to connect to secured sites when component is launched in-memory, either the trusted certificates should be added to 'jssecacerts' or the property 'javax.net.ssl.trustStore' (in %FIORANO_HOME%\esb\server\bin\server.conf, if FPS is launched using server.bat/server.sh and %FIORANO_HOME%\esb\fps\bin\fps.conf, if FPS is launched using fps.bat/fps.sh) should be set to %JAVA_HOME%\jre\lib\security\cacerts.

4.13.5 RESTStub

This component is used to expose an Event Process or a part of Event Process as a RESTful web service deployed in the Peer Server. A RESTful web service is a simple web service implemented using HTTP and the principles of REST. It is a collection of resources. WADL for the service deployed can be fetched when this component is running and the URL of the WADL can be used by clients to access the Web Service. Each instance of the component hosts a single web service. The component can be configured in three ways:

- Manually defining the resources and methods that are supported by the service in the CPS of the component.
- Loading a predefined WADL and editing the populated RESTfulweb service details if required
- For simple case of exposing single resource the resource URL can be specified and WADL can be generated using the analyze URI functionality.

The WADL generated for the component will differ from the WADL provided in following aspects:

- Base path of resources element
- If the provided WADL contains multiple 'resources' elements, only the first resources element will be loaded and used.

The component generates request and response port for each method present in the configured web service.

During the component launch, it checks if a web service with the same web service name is already deployed. If no other service with same name exits, the web service is deployed to the Jetty Server running in Peer Server that hosts the component.

The web service deployed in the Jetty accepts requests from RESTful web service clients and sends the request using internal JMS client to the request port of the operation there by injecting the request into the Fiorano Event Process for processing the request. On processing the request, the response should be generated and sent to the corresponding response ports. The web service receives the response from the internal JMS client and in turn sends the response as HTTP response to the client that made the request.

Note: The component acts as a helper to expose an event process as a RESTful service. The developer of an event process will have to take care that the functionality is exposed as per the guidelines of REST architecture.

4.13.5.1 Configuration

The component can be configured using the custom property sheet as shown in the following sections.

4.13.5.1.1 WADL Definition

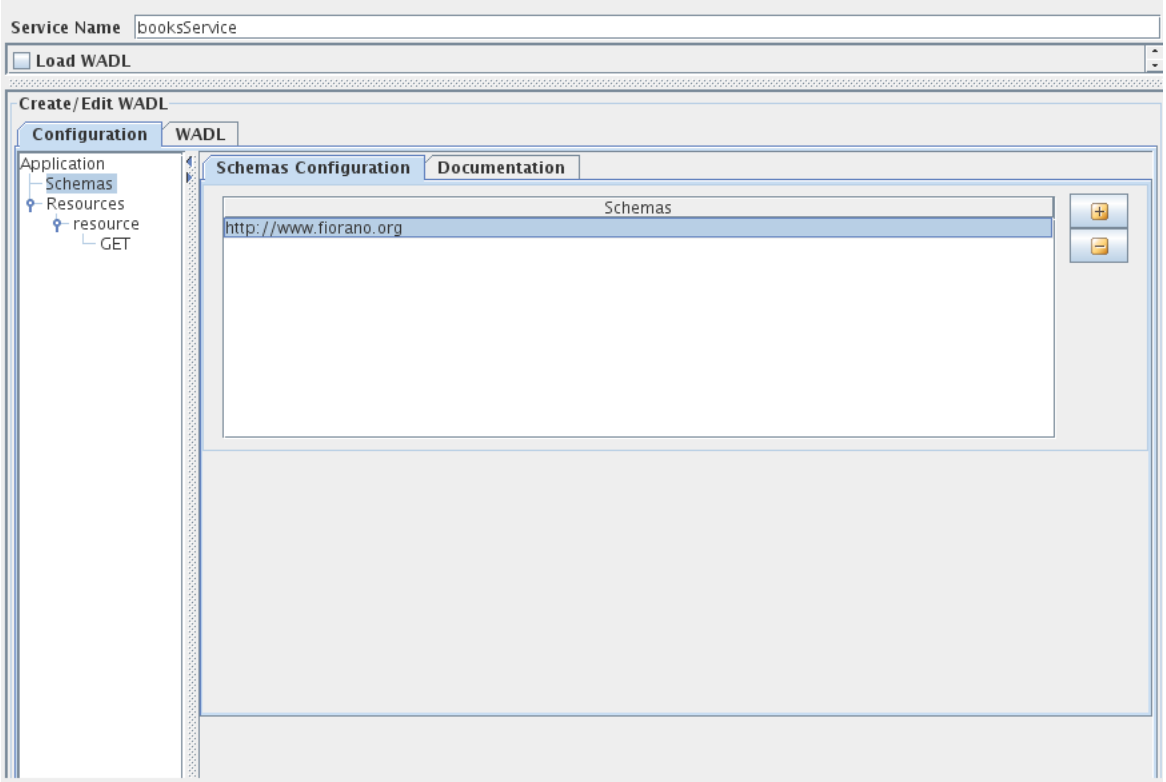


Figure 1: Configuration Property Sheet

This page is used to define the web service using a user interface for WADL User can create a WADL by manually defining different aspects of the service or can load an existing WADL and edit the configurations. For detailed information on WADL please refer http://www.w3.org/Submission/wadl

Service Name

The name of the service with which the event process functionality is exposed as RESTful service can be specified here. The base path for the resources defined by this component is calculated with the help of the service name as follows

```
http(s)://<Peer Server IP>:<Peer Server Port>/restgateway/services/<service name>
```

The URL can be viewed by selecting the option "View WADL" from the context menu of the component in studio.

4.13.5.1.1.1 Loading WADL

WADL file can be loaded to define the configuration as shown in **Figure 2**. To load an existing WADL file, the checkbox **Load WADL** must be selected which shows the option below.

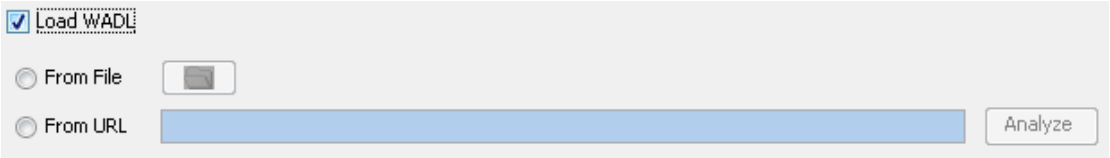


Figure 2: Loading WADL

From File

The WADL file can be loaded by selecting the corresponding radio button using the file dialog that opens up by clicking the icon. If the loaded WADL file has more than one resources element defined, then the user is prompted to select one for configuration as shown in **Figure 3**.

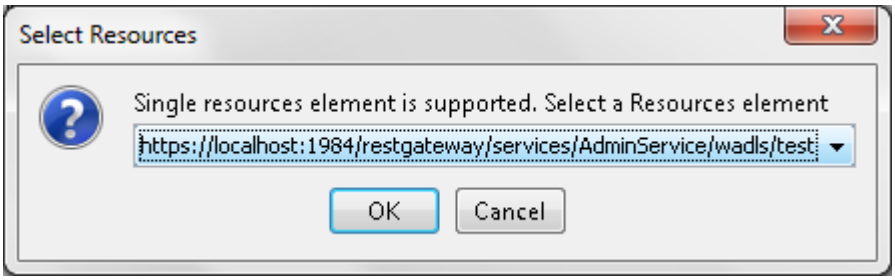


Figure 3: Resources selection

From URL

The WADL also be loaded by analyzing an URL. This option can be used to avoid creating a resource hierarchy from scratch. A Valid URL has to be specified with protocol. Resource elements are generated based on the path segments and a GET method is generated for the last path segment. Any Query parameters that are part of the resource will be added as parameters to the method's Request with param style as Query and default value as value specified.

The following figure illustrates the generation of WADL for the sample yahoo news search API URL.

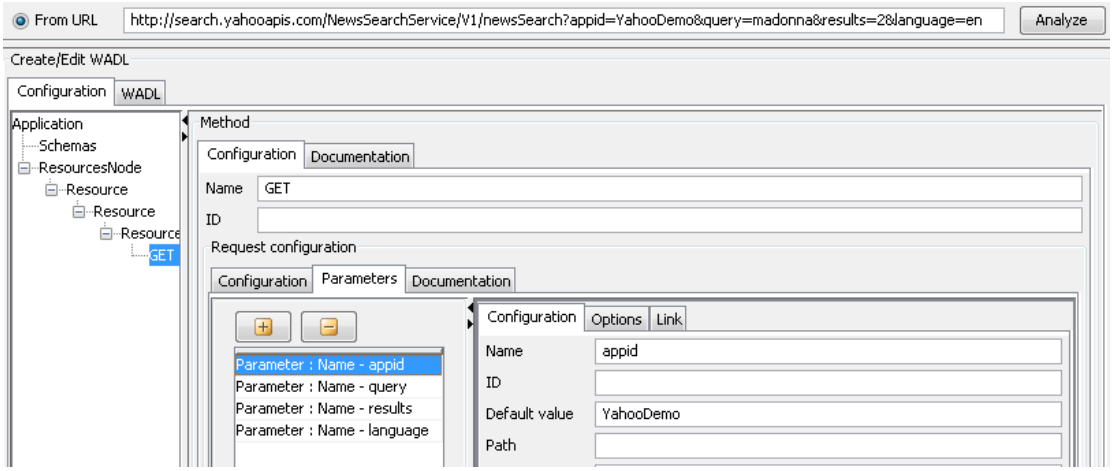


Figure 4: Analyzing URL

4.13.5.1.1.2 WADL Definition Views

Details of the selected WADL are shown in 2 different views – Configuration and WADL. However, web service can be configured using Configuration view only.

4.13.5.1.1.2.1 Configuration View

The RESTful service can be configured by selecting the nodes in the tree on the left side. When a node is selected in the tree the panel corresponding to its configuration is shown in the right side pane. The details of different panels corresponding to the nodes are as explained in the following sections.

4.13.5.1.1.2.1.1 Schemas

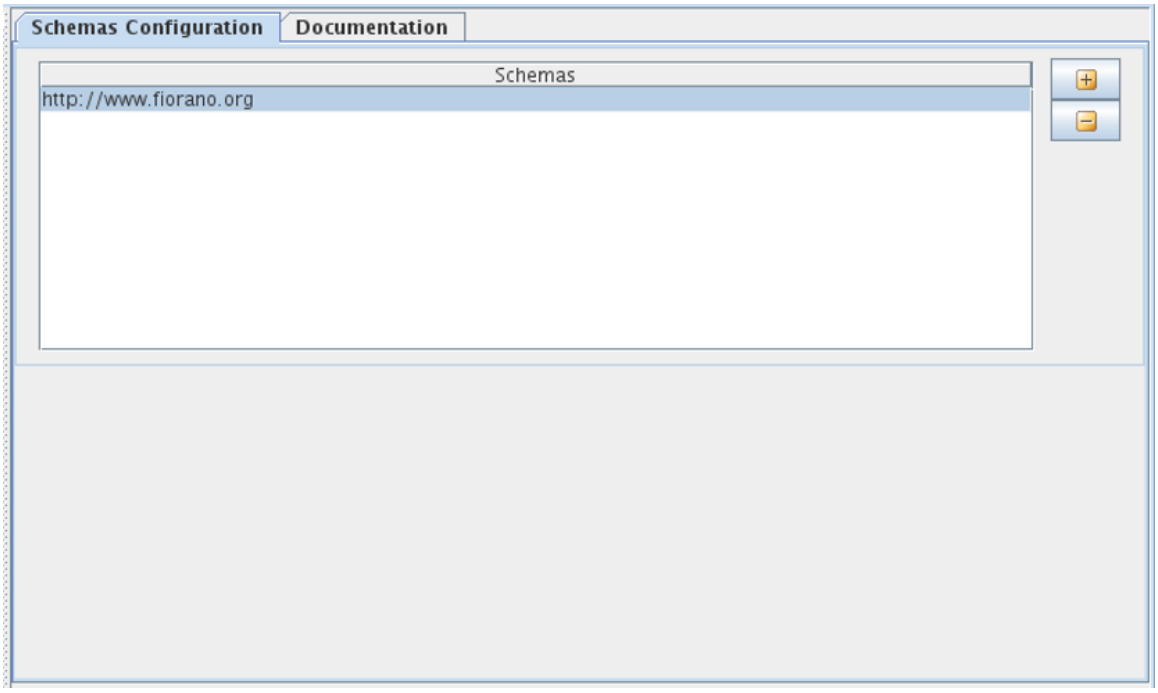


Figure 5: Schemas Configuration

The XML schemas which are required for the request and response messages have to be specified in this panel. A schema can be added by clicking the button in the table shown in the **figure 5**.

There are 3 ways to add schema to schema repository; Load from File, Load from Port Schemas and Load from Schema Repository. When **Load from File** is chosen, selected content in the schema file is shown in the area given. When **Load from Port Schemas** is selected, schema is directly added to the schema repository. When **Load from Schema Repository** is selected, schema can be fetched from schema repository. In this case, the required schemas have to be added to the schema repository using Studio.

Note: The elements dialog in representation will be populated from the schemas that are specified in this panel.

4.13.5.1.1.2.1.2 Resources

The resources node will act as parent of all the resources that are added as part of this service. It provides the base path for all these resources.

In case of RESTStub component the base path is automatically generated during the deployment of the component. The value will be replaced by

```
http(s)://<Peer Server IP>:<Peer Server Port>/restgateway/services/<service name>
```

at runtime.

4.13.5.1.1.2.1.3 Resource

A resource logically represents a entity whose operations have to be exposed using the RESTful service. It may optionally be used to specify just a path segment for a child URL with no methods defined on the resource itself.

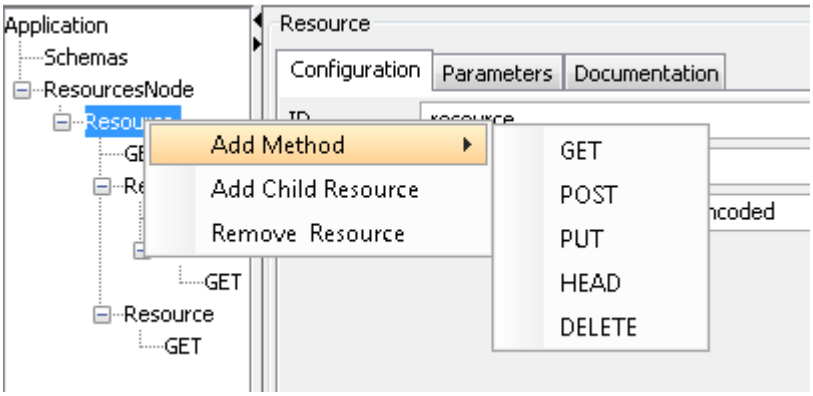


Figure 6: Adding child resources and methods

A resource can be added to a resources/resource element by selecting the option Add Resource/Add Child Resource respectively as shown in **Figure 6.**

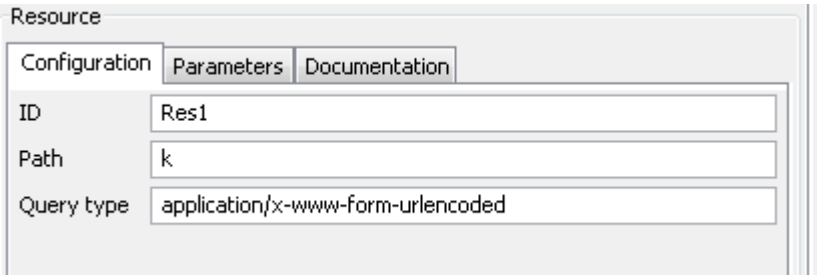


Figure 7: Resource configuration

The configuration of a resource contains fields as shown in **Figure 7** as explained below.

ID

The identifier for the resource can be specified using this element. This identifier is used in the generation of port names and schemas corresponding to the methods that are defined on this resource or any of its child resources.

The ID has to be unique among its siblings. The complete ID of a resource will be computed using information of parent IDs as below.

Example:

If the resource with ID "parent" is added to Resources element (using Add Resource option) and a child is added to the resource (using "Add Child Resource" option) with ID "child", the complete ID of the resource will be **parent__child**.

Path

The path of the resource, relative to its parent resource/resources element has to be specified against this field.

The path of a resource has to be unique among the siblings of a resource. The path of a resource will be computed including information of parent paths as below.

Example:

If the resource with path "parent" is added to Resources element (using Add Resource option) and a child is added to the resource (using "Add Child Resource" option) with path "child", the actual path of the resource will be <Resources Base Path>/parent/child.

Templates are allowed in the path element. A template will have the form {<template name>}. When a template element is used a parameter with same name and style as Template have to be defined. The request message on the port corresponding to this resource will have the value from the URL as explained below.

Example:

If a resource has ID "resource" and path as {studentID}, then for a request to URL <Parent Path>/178 will result in request message having a parameter as follows

```
.. <resource>
        <studentID>178</studentID>
    </resource>..
```

Please refer to generation of port schemas for more information.

QueryType

The property defines the media type for the query component of the resource URI. The value defaults to 'application/x-ww-form-urlencoded' if not specified which results in query strings being formatted as specified in section 17.13 of HTML 4.01.

4.13.5.1.1.2.1.4 Method

Methods that resource supports can be added by using the context menu of a resource as shown in **Figure 6**.

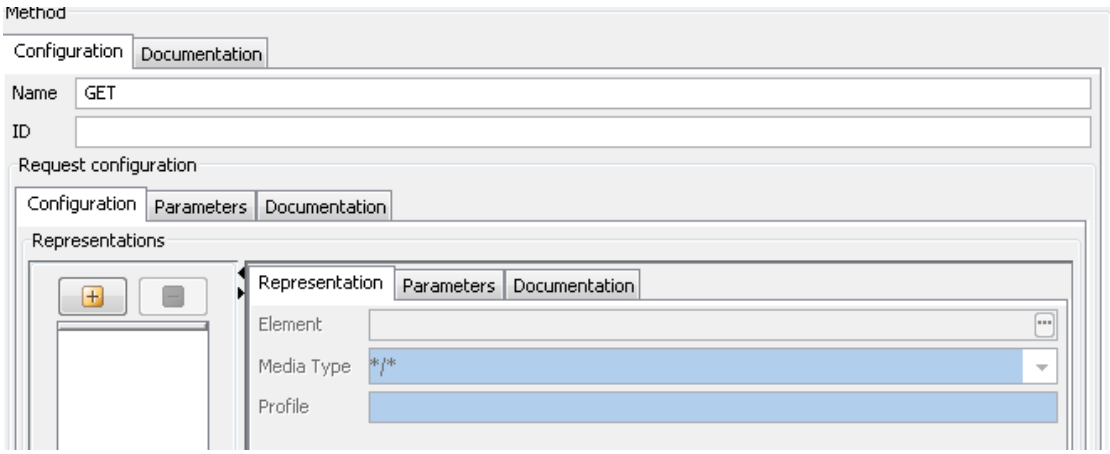


Figure 8: Method Configuration

The various attributes of a method are as shown in **Figure 8**. The request configuration of a method can also be configured in this panel.

Name

The attribute hold the name of the method that the resource supports. The attribute is non editable. The name will be used in the generation of ports and schemas for the component.

Two methods on a resource can have the same method name. In such case ID has to be specified for the method to differentiate between the methods.

ID

The method ID acts as a differentiator between methods of a same resource with same name. This is required only if the resource defines more than one method with the same name.

4.13.5.1.1.2.1.5 Request configuration

The request for a message can be configured in the panel corresponding to the method as shown in **Figure 8.** As part of request one can define multiple representations. Different representations can have different media types and elements defined. The media Type requested will be sent as part of the request message. It's the responsibility of the event process developer to return the appropriate content type based on the request.

Configuring representations is explained in detail in section 4.13.5.1.1.2.1.6

Similarly the request parameters can be configured for a particular method by using the parameters tab. Configuring parameters is explained in detail in section 4.13.5.1.1.2.1.7

4.13.5.1.1.2.1.6 Response configuration

A method can have multiple responses. Different responses cater to different sets of status codes. A response element can be added by using the context menu on a method as shown in **Figure 9.**

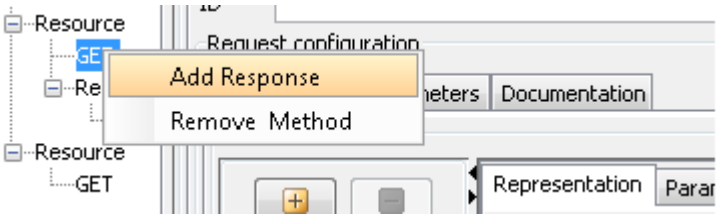


Figure 9: Adding Response

The response can be configured using the panel as shown in **Figure 10.**

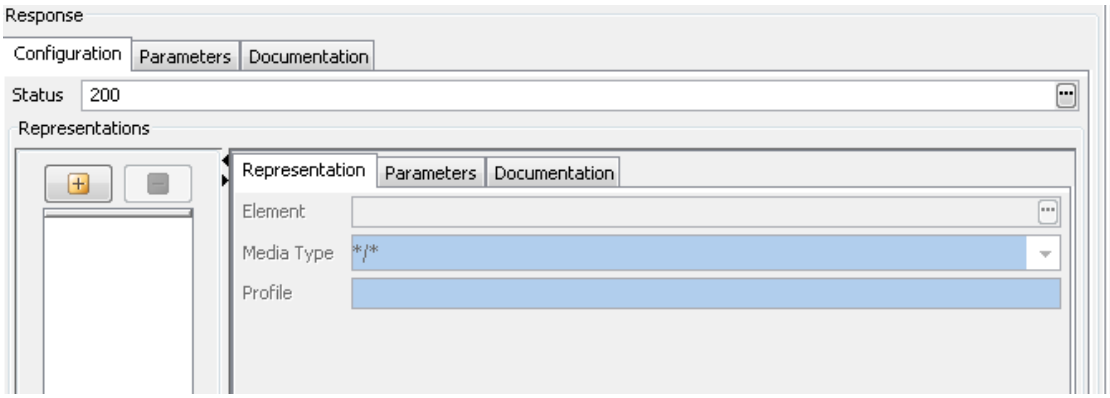


Figure 10: Response configuration

Status

The status of response can be space separate list of the standard HTTP status codes or any custom codes defined in the application. The status codes can be selected among standard HTTP status codes by clicking on ellipsis against this property as shown in **Figure 11**

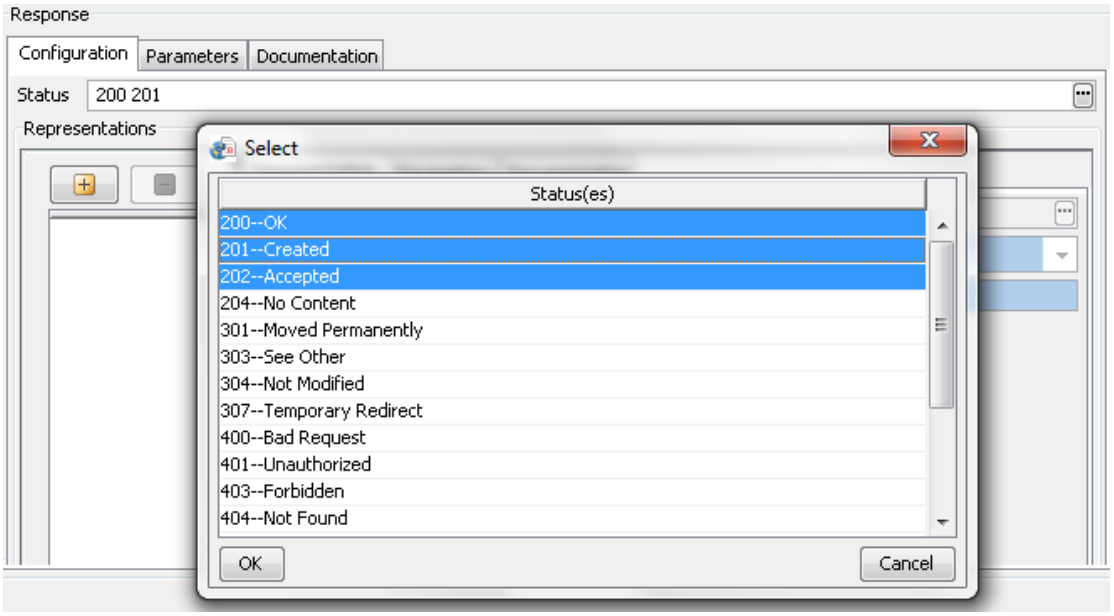


Figure 11: Configuring status

Representations

Similar to a request, each response (corresponding to a set of statuses) can have multiple representations. The representations provide different views of same response as defined by the media type.

4.13.5.1.1.2.1.7 Representation configuration

Multiple representations can be defined for both request and response of a method as shown in **Figure 12.**

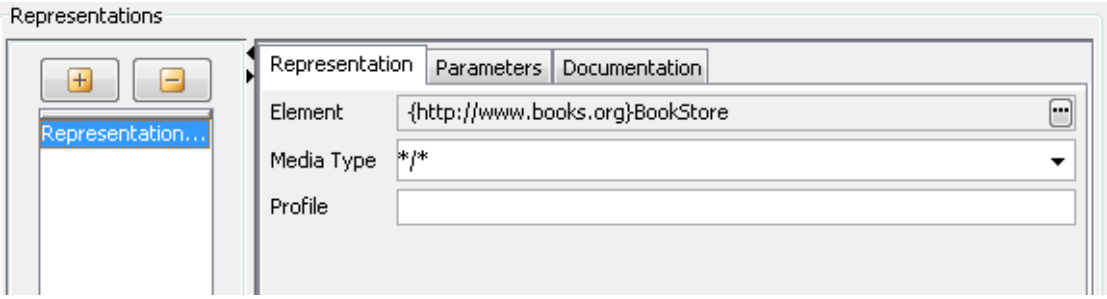


Figure 12: Representation Configuration

Element

The element can be used to specify the xml schema of an element corresponding to the representation. The schema corresponding to the element has to be added to the schema repository using studio and an include has to be added using the Schemas panel as explained in section 4.13.5.1.1.2.1.1 Schemas.

The element can then be selected by clicking on the ellipsis and selecting element as shown in the **Figure 13: Configuring elements.**

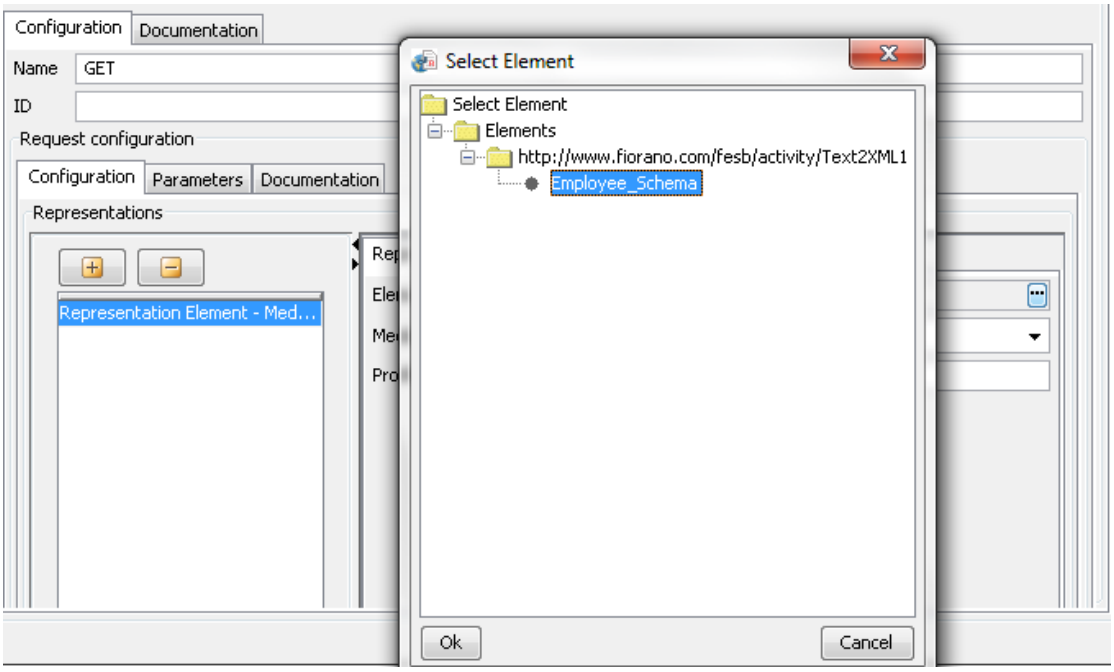


Figure 13: Configuring elements

MediaType

The media type of a representation defines the type of the content that is present in the representation. The media type can be configured by selecting the appropriate one from the drop down or manually specifying in the combo as shown in **Figure 14: Media Type configuration**

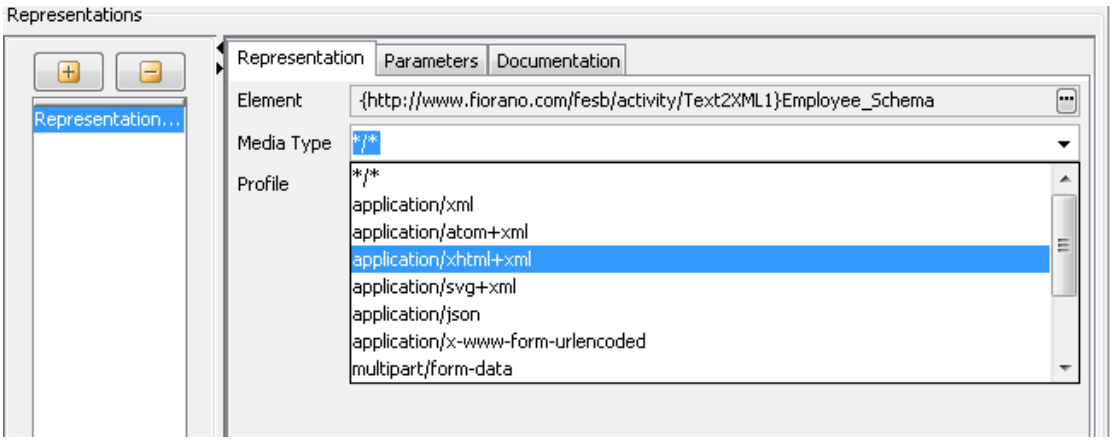


Figure 14: Media Type configuration

Profile

The attribute is similar to the HTML profile attribute, gives the location of one or more meta data profiles, separated by white space. The meta-data profiles define the meaning of the rel and rev attributes of descendent link elements

4.13.5.1.1.2.1.8 Parameter configuration

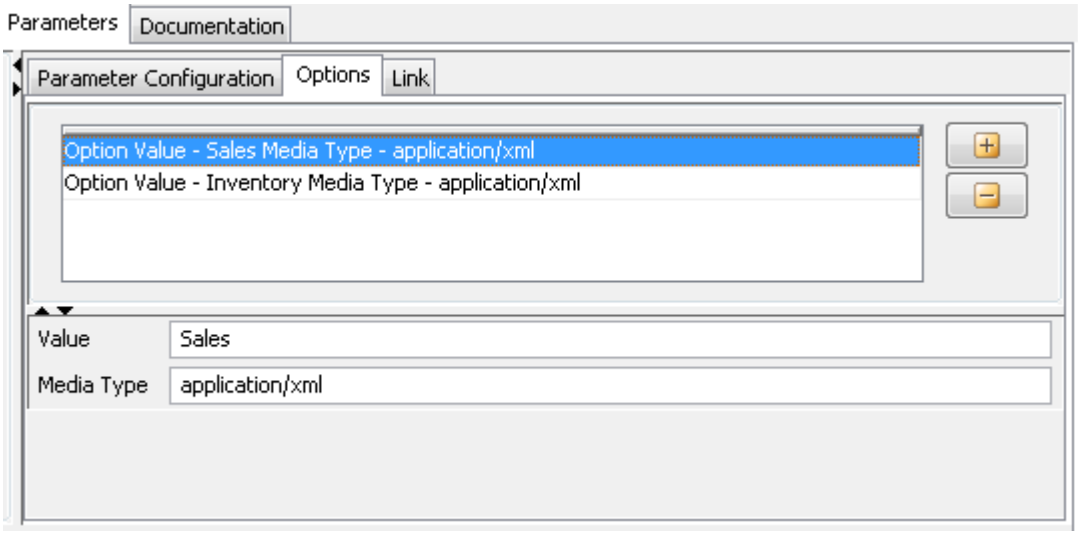


Figure 15: Parameter Configuration

Parameters are present at different levels in a resource tree. These can be configured using the UI as shown in **Figure 15**.

The different attributes of a parameter are as follows

Name: The name of the parameter as an xsd:NMTOKEN. Required.

Style: Indicates the parameter style, only the allowed values are populated in the combo box of a certain configuration.

Type: Optionally indicates the type of the parameter as an XML qualified name, defaults to xsd:string.

Default: Optionally provides a value that is considered identical to an unspecified parameter value.

Path: When the parent element is a representation element, this attribute optionally provides a path to the value of the parameter within the representation. For XML representations, use of XPath 1.0 is recommended.

Required: Optionally indicates whether the parameter is required to be present or not, defaults to false (parameter not required).

Repeating: Optionally indicates whether the parameter is single valued or may have multiple values, defaults to false (parameter is single valued).

Fixed: Optionally provides a fixed value for the parameter.

Options: An option element defines one of a set of possible values for the parameter represented by its parent parameter. An option has the following attributes:

Value: A required attribute that defines one of the possible values of the parent parameter.

mediaType: When present this indicates that the parent parameter acts as a media type selector for responses. The value of the attribute is the media type that is expected when the parameter has the value given in the value attribute.

Link: A link element is used to identify links to resources within representations. A link element is a child of a parameter whose path attribute identifies the portion of its parent representation that contains a link URI.

4.13.5.1.1.2.1.9 Documentation configuration

Documentation can be configured for all panels by selecting the documentation tab as shown in **Figure 16.**

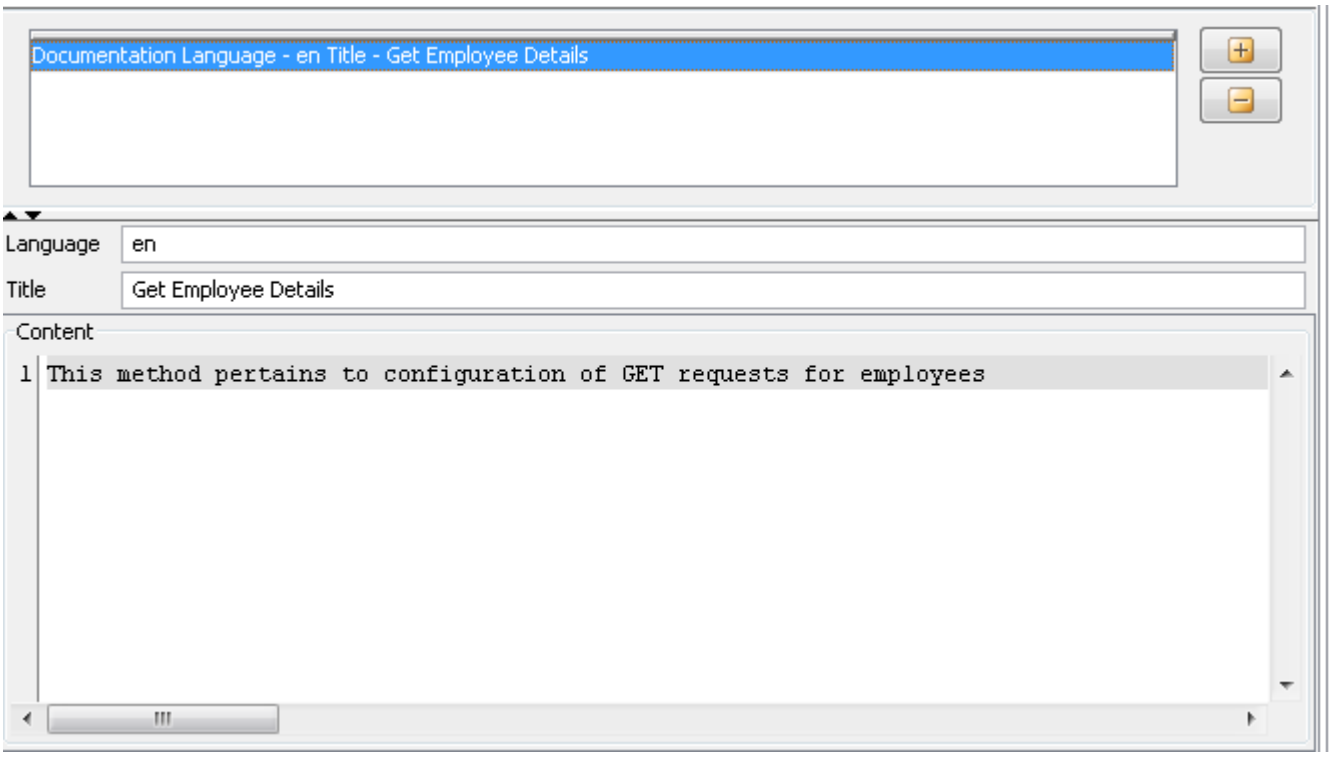


Figure 16: Configuring documentation.

Language: The attribute defines the language (xml:lang) for the title attribute value and the contents of the doc element. If an element contains more than one doc element then they MUST have distinct values for their xml:lang attribute.

Title: A short plain text description of the element being documented, the value SHOULD be suitable for use as a title for the contained documentation.

Content: The content of the documentation has to be specified here.

4.13.5.1.1.2.1.10 Error Notifications

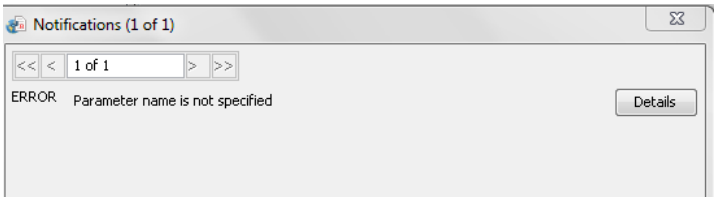


Figure 17: Error Notifications

The validation for a particular panel is done when the user tries to navigate away from the tree node corresponding to the panel. Validation errors corresponding to the particular panel are shown using the error notifying dialog as shown in the Figure 17.

4.13.5.1.1.2.2 WADL

This is a read only view of the WADL that is generated from the configuration done using configuration view.

Figure 18: WADL view

4.13.5.1.2 Transport Security

4.13.5.1.2.1 SSL Configuration

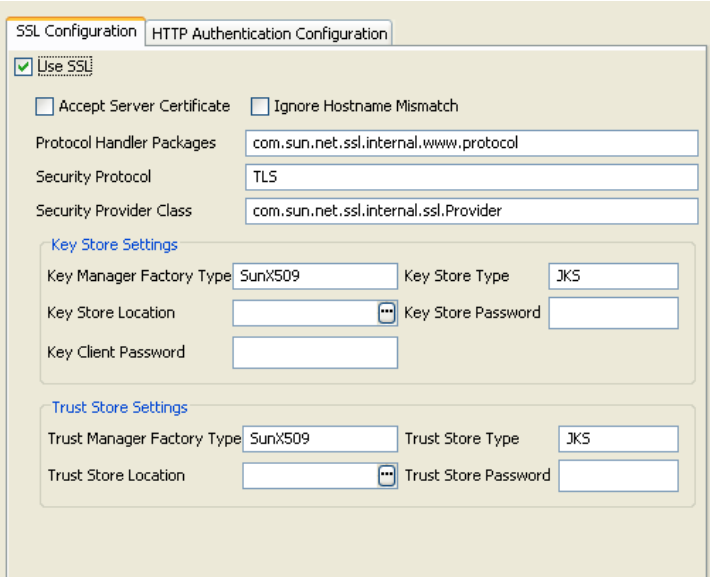


Figure 19: SSL Configuration

Use SSL: Select this option to enable SSL Settings. Rest of the properties in this editor are enabled and configurable only when this property is checked.

Accept Server Certificate: When accessing https URLs, this property determines whether the server certificates should be accepted or not. If selected, the certificate will be accepted without any validation otherwise it will result in an error.

Ignore Hostname Mismatch: If this option is selected the certificate will be accepted even if hostname in the certificate does not match with the hostname in the request URL. If its not selected exception is thrown.

Protocol Handler Packages: Determines protocol handler packages property.

Security Protocol: Determines Security protocol

Security Provider Class: Determines Security provider class.

Key Manager Factory Type: Algorithm for the Key Manager Factory.

Key Store Type: Type of the Key Store whose location is specified by **Key Store Location** should be specified in the field.

Key Store Location: Location of the key store file can be provided using the file dialog that opens up on clicking the ellipsis ⊡. The KeyStore is used by the component for client authentication.

Key Store Password: Password of the specified key store can be specified in the field.

Key Store Client Key: Determines Key Store Client Key

Trust Manager Factory Type: Algorithm for the Trust Manager Factory.

Trust Store Location: Location of the trust store file should be specified. TrustStore is a file where digital certificates of trusted sites are stored and retrieved for authentication during an SSL connection. TrustStore is used to authenticate a server in SSL authentication.

Trust Store Type: Type of Trust Store whose location is specified by property **Trust Store Location.**

Trust Store Password: Password of the specified trust store should be specified in the field.

4.13.5.1.2.2 HTTP Authentication

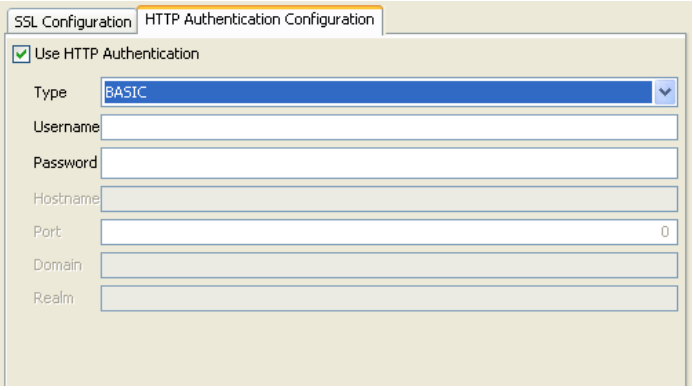


Figure 20: HTTP Authentication

Use HTTP Authentication: HTTP authentication support for the Web Services can be provided by selecting this property.

Type: The type of HTTP Authentication can be selected for the drop down. At present only **BASIC** authentication is supported.

Username: Username to be used by the RESTStub component while deploying the Web Service. The entry for this user name has to be present in Realm.properties file. If this user name is not present, the component will not launch.

Password: Password to be used by the RESTStub component while deploying the Web Service. If this password is not present in Realm.properties file the component will not launch.

Using Basic Authentication: Before enabling this property in RESTStub, basic authentication has to be enabled in FPS Jetty Server and in bcwsgateway web application (FIORANO_HOME/esb/server/jetty/fps/webapps/restgateway). Procedure to enable basic authentication in Jetty Server and in the web app is explained below.

1. Enabling Basic Authentication in Jetty Server
2. Stop the FPS Server if it is running.
3. Open FPS profile in Fiorano Studio and navigate to FPS→Fiorano→Esb→Jetty.
4. Select the Jetty mbean. In the properties window, set the property Basic Authentication to yes and give the fully qualified path of Realm.properties file against the property Realm Properties. More information on Realms and Realm.properties file content is discussed in the next section.

5. Save the profile.

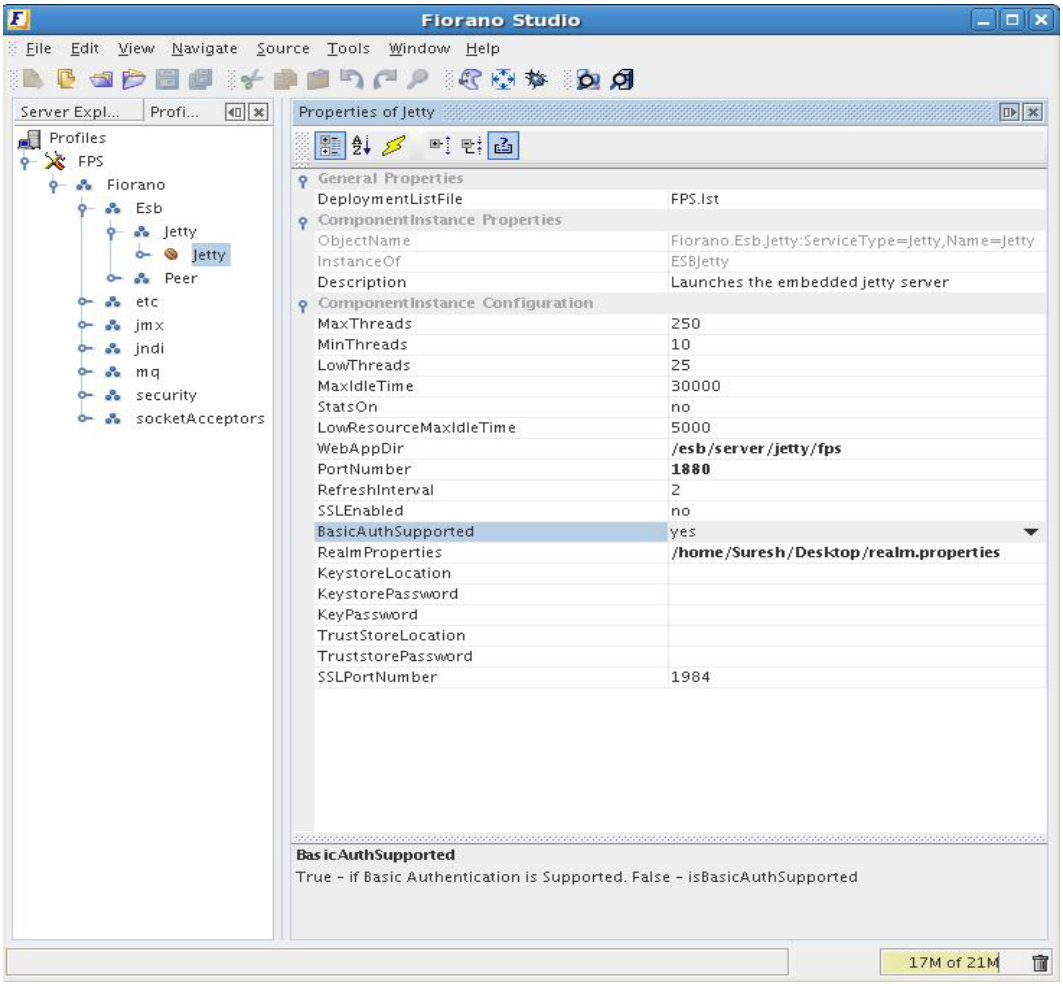


Figure 21: Enabling Basic Authentication in FPS Jetty server

Enabling Basic Authentication in webapp

1. Open Web.xml file located at FIORANO_HOME/esb/server/jetty/fps/webapps/restgateway/WEB-INF.
2. Uncomment the security-constraint and login-config elements and save the file.
3. Delete the Peer Server cached profile from FES runtime by deleting the folder FIORANO_HOME/runtimedata/EnterpriseServers/<Profile Name>/FES/peers/<peer profile>.
4. Start the FPS server.

The basic authentication is enabled in FPS Jetty Server and in the webapp. Now the authentication can be enabled in RESTStub by setting the value of Enable authentication property to yes. Provide a set of username and password which is present in Realm.properties file.

Note: To confirm the authentication, when the RESTStub component is launched right-click the component and select View WADL option. Web browser prompts the user to enter the username and password to display the WADL.

When the authentication is enabled in RESTStub, client can access the Web Service only if it uses the basic authentication with correct credentials.

UserName: Username to be used by the RESTStub component while deploying the Web Service. The entry for this user name has to be present in Realm.properties file. If this user name is not present, the component will not launch.

Password: Password to be used by the RESTStub component while deploying the Web Service. If this password is not present in Realm.properties file the component will not launch.

Realms: Security realms allow securing the web applications against unauthorized access. Protection is based on authentication (identifying who is requesting access to the webapp) and access control (restricting what can be accessed and how it is accessed within the webapp).

A webapp statically declares its security requirements in the web.xml file. Authentication is controlled by the <login-config> element. Access controls are specified by <security-constraint> and <security-role-ref> elements. When a request is received, the web container checks if the user performing the request is authenticated and if the user has a role assignment that permits access to the requested resource. Access to the resource is allowed only when the user is authorized and has the required permissions.

A realm has a unique name, and is composed of a set of users. Each user has authentication information (Example: a password) and a set of roles associated. More than one realm can be configured depending on the needs.

Fiorano uses a simple realm whose authentication and authorization information is stored in a properties file. Each line in the file contains a username, a password, and 0 (zero) or more role assignments. The format is:

```
username: password[,rolename ...]
```

where:

- username is the user's unique identity
- password is the user's password
- rolename is the user's role

Sample content of Realm.properties file is shown below:

- admin: admin,admin
- user: password,user
- guest: guest,read-only

4.13.5.1.3 Miscellaneous Configuration

4.13.5.1.3.1 FES Connection Configuration

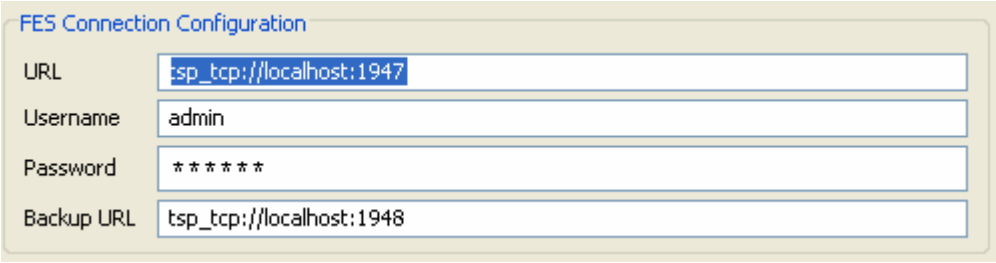


Figure 22: FES Connection Configuration

FES URL: The URL of Enterprise Server to which the Peer Server on which the component is running is connected.

Username: Username that should be used to connect to the enterprise server.

Password: Password that should be used to connect to the enterprise server.

FES Backup URL: The alternate URL that should be tried for connecting to the Enterprise Server if the Enterprise Server cannot be connected to using the URL mentioned against property **FES URL**.

> **Note:** In case of Enterprise Servers in HA mode, this should point to Secondary Server URL if the primary is set against Server URL property and vice-versa.

4.13.5.1.3.2 Execution Configuration

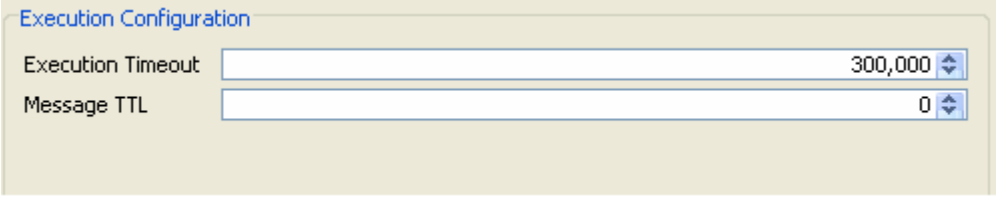


Figure 23: Execution Configuration

Execution Timeout: RESTStub component receives client request on its output port and sends the request message to the connected components. After the processing is completed by the connected components response message is sent to RESTStub input port and from there it will be sent to the client.

RESTStub component accepts the response if it reaches before the timeout period. If no response is received, on timeout Error message will be sent to the client saying No message received till the timeout period.

> **Note**: Request will be processed by the connected components in the Event Process even after the timeout but the response is not sent back to the client by RESTStub.

Execution Timeout of 0 indicates infinite timeout.

Request Timeout: This value is set as Time to live property on the request message. The request message will be discarded after the timeout.

0 indicates infinite time.

4.13.5.2 Port Generation

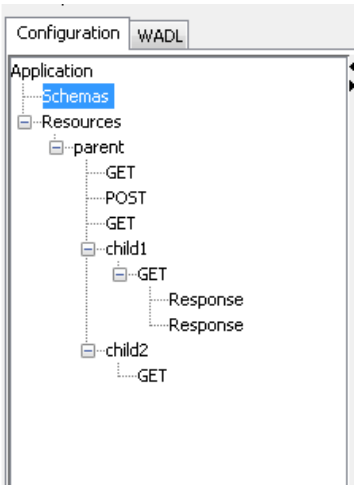


Figure 24: Sample Configuration

A request port and response port are generated for each method configured in the component with suffixes __Request and __Response respectively. The prefix of the port name will be the complete ID of the resource computed (as described in the section 4.13.5.1.1.2.1.3 Resource) followed by the method name.

If a resource has multiple methods with same name (two GET methods) then the method ID has to be specified for at least one of those methods. The method ID will be appended to the port name prefix to differentiate between these methods. For the sample configuration defined in **Figure 24,** the ports will be generated as shown in the **Figure 25.**

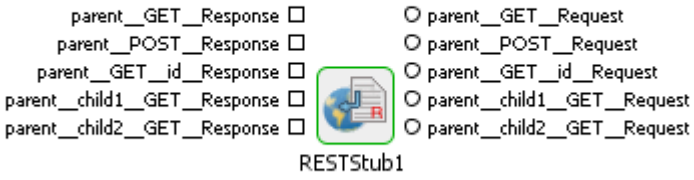


Figure 25: Port Generation

4.13.5.3 Schema Generation

The schemas are automatically generated for the ports based on the configuration

4.13.5.3.1 Request Schema

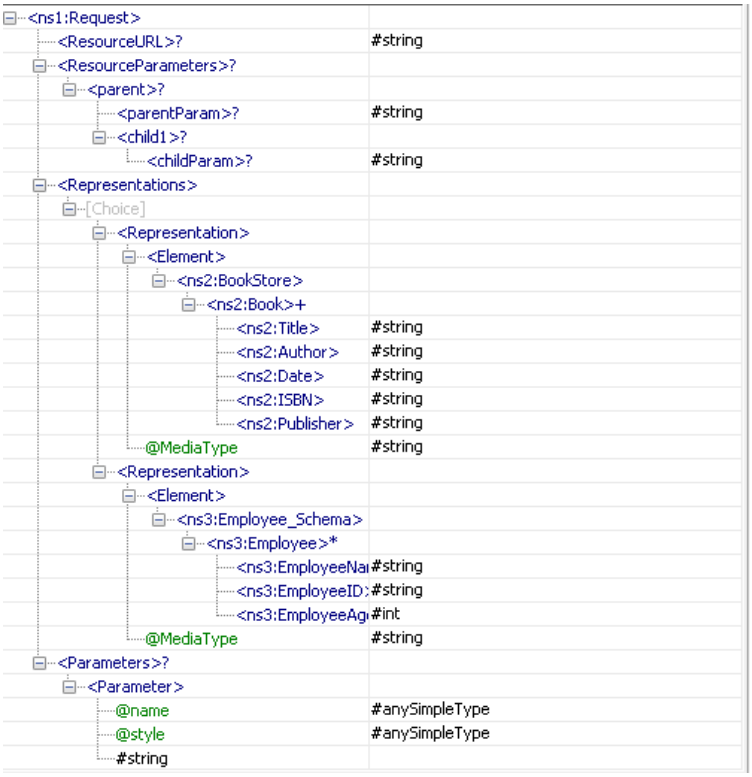
The schema of the output port corresponding to a method depends on its configuration and its parent resources.

The resource parameters are added as child element of the element **ResourceParameters**. The hierarchy of the resources is maintained as it is in the resource chain corresponding to the method. The parameters of a resource are added as its child elements.

The methods request parameters are added as child elements to the element **RequestParameters.** The various representations of the request are added as child elements of element **Representations.** The element specified is present as child element of **Element** and the representation's parameters are added as child elements of element **representationParameters**.

If there are any query parameters other than those specified in the configuration for the method, these parameters are added as part of **Parameters** element.

For the sample configuration shown in **Figure 24**, the schema generated for port **parent __POST_Request** is as in **Figure 26**



<ns1:Request>	
<ResourceParameters>	
<parent>?	
<parentParam>?	#string
<child1>?	
<childParam>?	#string
<RequestParameters>?	
<requestParam>?	#string
<Representations>	
[Choice]	
<Representation>	
<Element>	
<ns2:BookStore>	
<ns2:Book>+	
<ns2:Title>	#string
<ns2:Author>	#string
<ns2:Date>	#string
<ns2:ISBN>	#string
<ns2:Publisher>	#string
@MediaType	#string
<Representation>	
<Element>	
<ns3:Employee_Schema	
<ns3:Employee>*	
<ns3:Employee	#string
<ns3:Employee	#string
<ns3:Employee	#int
@MediaType	#string

Figure 26: Schema of port parent __POST_Request

4.13.5.3.2 Response Schema

The schema of the input port corresponding to a method depends on its configuration. Schema is populated on the response port of a method only if it has a response as its child element.

For each response configured a **Response** element is added as child element of the root element. The status of the response is present as attribute of this element. The response parameters corresponding to a particular Response are added as child elements to the element **ResponseParameters.**

The various representations of the response are added as child elements of element **Representations.** The element specified is present as child element of **Element** and the representation's parameters are added as child elements of element **representationParameters**.

For the sample configuration shown in **Figure 24**, the schema generated for port **parent__child1__GET_Response** is as in **Figure 27**.

<ns1:Responses>	
[Choice]	
<Response>	
<ResponseParameters>?	
<responseParam1>?	#string
<Representations>	
[Choice]	
<Representation>	
<Element>	
<ns3:Employee_Schema>	
<ns3:Employee>*	
<ns3:EmployeeName>	#string
<ns3:EmployeeID>	#string
<ns3:EmployeeAge>	#int
@MediaType	#string
<Representation>	
<Element>	
<ns2:Book>	
<ns2:Title>	#string
<ns2:Author>	#string
<ns2:Date>	#string
<ns2:ISBN>	#string
<ns2:Publisher>	#string
@MediaType	#string
@Status	#int
<Response>	
<ResponseParameters>?	
<responseParam2>?	#string
<Representations>	
[Choice]	
<Representation>	
<Element>	
<ns2:Date>	#string
@MediaType	#string
@Status	#int

Figure 27: Schema of port parent__child1__GET_Response

4.13.5.4 Functional Demonstration

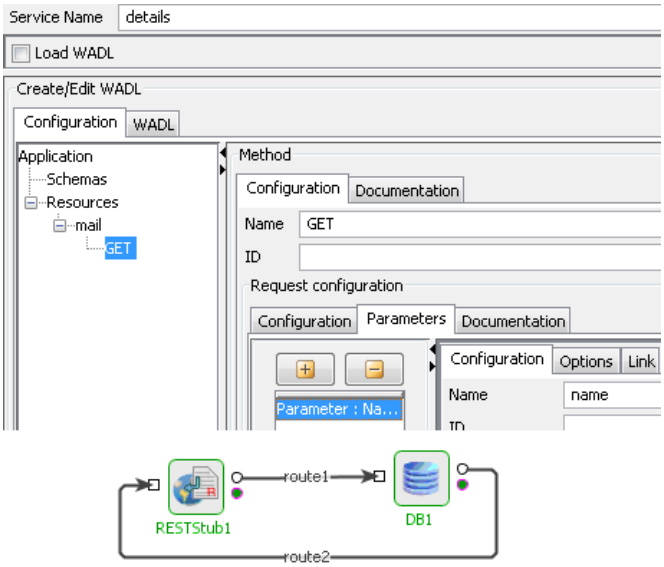


Figure 27: RESTStub being used to expose a database

Figure 27 demonstrates a scenario where the RESTStub component is used to expose a database query as a RESTful service. The service queries a database with a name and fetches the email id of the particular user from the table. The RESTStub component is configured with a single resource and request query parameter name is added to the GET method. This parameter is mapped to the name column of a database by using the route transformation.

No Response is added to the method so that the response string will be passed as is to the client. The route from DB to RESTStub (route 2) is configured to transform the DB response to just the mail ID. A sample invocation is as shown in figure 28.

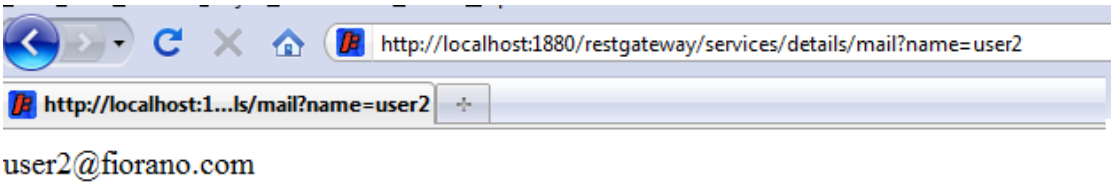


Figure 28: using service to lookup mail details of an user.

4.13.5.5 Useful Tips

- For SSL to work properly the JDK version used should be greater than 1.5.0_08 and the jars present in JDK_HOME/jre/lib/ext should be copied to $FIORANO_HOME/esb/lib/ext.
- When RESTStub component is configured to launch on HA (High Availability) Peer Servers
    - If both Primary and Secondary Servers are on the same machine

      Initially, if RESTStub is launched on the Primary Server, the generated WADL URL contains Primary Server's jetty port number. In case of Primary Server failover, the Secondary Server becomes Active and relaunches the component. WADL will be regenerated and if the Secondary Server uses a different jetty port then the WADL URL is changed. The clients have to be reconfigured to use new URL in this case.

      To avoid this situation, it is recommended to use same jetty ports for both Primary and Secondary Peer Servers.

      Jetty service will be started only after the server is started successfully. In case of HA, only one server will be active at a given time and the Jetty Server will be running only in the active server and there will be no bind exceptions even if both the servers use same port number for Jetty.
    - If both Primary and Secondary servers are on different machines

      In this case if the failover happens the hostname/IP address in the WADL URL will be changed. So the clients have to be reconfigured accordingly.
- The service name must be unique for RESTStub instances configured on same peer server.
- The component acts as a helper to expose an event process as a RESTful service. The developer of an event process will have to take care that the functionality is exposed as per the guidelines of REST architecture.

- When the content type of request is application/octet-stream the content is sent as bytes content in the message, this will be present as bytes property. This can be retrieved as Base 64 encoded string in mapper using the funclet Byte-Content in JMS Message funclets.

4.13.6 RESTConsumer

This component is used to expose an Event Process or a part of Event Process as a RESTful web service deployed in the Peer Server. A RESTful web service is a simple web service implemented using HTTP and the principles of REST. It is a collection of resources. WADL for the service deployed can be fetched when this component is running and the URL of the WADL can be used by clients to access the Web Service. Each instance of the component hosts a single web service. The component can be configured by:

- Manually defining the resources and methods that are supported by the service in the CPS of the component.

The WADL generated for the component will differ from the WADL provided in following aspects:

- If the provided WADL contains multiple 'resources' elements, only the first resources element will be loaded and used.

The component generates request and response port for each method present in the configured web service.

4.13.6.1. Configuration

The component can be configured using the custom property sheet as shown in the following sections.

4.13.6.1.1. WADL Definition

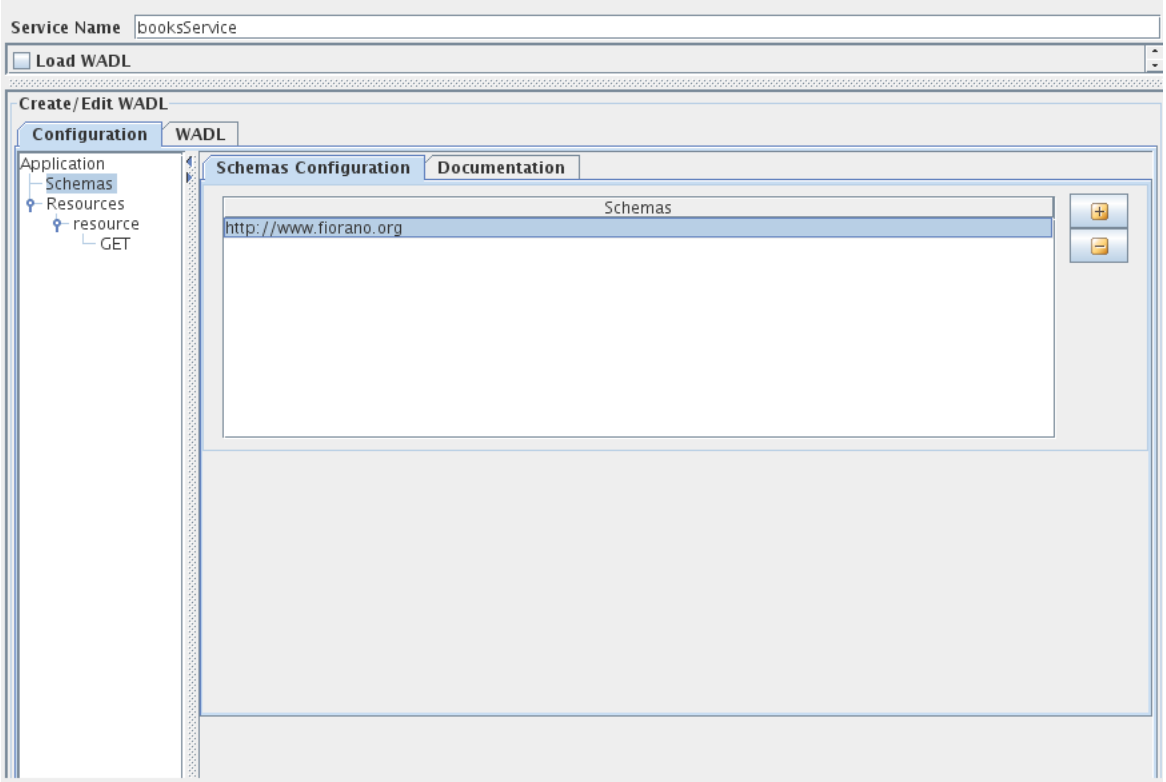


Figure 1: Configuration Property Sheet

This page is used to define the RESTFul service using a user interface for WADL. User can create a WADL by manually defining different aspects of the service.

4.13.6.1.1.2 WADL Definition Views

Details of the selected WADL are shown in 2 different views – Configuration and WADL. However, web service can be configured using Configuration view only.

4.13.6.1.1.2.1 Configuration View

The RESTful service can be configured by selecting the nodes in the tree on the left side. When a node is selected in the tree the panel corresponding to its configuration is shown in the right side pane. The details of different panels corresponding to the nodes are as explained in the following sections.

4.13.6.1.1.2.1.1 Schemas

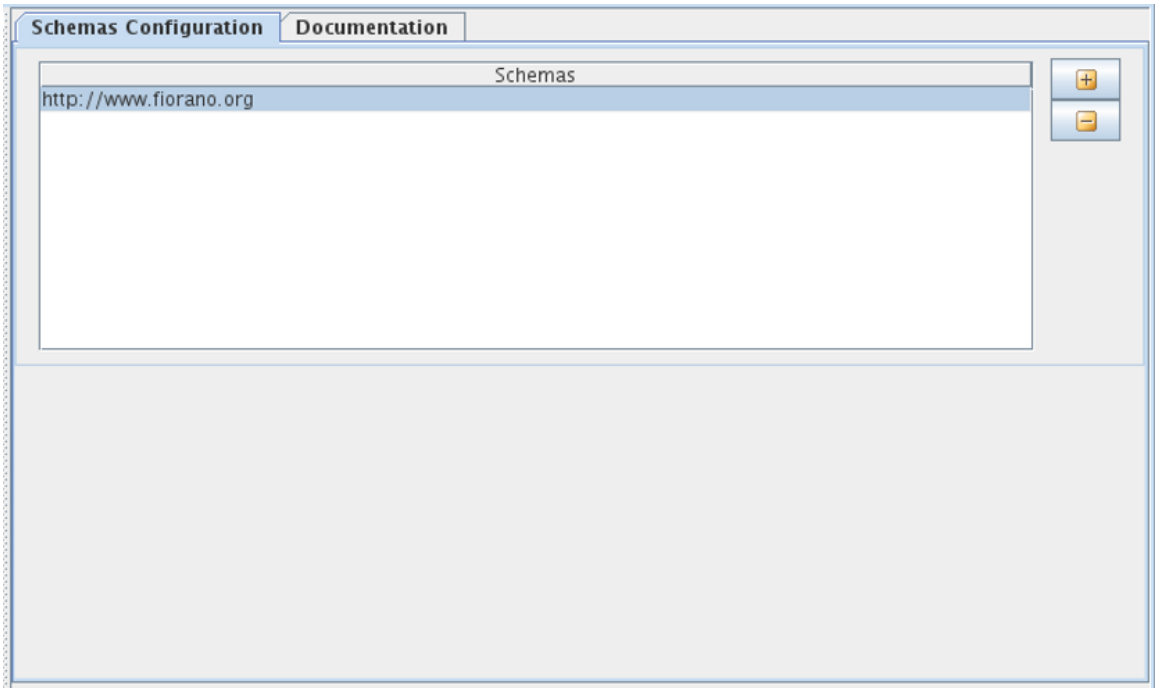


Figure 5: Schemas Configuration

The XML schemas which are required for the request and response messages have to be specified in this panel. A schema can be added by clicking the button in the table shown in the **figure 5**.

There are 3 ways to add schema to schema repository; Load from File, Load from Port Schemas and Load from Schema Repository. When Load from File is chosen, selected content in the schema file is shown in the area given. When Load from Port Schemas is selected, schema is directly added to the schema repository. When Load from Schema Repository is selected, schema can be fetched from schema repository. In this case, the required schemas have to be added to the schema repository using Studio.

Note: The elements dialog in representation will be populated from the schemas that are specified in this panel.

4.13.6.1.1.2.1.2 Resources

The resources node will act as parent of all the resources that are added as part of this service. It provides the base path for all these resources.

The base URI of the service that is being consumed has to be specified here. In general it will be of the form,

```
http(s)://<hostname:<port>/<static path segment>.
```

Note: If the base URI changes across invocations, it can be specified in the input message using the **ResourceURL** element.

4.13.6.1.1.2.1.3 Resource

A resource logically represents a entity whose operations have to be exposed using the RESTful service. It may optionally be used to specify just a path segment for a child URL with no methods defined on the resource itself.

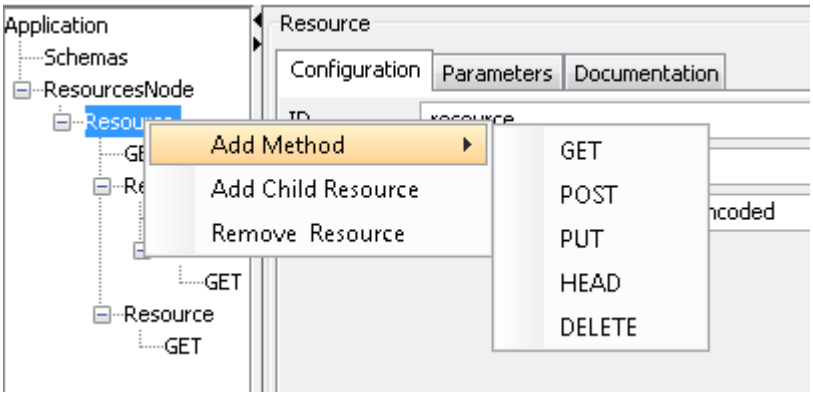


Figure 6: Adding child resources and methods

A resource can be added to a resources/resource element by selecting the option Add Resource/Add Child Resource respectively as shown in **Figure 6.**

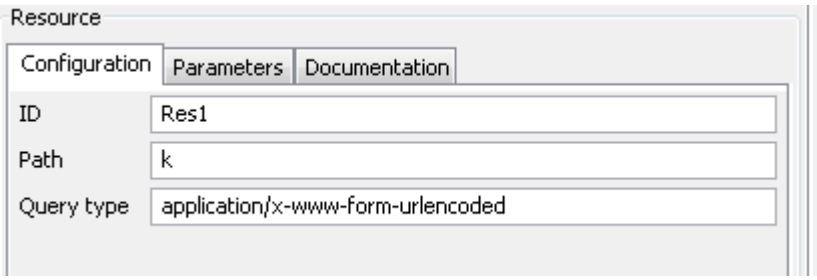


Figure 7: Resource configuration

The configuration of a resource contains fields as shown in **Figure 7** as explained below.

ID

The identifier for the resource can be specified using this element. This identifier is used in the generation of port names and schemas corresponding to the methods that are defined on this resource or any of its child resources.

The ID has to be unique among its siblings. The complete ID of a resource will be computed using information of parent IDs as below.

Example:

If the resource with ID "parent" is added to Resources element (using Add Resource option) and a child is added to the resource (using "Add Child Resource" option) with ID "child", the complete ID of the resource will be **parent__child**.

Path

The path of the resource, relative to its parent resource/resources element has to be specified against this field.

The path of a resource has to be unique among the siblings of a resource. The path of a resource will be computed including information of parent paths as below.

Example:

If the resource with path "parent" is added to Resources element (using Add Resource option) and a child is added to the resource (using "Add Child Resource" option) with path "child", the actual path of the resource will be <Resources Base Path>/parent/child.

Templates are allowed in the path element. A template will have the form {<template name>}. When a template element is used a parameter with same name and style as Template have to be defined. The request message on the port corresponding to this resource must have the value of the template parameter as the value that is being requested.

Example:

If a request has to be sent to URL <Parent Path>/<template> where the value of template name is dynamic then

- The resource(id resourceID) path must be specified as {<templateName>}.
- A parameter templateName must be specified with style "Template" for this resource.
- The actual value of the template must be passed in the request as shown below.

```
.. <resourceID>
        <templateName>178</templateName>
    </resourceID>..
```

Please refer to generation of schemas for more information on parameter.

QueryType

The property defines the media type for the query component of the resource URI. The value defaults to 'application/x-ww-form-urlencoded' if not specified which results in query strings being formatted as specified in section 17.13 of HTML 4.01.

4.13.6.1.1.2.1.3 Method

Methods that resource supports can be added by using the context menu of a resource as shown in **Figure 6**.

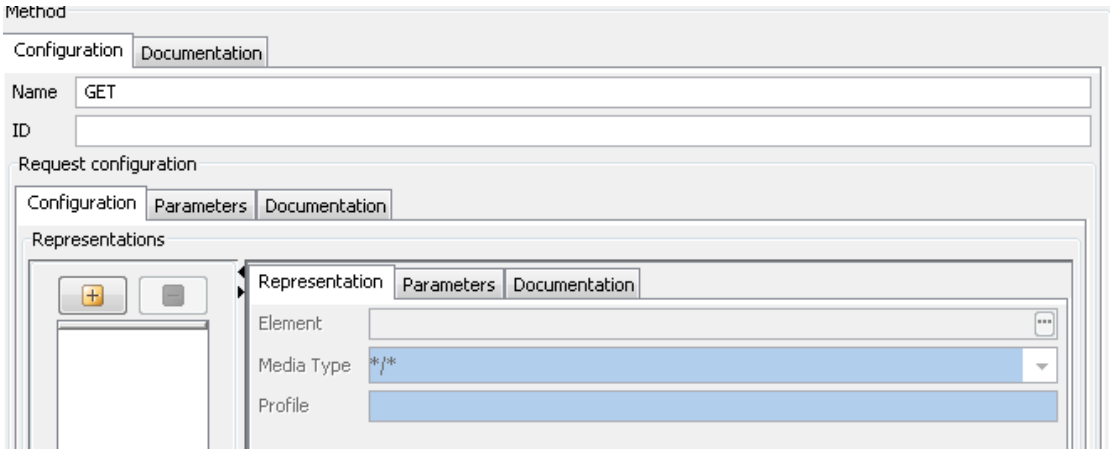


Figure 8: Method Configuration

The various attributes of a method are as shown in **Figure 8**. The request configuration of a method can also be configured in this panel.

Name

The attribute hold the name of the method that the resource supports. The attribute is non editable. The name will be used in the generation of ports and schemas for the component.

Two methods on a resource can have the same method name. In such case ID has to be specified for the method to differentiate between the methods.

ID

The method ID acts as a differentiator between methods of a same resource with same name. This is required only if the resource defines more than one method with the same name.

4.13.6.1.1.2.1.4 Request configuration

The request for a message can be configured in the panel corresponding to the method as shown in **Figure 8.** As part of request one can define multiple representations. Different representations can have different media types and elements defined. The media Type requested will be sent as part of the request message. It's the responsibility of the event process developer to return the appropriate content type based on the request.

Configuring representations is explained in detail in section 4.13.6.1.1.2.1.6

Similarly the request parameters can be configured for a particular method by using the parameters tab. Configuring parameters is explained in detail in section 4.13.6.1.1.2.1.7

4.13.6.1.1.2.1.5 Response configuration

A method can have multiple responses. Different responses cater to different sets of status codes. A response element can be added by using the context menu on a method as shown in **Figure 9.**

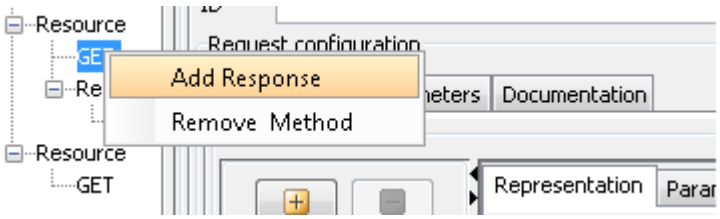


Figure 9: Adding Response

The response can be configured using the panel as shown in **Figure 10.**

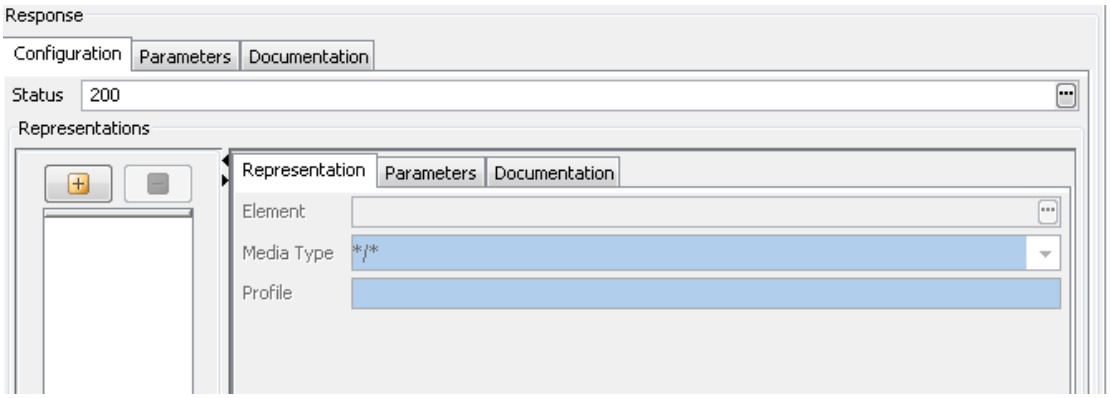


Figure 10: Response configuration

Status

The status of response can be space separate list of the standard HTTP status codes or any custom codes defined in the application. The status codes can be selected among standard HTTP status codes by clicking on ellipsis against this property as shown in **Figure 11**

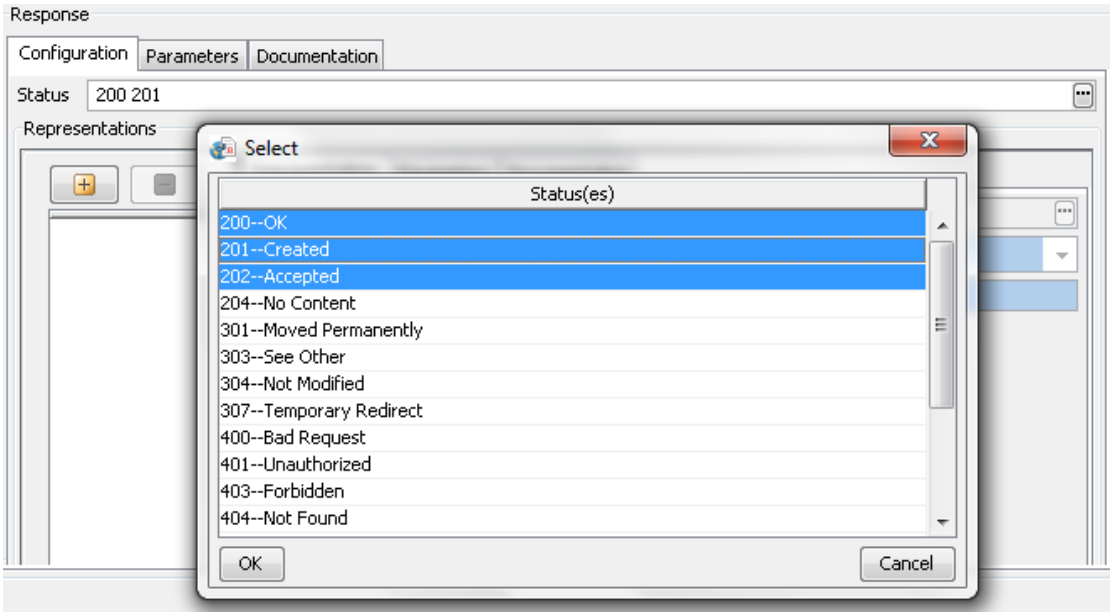


Figure 11: Configuring status

Representations

Similar to a request, each response (corresponding to a set of statuses) can have multiple representations. The representations provide different views of same response as defined by the media type.

4.13.6.1.1.2.1.6 Representation configuration

Multiple representations can be defined for both request and response of a method as shown in **Figure 12.**

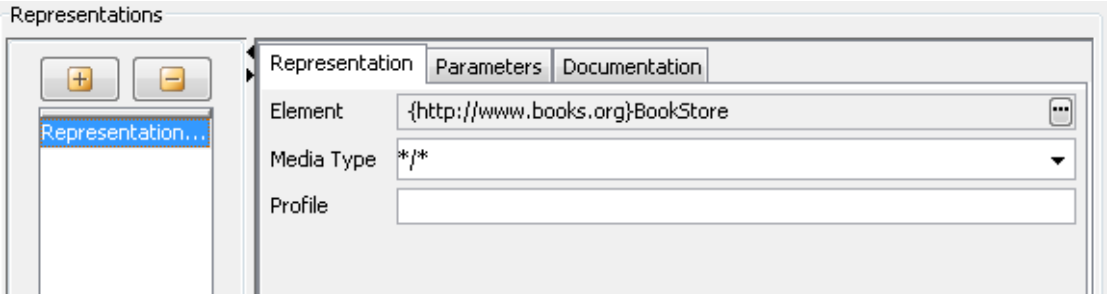


Figure 12: Representation Configuration

Element

The element can be used to specify the xml schema of an element corresponding to the representation. The schema corresponding to the element has to be added to the schema repository using studio and an include has to be added using the Schemas panel as explained in section 1.1.2.1.1 Schemas.

The element can then be selected by clicking on the ellipsis and selecting element as shown in the **Figure 13: Configuring elements.**

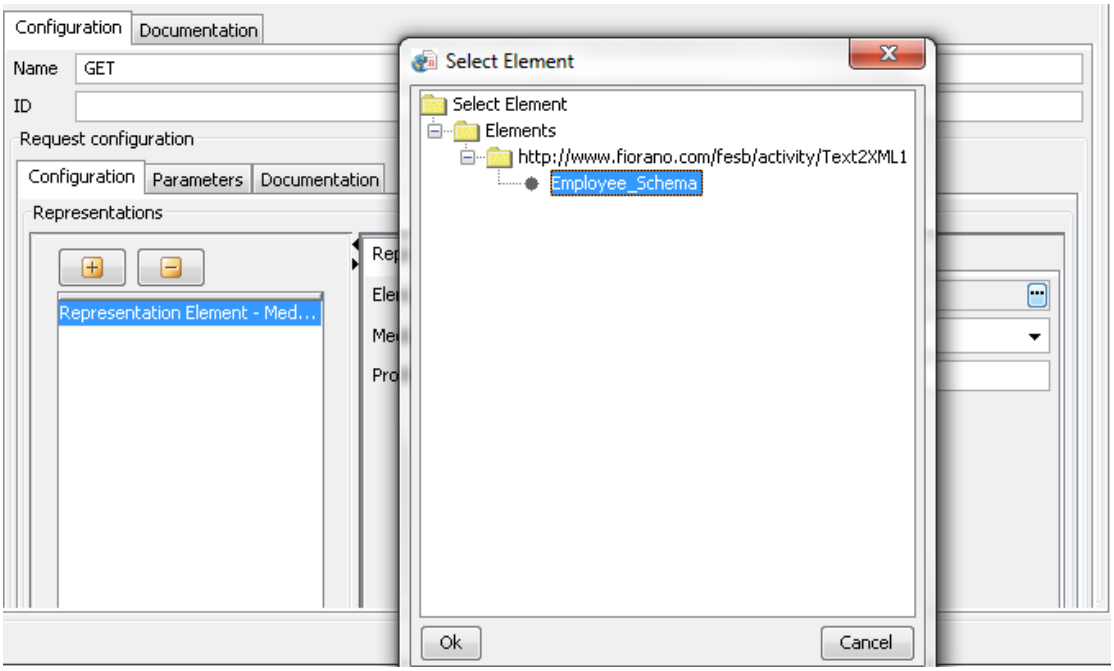


Figure 13: Configuring elements

MediaType

The media type of a representation defines the type of the content that is present in the representation. The media type can be configured by selecting the appropriate one from the drop down or manually specifying in the combo as shown in **Figure 14: Media Type configuration**

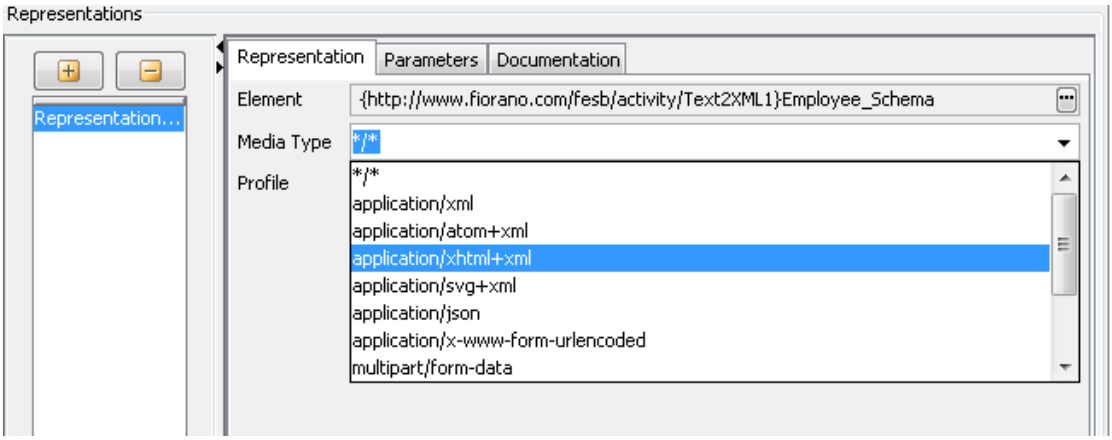


Figure 14: Media Type configuration

Profile

The attribute is similar to the HTML profile attribute, gives the location of one or more meta data profiles, separated by white space. The meta-data profiles define the meaning of the rel and rev attributes of descendent link elements

4.13.6.1.1.2.1.7 Parameter configuration

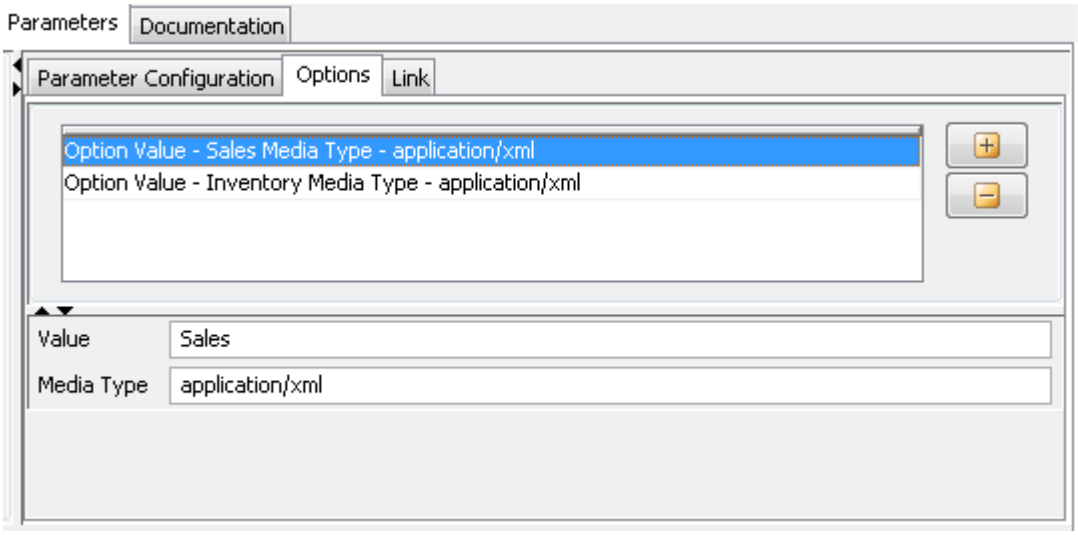


Figure 15: Parameter Configuration

Parameters are present at different levels in a resource tree. These can be configured using the UI as shown in **Figure 15**.

The different attributes of a parameter are as follows

Name: The name of the parameter as an xsd:NMTOKEN. Required.

Style: Indicates the parameter style, only the allowed values are populated in the combo box of a certain configuration.

Type: Optionally indicates the type of the parameter as an XML qualified name, defaults to xsd:string.

Default: Optionally provides a value that is considered identical to an unspecified parameter value.

Path: When the parent element is a representation element, this attribute optionally provides a path to the value of the parameter within the representation. For XML representations, use of XPath 1.0 is recommended.

Required: Optionally indicates whether the parameter is required to be present or not, defaults to false (parameter not required).

Repeating: Optionally indicates whether the parameter is single valued or may have multiple values, defaults to false (parameter is single valued).

Fixed: Optionally provides a fixed value for the parameter.

Options: An option element defines one of a set of possible values for the parameter represented by its parent parameter. An option has the following attributes:

Value: A required attribute that defines one of the possible values of the parent parameter.

mediaType: When present this indicates that the parent parameter acts as a media type selector for responses. The value of the attribute is the media type that is expected when the parameter has the value given in the value attribute.

Link: A link element is used to identify links to resources within representations. A link element is a child of a parameter whose path attribute identifies the portion of its parent representation that contains a link URI.

4.13.6.1.1.2.1.8 Documentation Configuration

Documentation can be configured for all panels by selecting the documentation tab as shown in **Figure 16.**

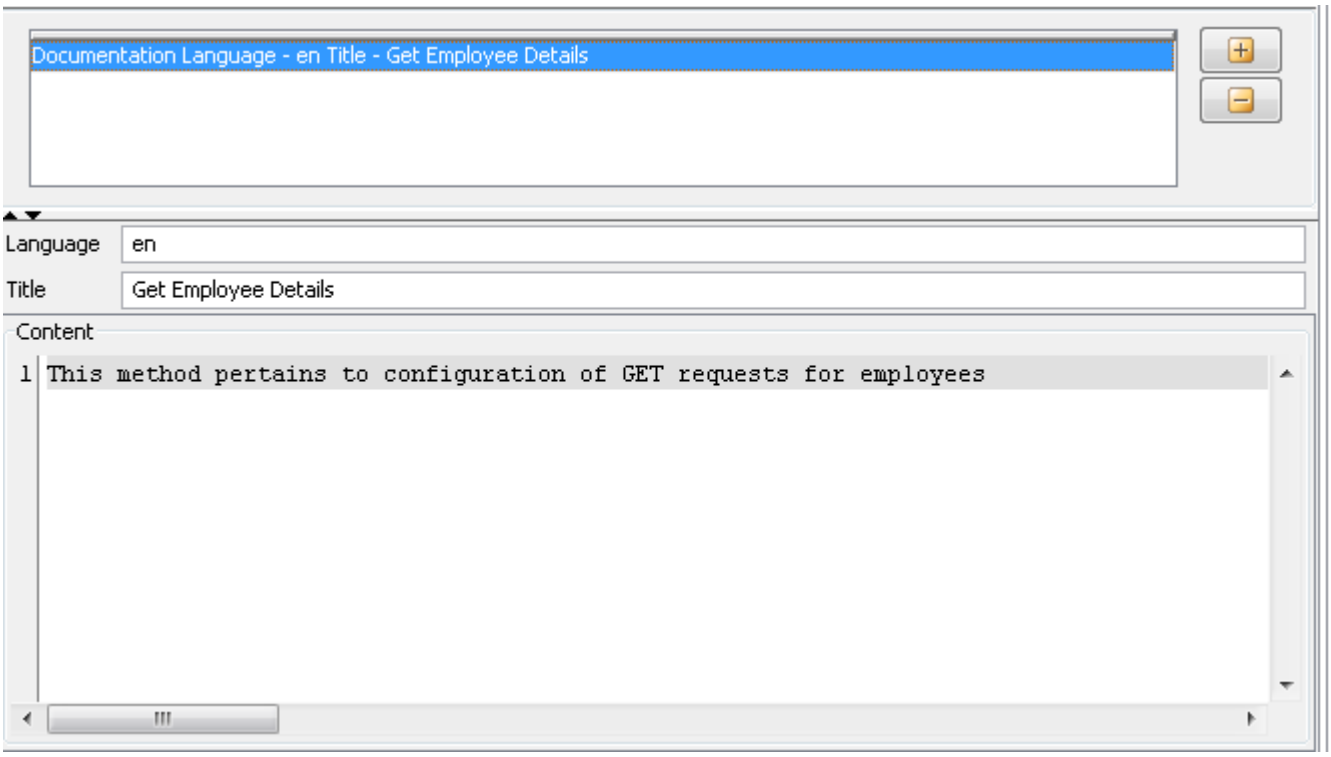


Figure 16: Configuring documentation

Language

The attribute defines the language (xml:lang) for the title attribute value and the contents of the doc element. If an element contains more than one doc element then they MUST have distinct values for their xml:lang attribute.

Title

A short plain text description of the element being documented, the value SHOULD be suitable for use as a title for the contained documentation.

Content

The content of the documentation has to be specified here.

4.13.6.1.1.2.1.9 Error Notifications

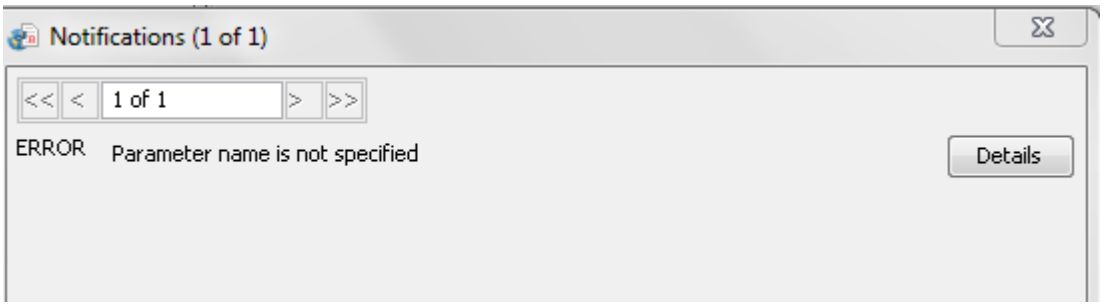


Figure 17: Error Notifications

The validation for a particular panel is done when the user tries to navigate away from the tree node corresponding to the panel. Validation errors corresponding to the particular panel are shown using the error notifying dialog as shown in the **Figure 17.**

4.13.6.1.1.2.2. WADL

This is a read only view of the WADL that is generated from the configuration done using configuration view.

Figure 18: WADL view

4.13.6.1.2 Transport Security

4.13.6.1.2.1 SSL Configuration

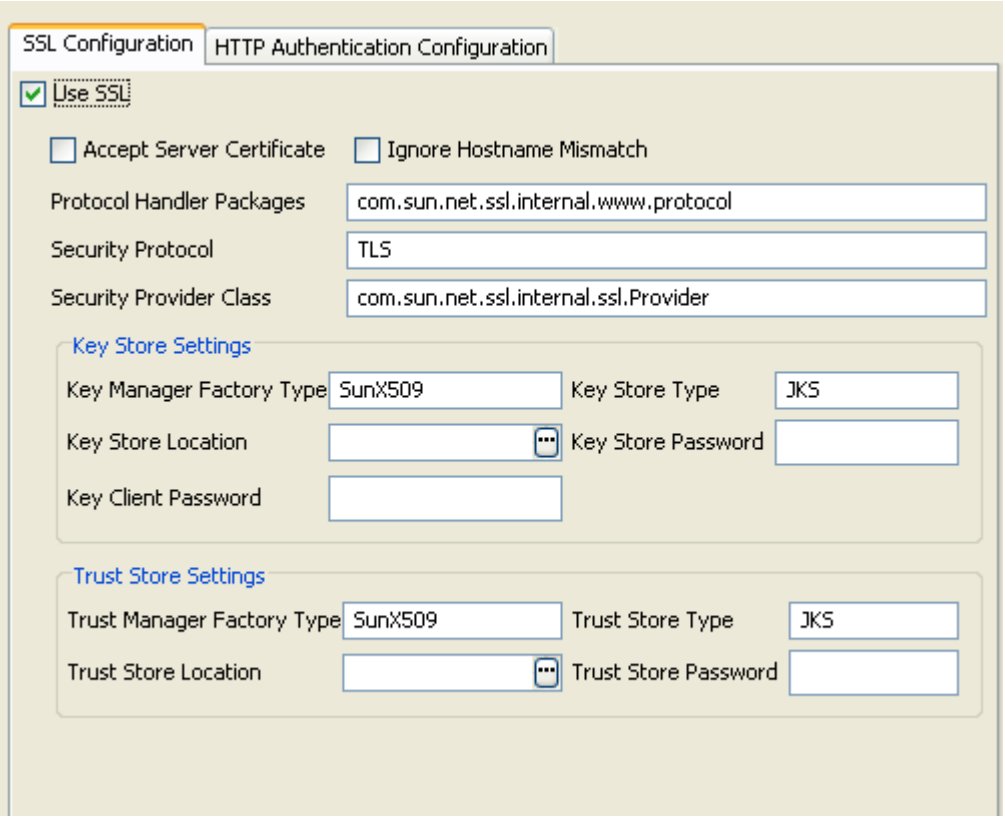


Figure 19: SSL Configuration

To consume a RESTful service using SSL the component must be configured to use SSL Configuration.

Use SSL

Select this option to enable SSL Settings. Rest of the properties in this editor are enabled and configurable only when this property is checked.

Accept Server Certificate

When accessing https URLs, this property determines whether the server certificates should be accepted or not. If selected, the certificate will be accepted without any validation, otherwise it will result in an error.

Ignore Hostname Mismatch

If this option is selected the certificate will be accepted even if hostname in the certificate does not match with the hostname in the request URL. If its not selected exception is thrown.

Protocol Handler Packages

Determines protocol handler packages property.

Security Protocol

Determines Security protocol

Security Provider Class

Determines Security provider class.

Key Manager Factory Type

Algorithm for the Key Manager Factory.

Key Store Type

Type of the Key Store whose location is specified by **Key Store Location** should be specified in the fileld.

Key Store Location

Location of the key store file can be provided using the file dialog that opens up on clicking the ellipsis . The KeyStore is used by the component for client authentication.

Key Store Password

Password of the specified key store can be specified in the field.

Key Store Client Key

Determines Key Store Client Key

Trust Manager Factory Type

Algorithm for the Trust Manager Factory.

Trust Store Location

Location of the trust store file should be specified. TrustStore is a file where digital certificates of trusted sites are stored and retrieved for authentication during an SSL connection. TrustStore is used to authenticate a server in SSL authentication.

Trust Store Type

Type of Trust Store whose location is specified by property **Trust Store Location.**

Trust Store Password

Password of the specified trust store should be specified in the field.

4.13.6.1.2.2 HTTP Authentication

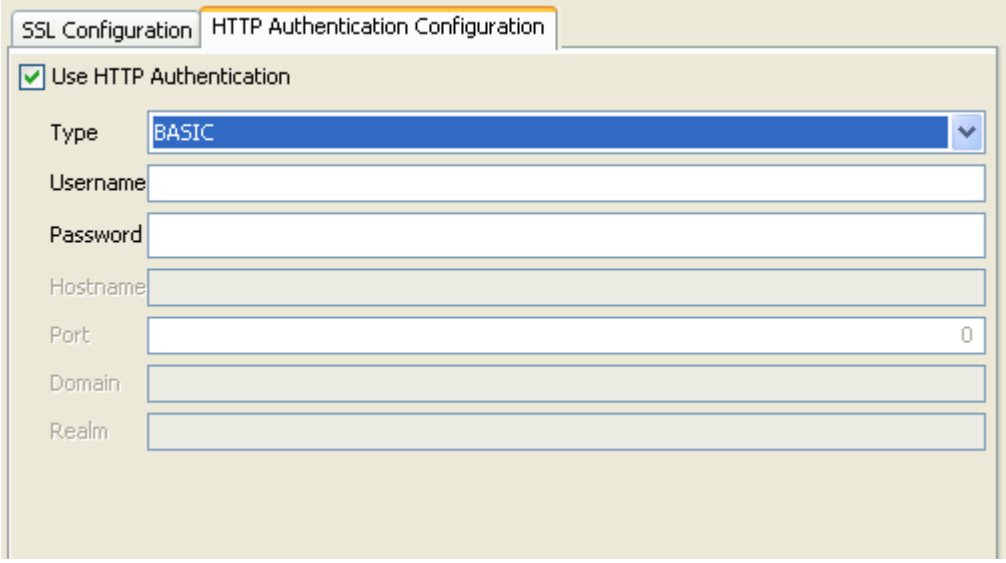


Figure 20: HTTP Authentication

Use HTTP Authentication

To connect to RESTful service where the basic authentication is present, the HTTP authentication has to be configured in the component.

Type

The type of HTTP Authentication can be selected for the drop down. At present only **BASIC** authentication is supported.

Username

Username to be used to connect to the Web Service has to be specified here.

Password

Password for the username specified can be specified using this field.

4.13.6.1.3 Additional Configuration

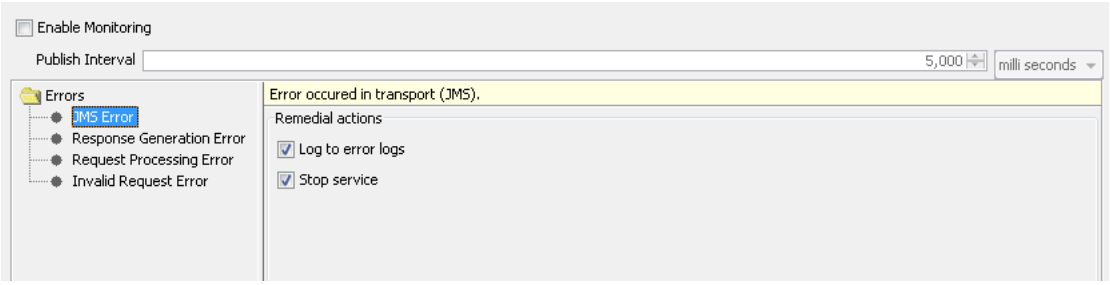


Figure 21: Additional Configuration

Additional configuration for the component includes monitoring and error handling configuration. Details of these configurations can be referred in common configurations.

4.13.6.2 Port Generation

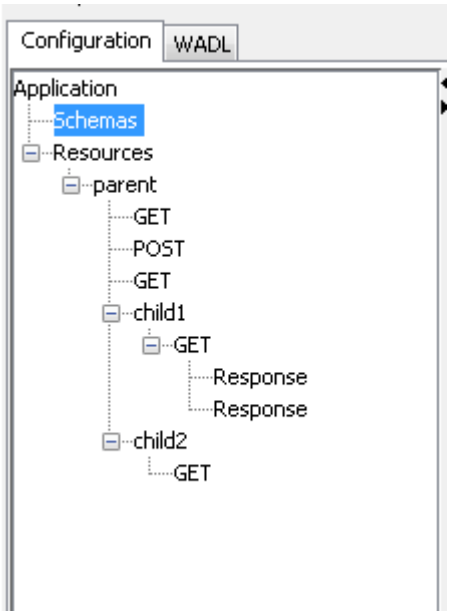


Figure 23: Sample Configuration

A request port and response port are generated for each method configured in the component with suffixes __Request and __Response respectively. The prefix of the port name will be the complete ID of the resource computed (as described in the section 4.13.61.1.2.1.3 Resource) followed by the method name.

If a resource has multiple methods with same name (two GET methods) then the method ID has to be specified for at least one of those methods. The method ID will be appended to the port name prefix to differentiate between these methods. For the sample configuration defined in **Figure 23,** the ports will be generated as shown in the **Figure 24.**

Figure 24: Port Generation

4.13.6.3 Schema Generation

The schemas are automatically generated for the ports based on the configuration

4.13.6.3.1 Request Schema

The schema of the input port corresponding to a method depends on its configuration and its parent resources.

The optional element **ResourceURL** can be used to dynamically invoke a RESTful service based on the input message. The value specified will be given preference over the base URL specified in the CPS.

The resource parameters configured are added as child element of the element **ResourceParameters**. The hierarchy of the resources is maintained as it is in the resource chain corresponding to the method. The parameters of a resource are added as its child elements.

The methods request parameters are added as child elements to the element **RequestParameters.** The various representations of the request are added as child elements of element **Representations.** The element specified is present as child element of **Element** and the representation's parameters are added as child elements of element **representationParameters**.

The optional **Parameters** element can be used to send any parameters that are not present as part of the configuration.

For the sample configuration shown in **Figure 24**, the schema generated for port **parent__POST_Request** is as in **Figure 26**

<ns1:Request>	
<ResourceURL>?	#string
<ResourceParameters>?	
<parent>?	
<parentParam>?	#string
<child1>?	
<childParam>?	#string
<Representations>	
[Choice]	
<Representation>	
<Element>	
<ns2:BookStore>	
<ns2:Book>+	
<ns2:Title>	#string
<ns2:Author>	#string
<ns2:Date>	#string
<ns2:ISBN>	#string
<ns2:Publisher>	#string
@MediaType	#string
<Representation>	
<Element>	
<ns3:Employee_Schema>	
<ns3:Employee>*	
<ns3:EmployeeNa	#string
<ns3:EmployeeID	#string
<ns3:EmployeeAg	#int
@MediaType	#string
<Parameters>?	
<Parameter>	
@name	#anySimpleType
@style	#anySimpleType
#string	

<ns1:Request>	
<ResourceParameters>	
<parent>?	
<parentParam>?	#string
<child1>?	
<childParam>?	#string
<RequestParameters>?	
<requestParam>?	#string
<Representations>	
[Choice]	
<Representation>	
<Element>	
<ns2:BookStore>	
<ns2:Book>+	
<ns2:Title>	#string
<ns2:Author>	#string
<ns2:Date>	#string
<ns2:ISBN>	#string
<ns2:Publisher>	#string
@MediaType	#string
<Representation>	
<Element>	
<ns3:Employee_Schema	
<ns3:Employee>*	
<ns3:Employee	#string
<ns3:Employee	#string
<ns3:Employee	#int
@MediaType	#string

Figure 26: Schema of port parent__POST_Request

4.13.6.3.2 Response Schema

The schema of the output port corresponding to a method depends on its configuration. Schema is populated on the response port of a method only if it has a response as its child element.

For each response configured a **Response** element is added as child element of the root element. The status of the response is present as attribute of this element. The response parameters corresponding to a particular Response are added as child elements to the element **ResponseParameters.**

The various representations of the response are added as child elements of element **Representations.** The element specified is present as child element of **Element** and the representation's parameters are added as child elements of element **representationParameters**.

For the sample configuration shown in **Figure 24**, the schema generated for port **parent__child1__GET_Response** is as in **Figure 27**.

<ns1:Responses>	
[Choice]	
<Response>	
<ResponseParameters>?	
<responseParam1>?	#string
<Representations>	
[Choice]	
<Representation>	
<Element>	
<ns3:Employee_Schema>	
<ns3:Employee>*	
<ns3:EmployeeName>	#string
<ns3:EmployeeID>	#string
<ns3:EmployeeAge>	#int
@MediaType	#string
<Representation>	
<Element>	
<ns2:Book>	
<ns2:Title>	#string
<ns2:Author>	#string
<ns2:Date>	#string
<ns2:ISBN>	#string
<ns2:Publisher>	#string
@MediaType	#string
@Status	#int
<Response>	
<ResponseParameters>?	
<responseParam2>?	#string
<Representations>	
[Choice]	
<Representation>	
<Element>	
<ns2:Date>	#string
@MediaType	#string
@Status	#int

Figure 27: Schema of port parent__child1__GET_Response

4.13.6.4 Functional Demonstration

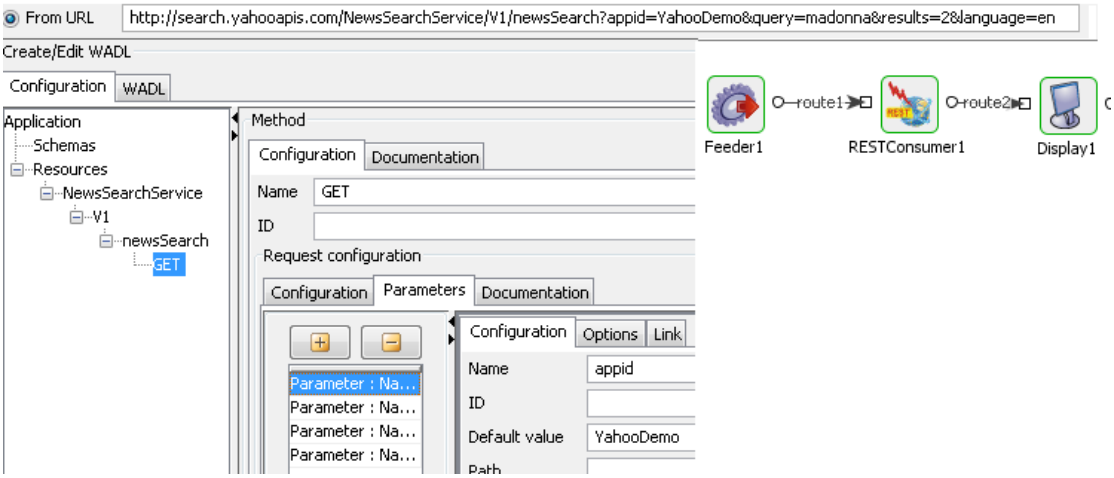


Figure 28: Invoking Yahoo RESTful search API using RESTConsumer.

Figure 28 illustrates using RESTConsumer to invoke the Yahoo Search API. The WADL is populated from the sample invocation of yahoo API using Load from URL functionality described in the configuration section. The sample request and response messages are as shown in figure 29 and figure 30 respectively.

```
<ns1:Request xmlns:ns1="http://www.fiorano.com/services/rest">
   <ns1:Request xmlns:ns1="http://www.fiorano.com/services/rest">
   <ResourceParameters>
      <NewsSearchService>
         <V1>
            <newsSearch/>
         </V1>
      </NewsSearchService>
   </ResourceParameters>
<RequestParameters>
      <appid>YahooDemo</appid>
      <query>RESTFul service</query>
      <results>1</results>
      <language>en</language>
   </RequestParameters>
</ns1:Request>
</ns1:Request>
```

Figure 28: Sample request

```
<Response Status="200">
   <ResponseParameters>
      <Connection>close</Connection>
      <Date>Wed, 09 Mar 2011 02:49:21 GMT</Date>
      <Content-Type>text/xml; charset=utf-8</Content-Type>
      <P3P>policyref="http://info.yahoo.com/w3c/p3p.xml", CP="CAO DSP COR CUR ADM DEV TAI PSA PSD IVAi IVDi CONi TELo OTPi OU
      <Transfer-Encoding>chunked</Transfer-Encoding>
      <Cache-Control>private</Cache-Control>
      <Vary>Accept-Encoding</Vary>
   </ResponseParameters>
   <Representation MediaType="text">
      <Element>
         <ResultSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="urn:yahoo:yn" xsi:schemaLocation="urn:yahoo:
            <Result>
               <Title>Thousand of APIs Paint a Bright Future for the Web</Title>
               <Summary>Once a novel idea that seemed limited to Flickr, the web-based API is now everywhere you turn — Twit
               <Url>http://www.webmonkey.com/2011/03/thousand-of-apis-paint-a-bright-future-for-the-web/</Url>
               <ClickUrl>http://www.webmonkey.com/2011/03/thousand-of-apis-paint-a-bright-future-for-the-web/</ClickUrl>
               <NewsSource>Wired News</NewsSource>
               <NewsSourceUrl>http://www.wired.com/</NewsSourceUrl>
               <Language>en</Language>
               <PublishDate>1299604019</PublishDate>
               <ModificationDate>1299604020</ModificationDate>
            </Result>
         </ResultSet>
      </Element>
   </Representation>
</Response>
```


Figure 28: Sample response

4.13.6.5 Useful Tips

- To invoke URL of a RESTful service, use the functionality Load WADL from URL as demonstrated in section 4.13.6.4.
- If RESTConsumer is used to invoke RESTStub which is running on a peer server with both jms ssl enabled and jetty ssl enabled, the keystore used to connect to jetty ssl is to be imported into jms ssl keystore %FIORANO_HOME%\esb\server\profiles\certs\jssecacerts.
- When the content type of request is application/octet-stream the content is sent as bytes content in the message, this will be present as bytes property. This can be retrieved as Base 64 encoded string in mapper using the funclet Byte-Content in JMS Message funclets.
- For invocation of RESTful services whose base URL is dynamic, use the elements ResourceURL and Parameters from the request schema.

4.14 WebService

The WebService category consists of components like Stub and WebServiceConsumer. The following section describes each component.

4.14.1 WSStub

This component is used to expose an Event Process or a part of Event Process as a Web Service deployed in the Peer Server. WSDL for the service deployed can be fetched when this component is running and the URL of the WSDL can be used by Web Service clients to access the Web Service. Each instance of the component hosts a single web service. The component can be configured in two ways:

- Manually defining the operations and related details in the UI
- Loading a predefined WSDL and editing the populated web service details if required

The WSDL generated for the component will differ from the WSDL provided in following aspects:

- End point address
- If the provided WSDL contains multiple services, only the selected service will be present

The component generates request, response and error ports for each operation present in the selected web service.

During the component launch, it checks if a web service with the same web service name is already deployed. If no other service with same name exits, the web service is deployed to the Jetty Server running in Peer Server that hosts the component.

The web service deployed in the Jetty accepts requests from web service clients and sends the SOAP XML using internal JMS client to the request port of the operation there by injecting the request into the Fiorano Event Process for processing the request. On processing the request, the response SOAP XML should be generated and sent to the corresponding response / error ports. The web service receives the response from the internal JMS client and in turn sends the response as SOAP response to the client that made the request.

4.14.1.1 Configuration

The component can be configured using the custom property sheet as shown in the following sections.

4.14.1.1.1 WS Definition

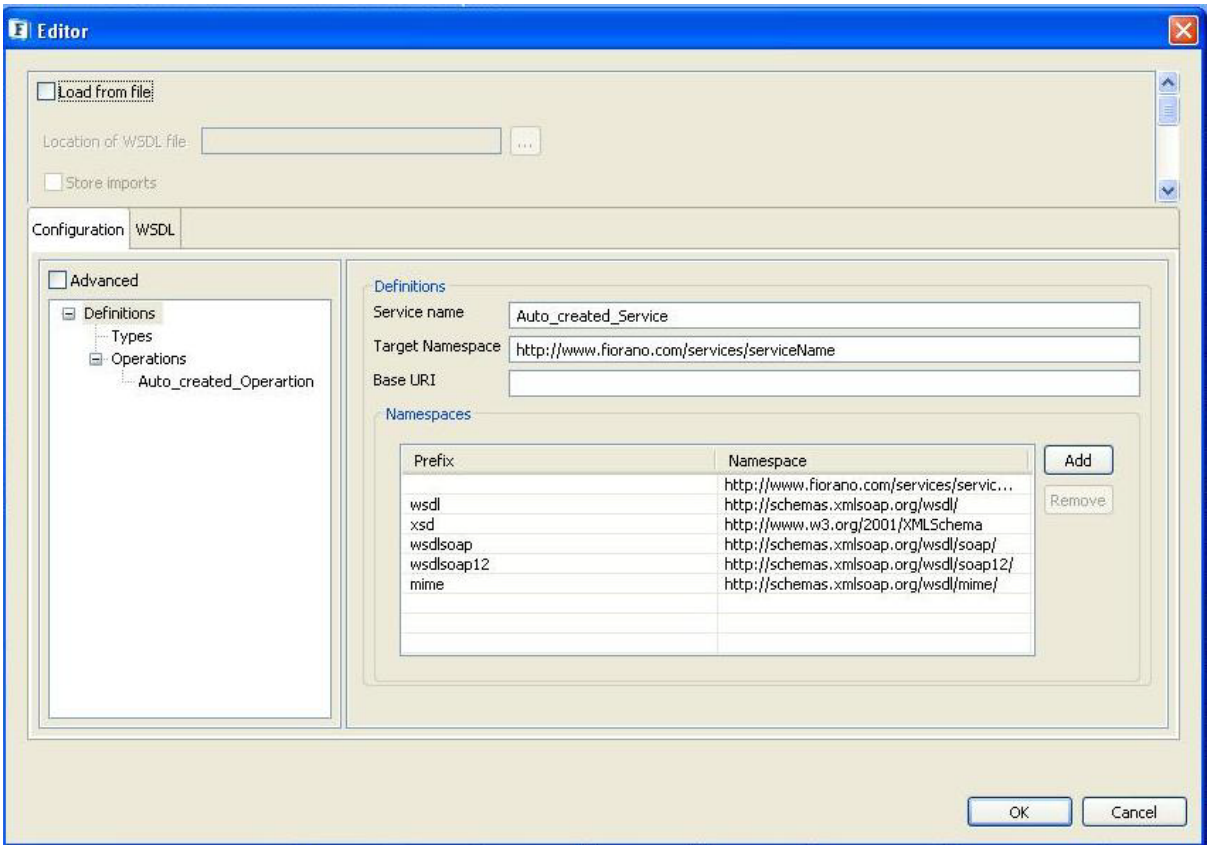


Figure 4.490: Configuration Property Sheet

This page is used to define the web service using a user interface for WSDL User can create a WSDL by manually defining different aspects of the service or can load an existing WSDL and edit the configurations.

4.14.1.1.1.1 Loading Existing WSDL

Existing WSDL file can be used to define the web service as shown in **Figure 4.491**. To load an existing WSDL file, the checkbox **Load from file** must be selected which shows the option below.

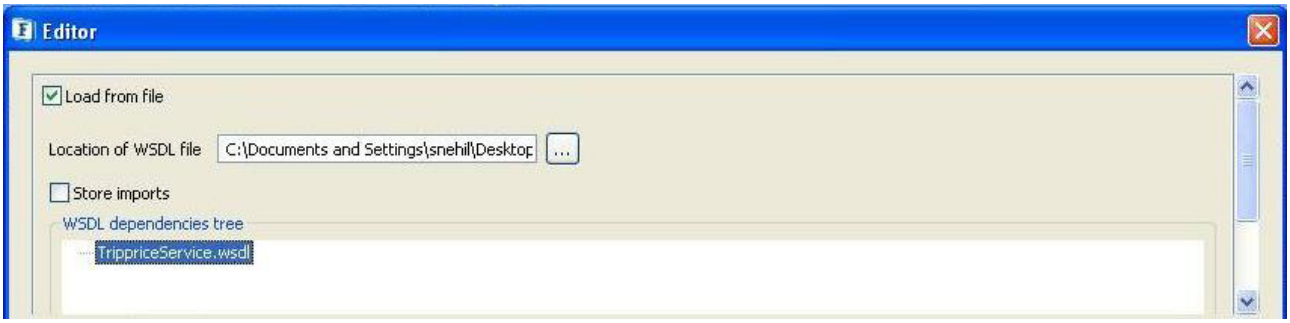


Figure 4.491: Loading Existing WSDL

Location of WSDL File

The WSDL file can loaded using the file dialog that opens up when the ellipsis is selected. The parent directory of the selected file is set as the base URI for this definition. It can be changed by clicking on the **definitions** node in Basic/Advanced view of the definition and changing the **Base URI** field. If the loaded WSDL file has more than one service defined, then the user is prompted to select one service for configuration as shown in **Figure 4.492**.

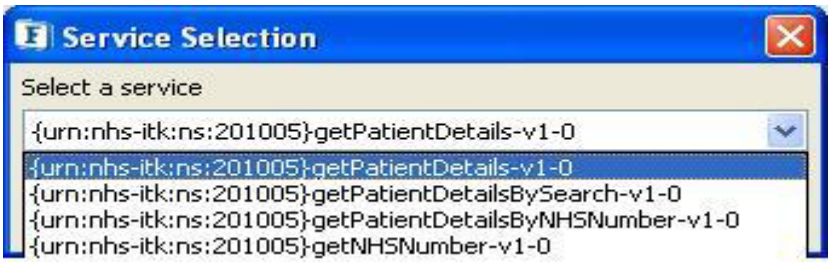


Figure 4.492: Service selection

Store Imports

This option can be selected if the WSDL files imported in the loaded WSDL have to be stored on the Jetty server. The imported files will be stored at FIORANO_HOME\esb\server\jetty\fps\webapps\bcwsgateway\wsdls\<ServiceName>\imports and their paths in the WSDL are updated when the component starts and deleted when the component stops.

Using this option allows the web service clients to access the import WSDLs as well relative to the actual WSDL. However, for this option to work the imported files should be present in the location specified in **Base URI** field and that location should be accessible from the Peer Server on which the component will be launched.

If this option is not specified, the web service clients should be provided with a copy of all imported WSDLs, if required.

WSDL dependencies Tree

WSDL dependency tree shows the WSDL and all the imported WSDL referenced directly or indirectly from the main WSDL. Each node in the tree represents a WSDL and the WSDLs that are imported in the WSDL represented by the node are represented by its child nodes. Selecting a node in this tree will show the corresponding WSDL definition in the panel below.

Editing or defining an imported WSDL is not currently. When a node representing an imported WSDL is selected the panel will be in read only mode.

4.14.1.1.1.2 WS Definition Views

Details of the selected WSDL are shown in two different views – Configuration (Basic, Advanced) and WSDL. However, web service can be configured using Basic and Advanced views only based on the expertise of the user.

4.14.1.1.2 Basic

This view is recommended for users who do not have a good knowledge of WSDL. In this view, web service definition can be created with the knowledge of input, output and fault type (if any). The SOAP bindings and other necessary elements for WSDL are created automatically when an operation is added in this view.

The web service definition can be configured by selecting the nodes in the tree on the left side. When a node is selected in the tree the panel corresponding to its configuration is shown in the right side pane.

4.14.1.1.2.1 Definition

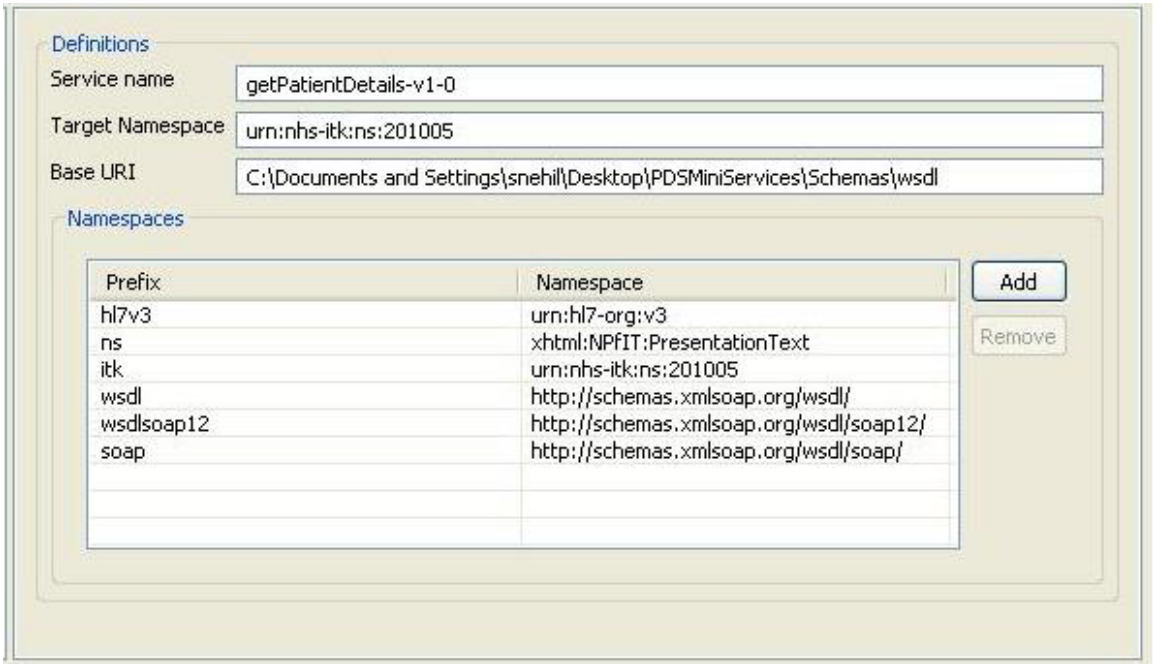


Figure 4.493: Definition Configuration

Service name: This field is used to specify the name of the service.

Target namespace: This field is used to specify the target namespace of the WSDL that is generated. Note when target namespace is changed an entry corresponding to its prefix must be added in the namespace prefix table.

Base URI: This field is used to provide the location from which the relative paths of imported WSDLs used in the selected WSDL will be resolved. By default it is populated with the location where the WSDL is present. It should be edited to the correct location if required.

If **Save Imports** is selected, this location should be accessible from the Peer Server on which the component will be launched.

Namespace prefix table: This table is used to assign prefixes to all namespaces that are used in the WSDL.

4.14.1.1.2.2 Types

The schemas which are necessary for defining the web services can be added in the view corresponding to the Types node as shown in **Figure 4.494**.

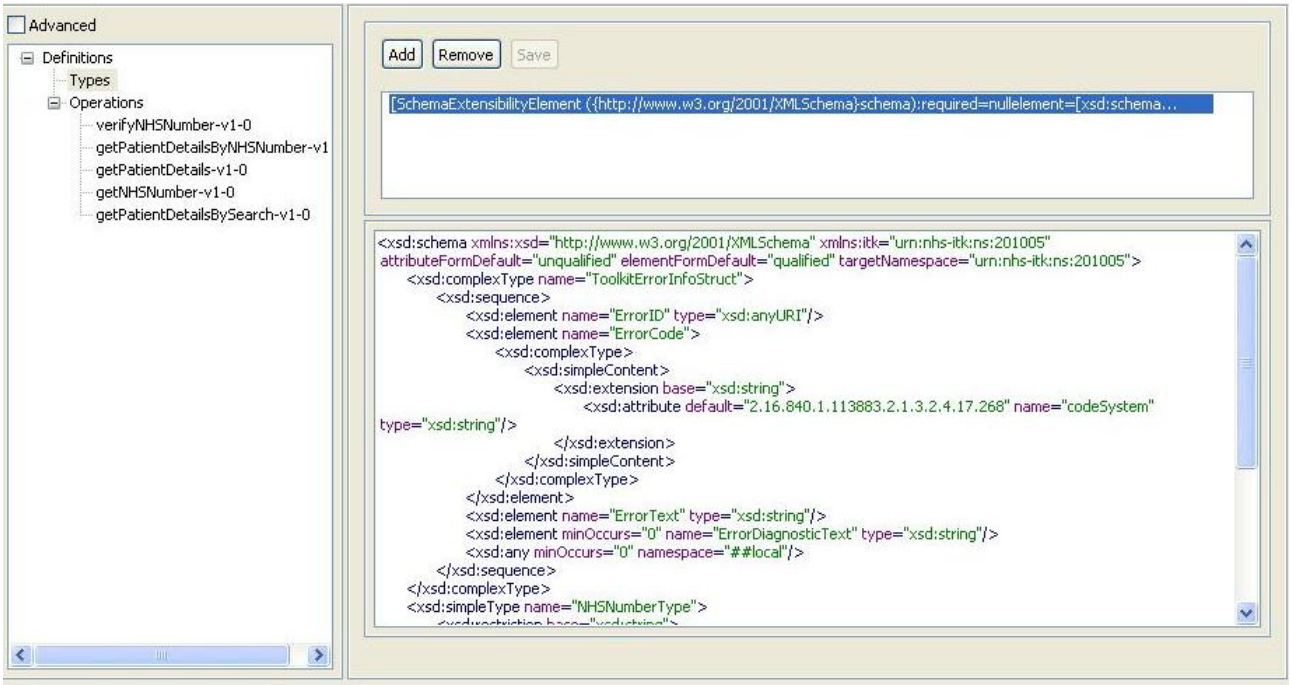


Figure 4.494: Types Configuration

The schema content can be loaded either from a file or manually by clicking the Add button and then saving the content.

The schemas that are added will be shown in the table as shown in the **Figure 4.494**. When a schema is added its target namespace and imported namespaces will be added to the WSDL with an auto generated prefix. Imported/included schemas if any have to be added to the schema repository.

4.14.1.1.2.3 Operations

In basic view, an operation can be added by selecting the option **Add Operation** from the context menu of the **Operations** node as shown in **Figure 4.495**.

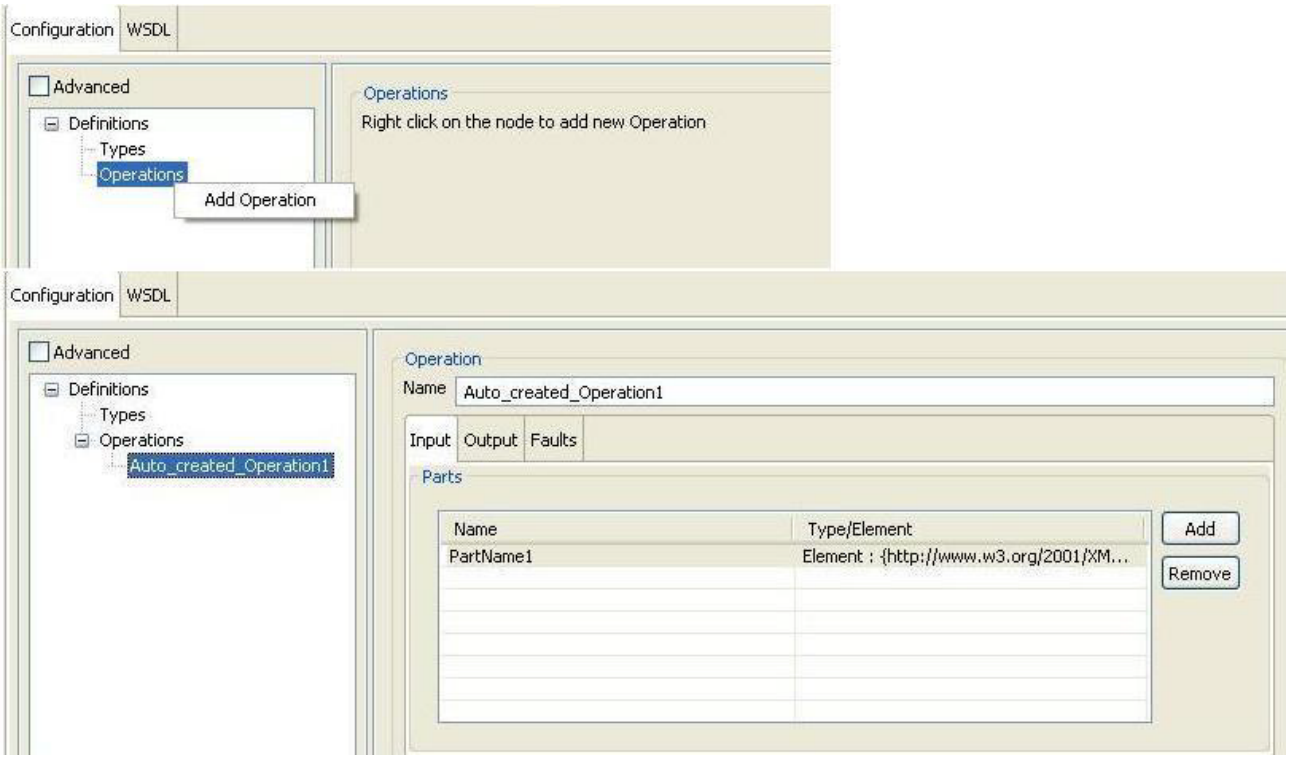


Figure 4.495: Basic Operation configuration

When an operation is added in basic view, the port type, binding, binding operation, Input and Output messages will be automatically created with prefix Auto_Created_. On deleting the operation from the basic view all corresponding elements which are no longer referenced will be deleted based on the input from the user.

The input and output and fault type/element must be specified using the dialog which gets opened when the ellipsis corresponding to input is clicked.

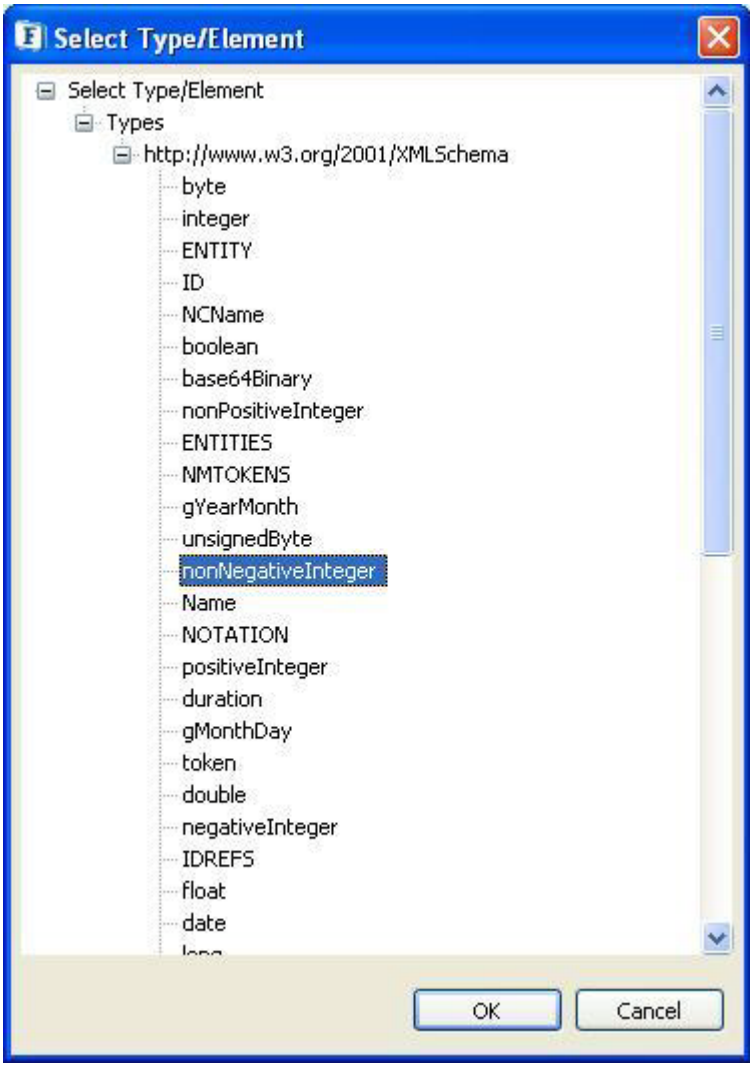


Figure 4.496: Type/Element Selection

The tree shows the types and elements present in the schemas and their imports as shown in **Figure 4.496**. The types are shown as children of **Types** node and any elements present will be shown under **Elements** node. The table cell corresponding to the types dialog is editable and is populated as follows.

When a type or element is selected, the cell will be populated with QName of the element/type prefixed by **Type :** or **Element :** respectively.

The QName can be given as **{namespace}localpart** manually. If there is no prefix attached then the QName will be assumed to be that of an element.

4.14.1.1.3 Advanced

This view is recommended for users who have fair understanding of creation of WSDL. In this view it is possible to configure all the elements of the WSDL. The tree in the advanced view shows much detailed view of the elements of the WSDL. The configuration of a particular element can be done by selecting the node which shows corresponding panel in the right side pane. For detailed information about elements of a WSDL please refer http://www.w3.org/TR/wsdl.

4.14.1.1.3.1 Definitions

This is same as the one shown in the Basic view.

4.14.1.1.3.2 Types

This is same as the one shown in the Basic view.

4.14.1.1.3.3 Messages

The messages for the web service can be added by selecting the option **Add Message** from the context menu which appears on right clicking on the **Messages** node. This will add a message node as a child node of the messages node. On selecting the added node, the panel as shown in **Figure 4.497** is displayed in the right side pane.

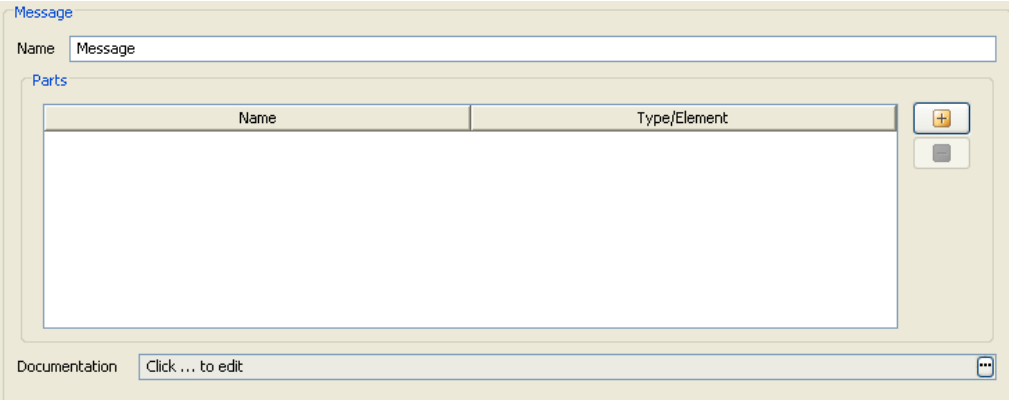


Figure 4.497: Message Configuration

Name: The name attribute of the message can be specified here. It provides a unique name among all messages defined within the enclosing WSDL document.

Parts: Messages consist of one or more logical parts. Each part is associated with a type from some type system using a message-typing attribute. Parts for the message can be added by using the add button in the table. The type/element for the part can be selected using the types dialog shown in the **Figure 4.496**.

4.14.1.1.3.4 Port Types

The port types for the web service can be added by selecting the option **Add Port type** from the context menu which appears on right clicking on the **Port Types** node. This will add a port type node as a child node of the **Port Types** node. On selecting the added node, the panel as shown in figure 9 is displayed in the right side pane.

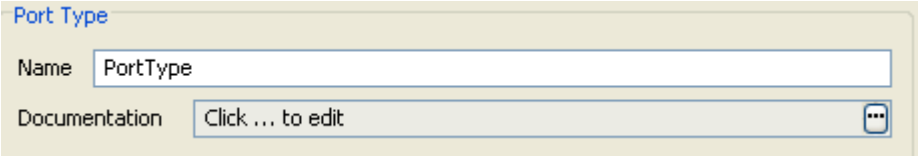


Figure 4.498: Port Type configuration

Name: The name attribute of the port type can be specified here. It provides a unique name among all port types defined within the enclosing WSDL document.

Operations: Operations can be added to a port by selecting the option **Add Operation** from the context menu which appears on right clicking on the added port type node as shown in **Figure 4.499**

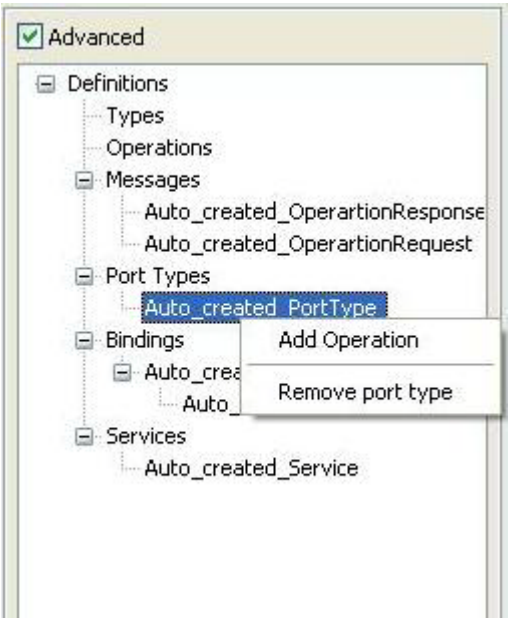


Figure 4.499: Adding Operation

This will add an operation node as a child node of the added port type node. On selecting the added node, the panel as shown in **Figure 4.500** is displayed in the right side pane.

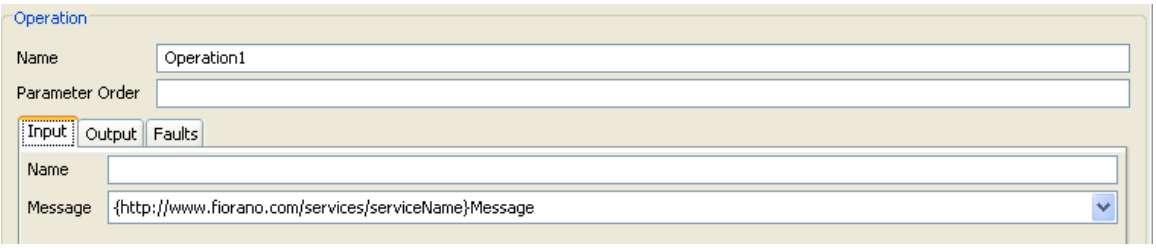


Figure 4.500: Advanced Operation Configuration

Name: The name attribute of the operation can be specified here. This need not be unique among the operations defined.

Parameter order: For RPC bindings, the parameter order can be specified using this field.

The value is a list of message part names separated by a single space. It MUST follow the following rules:

- The part name order reflects the order of the parameters in the RPC signature
- The return value part is not present in the list
    - If a part name appears in both the input and output message, it is an in/out parameter
    - If a part name appears in only the input message, it is an in parameter
    - If a part name appears in only the output message, it is an out parameter

Input tab

Name: The name of the input can be specified using this field. This name along with operation name and operation's output name uniquely identifies an operation within the enclosing port type.

Message: Input message can be selected from the combo box. It is recommended that the messages are added prior to configuring the operation.

Output tab

Name: The name of the output can be specified using this field. This name along with operation name and operation's input name uniquely identifies an operation within the enclosing port type.

Message: Output message can be selected from the combo box. It is recommended that the messages are added prior to configuring the operation.

Faults tab

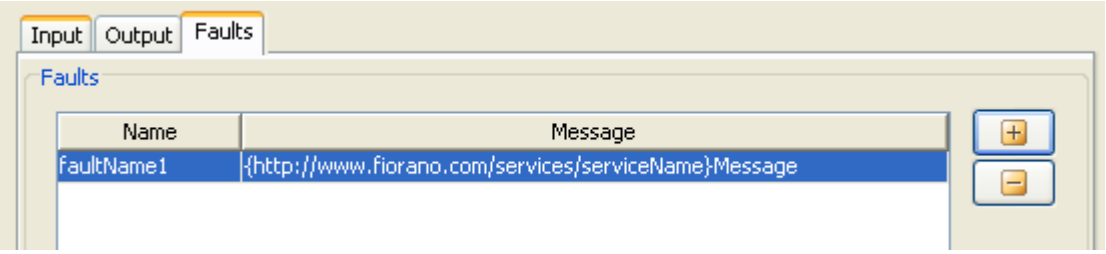


Figure 4.501: Adding faults to operation

Multiple faults can be added by using the table shown in **Figure 4.501**. The message corresponding to the fault can be selected from the drop down.

4.14.1.1.3.5 Bindings

The bindings for the web service can be added by selecting the option **Add Binding** from the context menu which appears on right clicking on the **Bindings** node. This will add a binding node as a child node of the **Bindings** node. On selecting the added node, the panel as shown in **Figure 4.502** is displayed in the right side pane.

Note: Port Types must be present in the definition or imported WSDLs before defining bindings

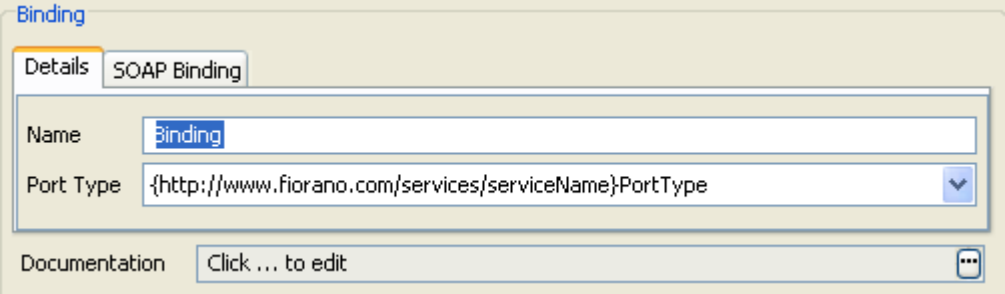


Figure 4.502: Binding configuration

Name: The name attribute of the binding can be specified here. It provides a unique name among all bindings defined within the enclosing WSDL document.

Port Type: The port type corresponding to this binding can be selected from the drop down list.

SOAP Binding

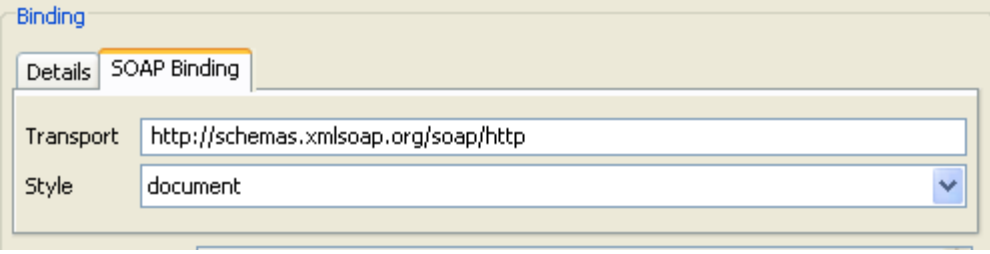


Figure 4.503: SOAP Binding configuration for binding

The SOAP binding information of the binding can be specified by selecting the tab SOAP binding as shown in the **Figure 4.503**.

Transport: The value of the required transport attribute indicates which transport of SOAP this binding corresponds to.

Style: The value of the style attribute is the default for the style attribute for each contained operation.

Binding Operations

Binding operations can be added to a port by selecting the option **Add Binding Operation** from the context menu which appears on right clicking on the added binding node as shown in **Figure 4.504.**

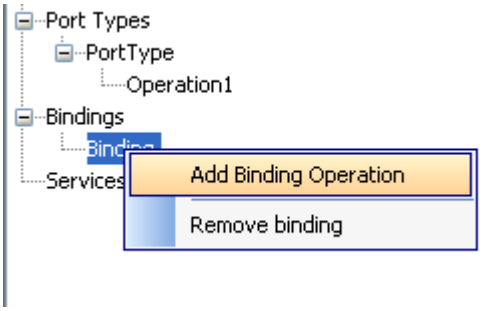


Figure 4.504: Adding binding operation

This will add binding operation node as a child node of the added binding node. On selecting the added node, the panel as shown in **Figure 4.505** is displayed in the right side pane

Note: Port Type corresponding to this binding must have operations defined for defining binding operation.

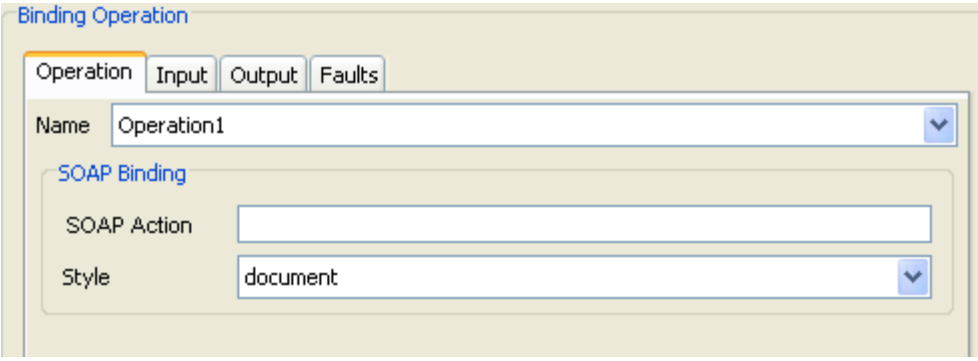


Figure 4.505: Binding operation configuration

The various details of binding operation can be specified by selecting the tabs as explained below.

4.14.1.1.3.5.1 Binding Configuration

Binding type: Select the type of binding to be used. It can be either simple soap or mime multipart. IF we need to send or receive attachments, then we need to select Binding type as mime multipart

SOAP Binding: The following binding (body and headers) information can be configured here when Binding type is **SOAP**.

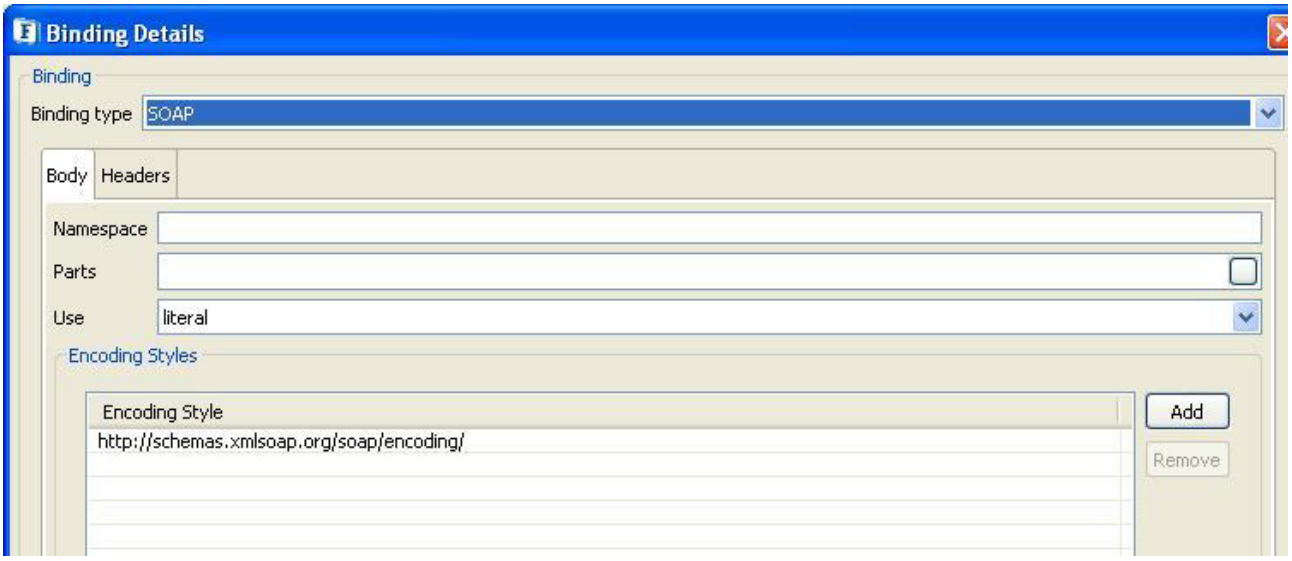


Figure 4.506: SOAP Binding Configuration

Body tab

This tab can be used to configure the attributes of the **soap:body** element.

Namespace: This can be used to specify the namespace corresponding to the SOAP body. This namespace should not be present in the namespaces defined in the definitions panel.

Parts: The optional parts attribute indicates which parts appear somewhere within the SOAP Body portion of the message. If the parts attribute is omitted, then all parts defined by the message are assumed to be included in the SOAP Body portion.

Use: The use attribute indicates whether the message parts are encoded using some encoding rules, or whether the parts define the concrete schema of the message.

Encoding Styles: The value of the encodingStyle attribute is a list of URIs. The URIs can be added using the table as shown in **Figure: SOAP Binding Configuration**. The URI represents encodings used within the message, in order from most restrictive to least restrictive.

Headers Tab

SOAP header corresponding to the input/output can be specified by adding them to the table as shown in the **Figure: 4.507**.

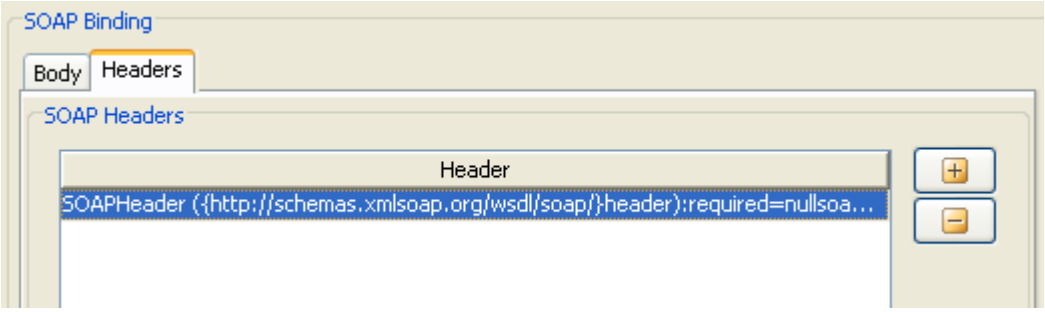


Figure 4.507: Adding SOAP Headers

The SOAP Headers can be configured by clicking on the ellipsis which opens the panel as shown in **Figure 4.508**.

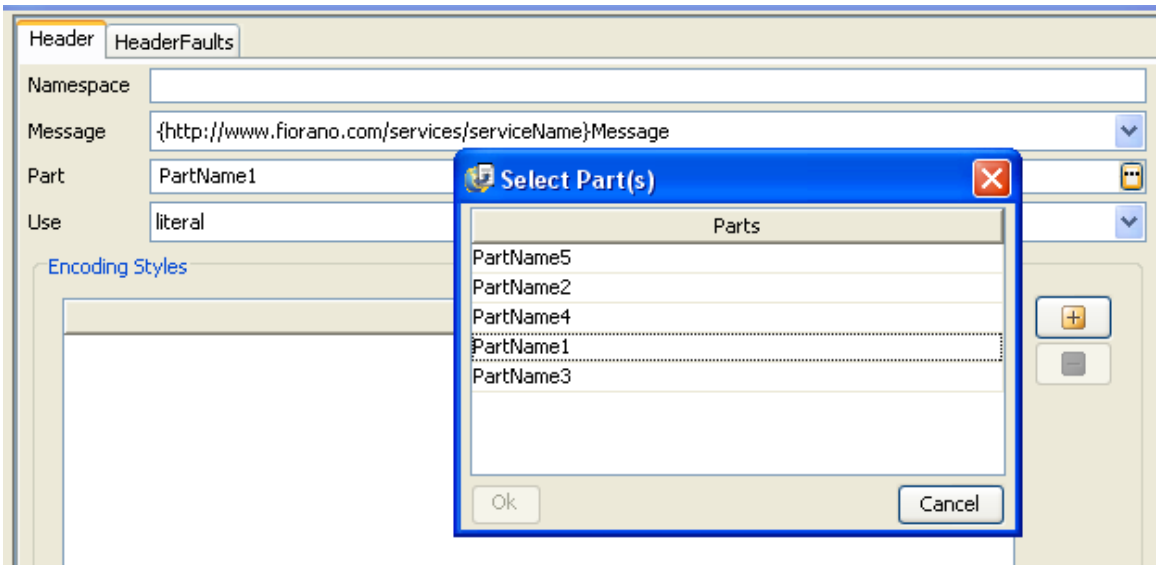


Figure 4.508: SOAP Header configuration

Namespace: This can be used to specify the namespace corresponding to the SOAP body. This namespace should not be present in the namespaces defined in the definitions panel.

Message: The message and the part reference the message part that defines the header type. The referenced message need not be the same as the message that defines the SOAP Body.

Part: The message and the part reference the message part that defines the header type. The schema referenced by the part MAY include definitions for the soap:actor and soap:mustUnderstand attributes if use="literal", but MUST NOT if use="encoded".

Use: The use attribute indicates whether the message part is encoded using some encoding rules, or whether the part defines the concrete schema of the message.

Encoding Styles: The value of the encodingStyle attribute is a list of URIs. The URIs can be added using the table as shown in **Figure 4.508**. The URI represents encodings used within the message, in order from most restrictive to least restrictive

Header Faults: The Header faults corresponding to a header can be defined by selecting the tab header Fault and adding an entry in the table **SOAP Header Faults** table. The SOAP Header fault can be configured by clicking on the ellipsis as shown in **Figure 4.509**.

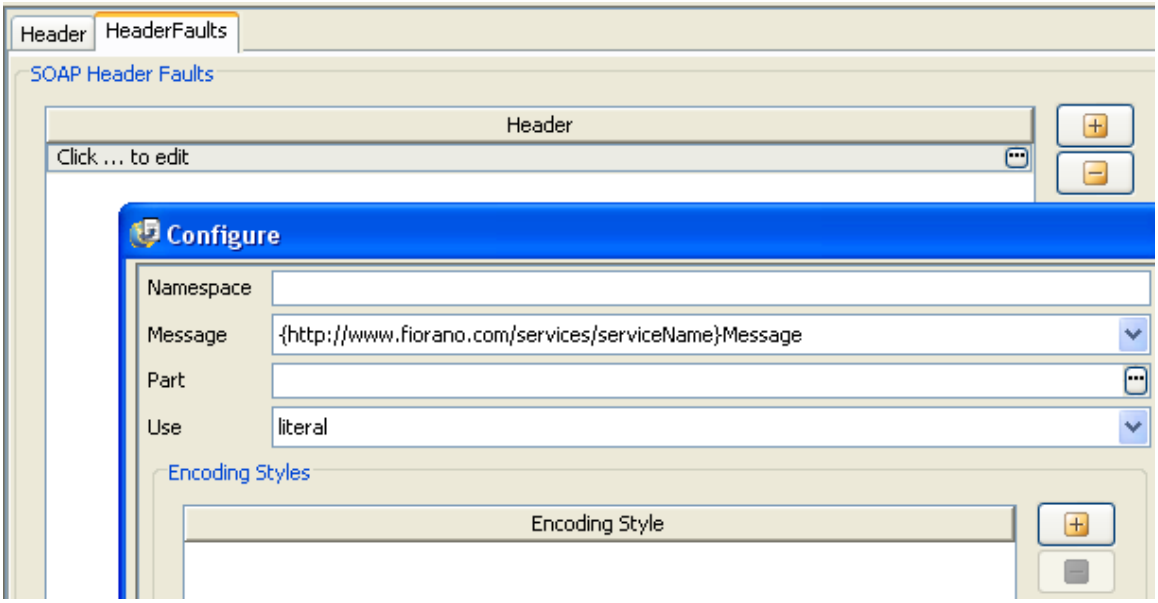


Figure 4.509: SOAP Header fault configuration

The header fault elements allows specification of the header type(s) that are used to transmit error information pertaining to the header defined by the soap:header. The attributes **namespace, message, part, use** and **encoding styles** are similar to those defined for SOAP Header.

MIME MultiPartRelated Binding

The following binding information can be configured here when Binding type is **MIME MultiPartRelated**.

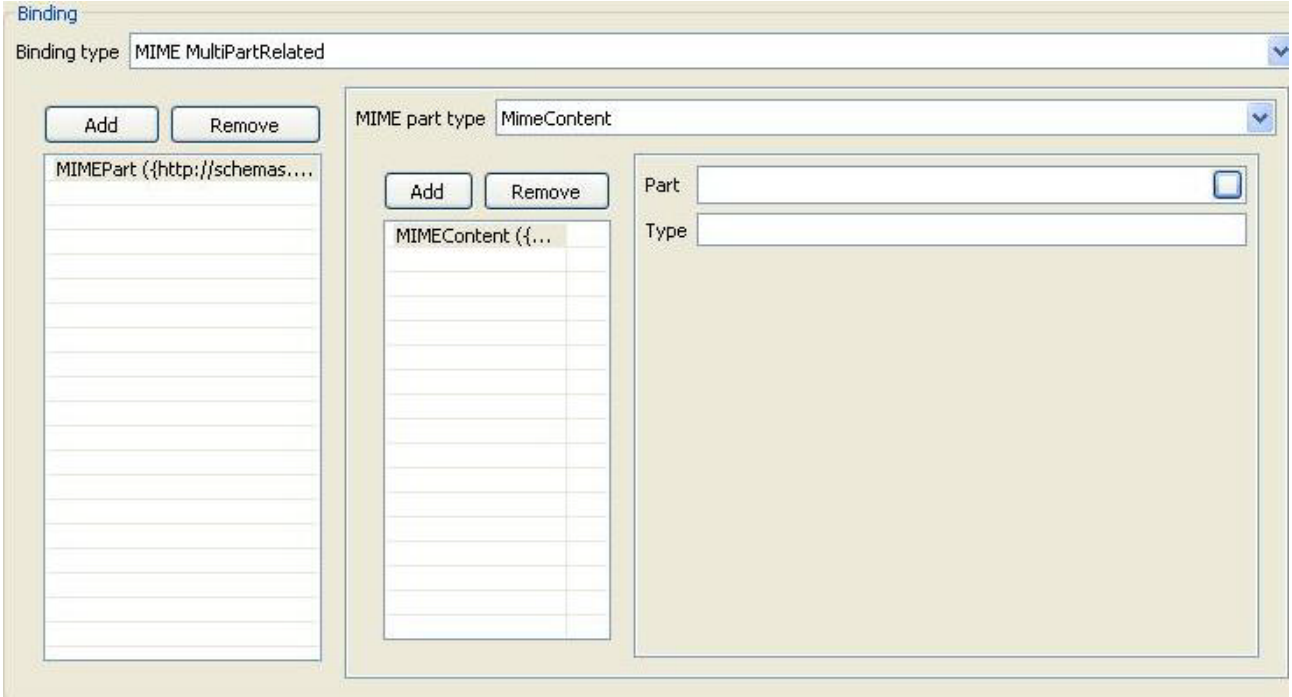


Figure: MIME MultiPartRelated Binding configuration

MIME part type: Mime part type defines the type of the mime part. We can select Mime part as either SOAP body or Mime Content.

MimeContent: MIME Contents can be added using the add button. Corresponding Part and Type of the MIME content can be configured.

Part: Select the corresponding Part.

Type: Type specifies the content type of this particular mime part as image/jpeg, text/*.
SOAP

This is same as SOAP Binding configuration explained above.

Operation tab

Name: The name of the operation to which this binding operation corresponds can be selected from the drop down.

SOAP Binding: This information corresponds to the attributes for the **soap:operation** element that will be added to the binding operation element.

SOAP Action: The soapAction attribute specifies the value of the SOAPAction header for this operation.

Style: The style attribute indicates whether the operation is RPC-oriented (messages containing parameters and return values) or document-oriented (message containing document(s))

Input tab

The input tab can be used to specify the binding input information as shown in **Figure 4.506**.

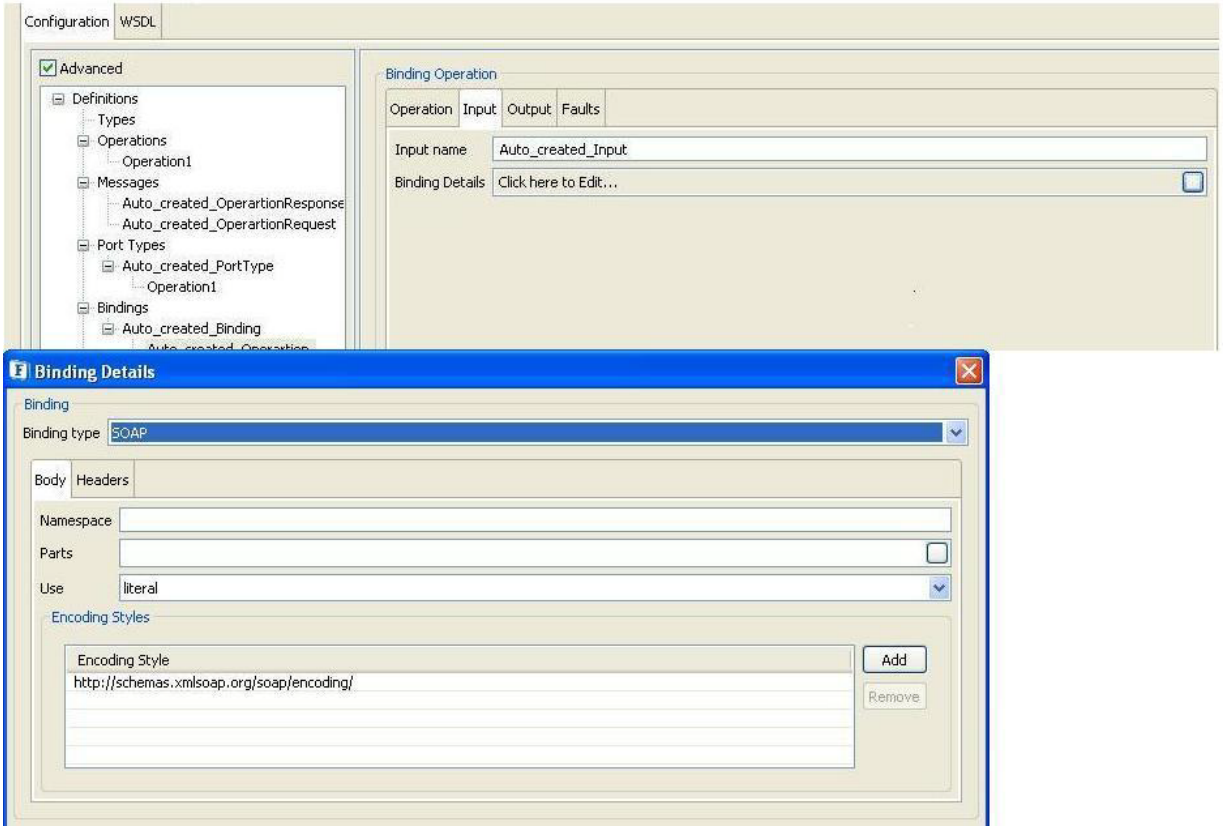


Figure: Binding input Configuration

Input name: The name of the input for the binding operation can be specified here. This along with operation name and the output name uniquely identifies the operation to which the binding operation corresponds to.

Binding: Section **4.14.1.1.3.5.1 Binding Configuration** explains binding configuration.

Output tab

The output tab can be used to specify the binding output information as shown in **Figure 4.510**.

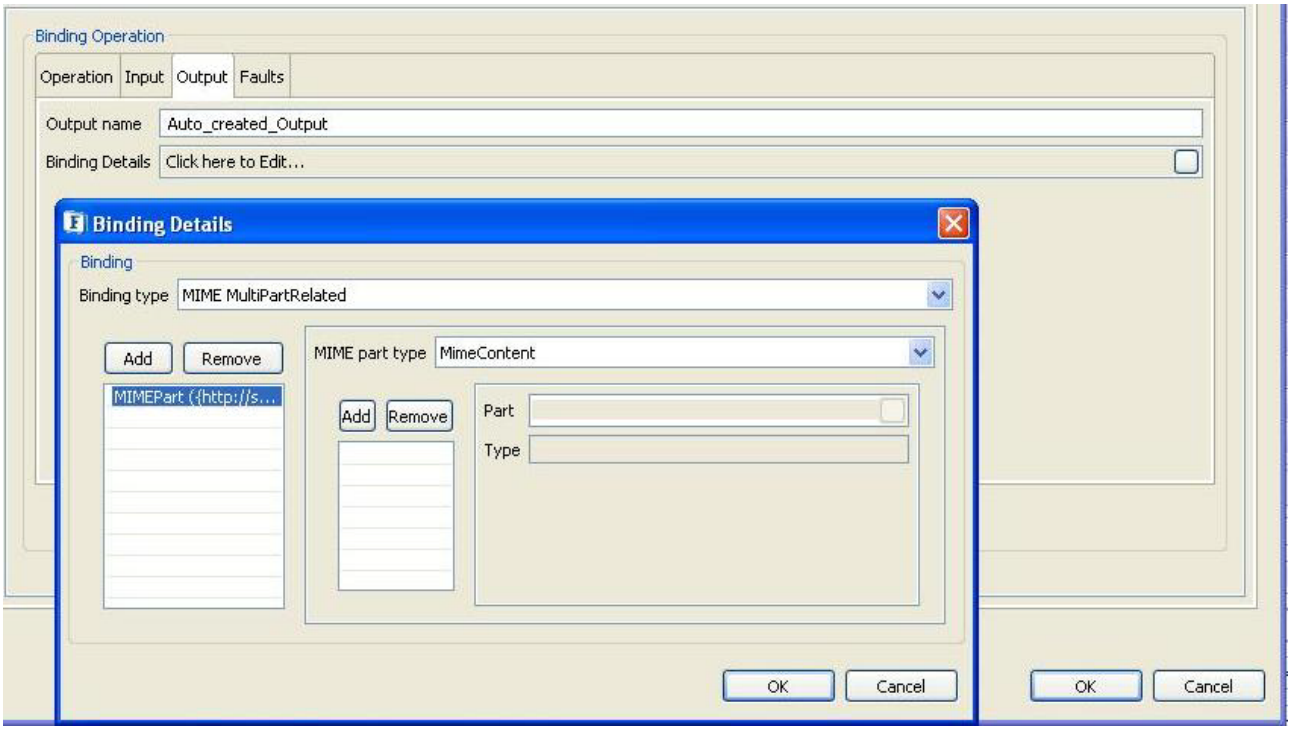


Figure 4.510: Binding Output configuration

Output name: The name of the output for the binding operation can be specified here. This along with operation name and the input name uniquely identifies the operation to which the binding operation corresponds to.

Binding: Section **4.14.1.1.3.5.1 Binding Configuration** explains binding configuration.

Faults tab

The binding information for faults can be defined by adding entry in the table and clicking on the ellipsis which opens dialog as shown in **Figure 4.511**.

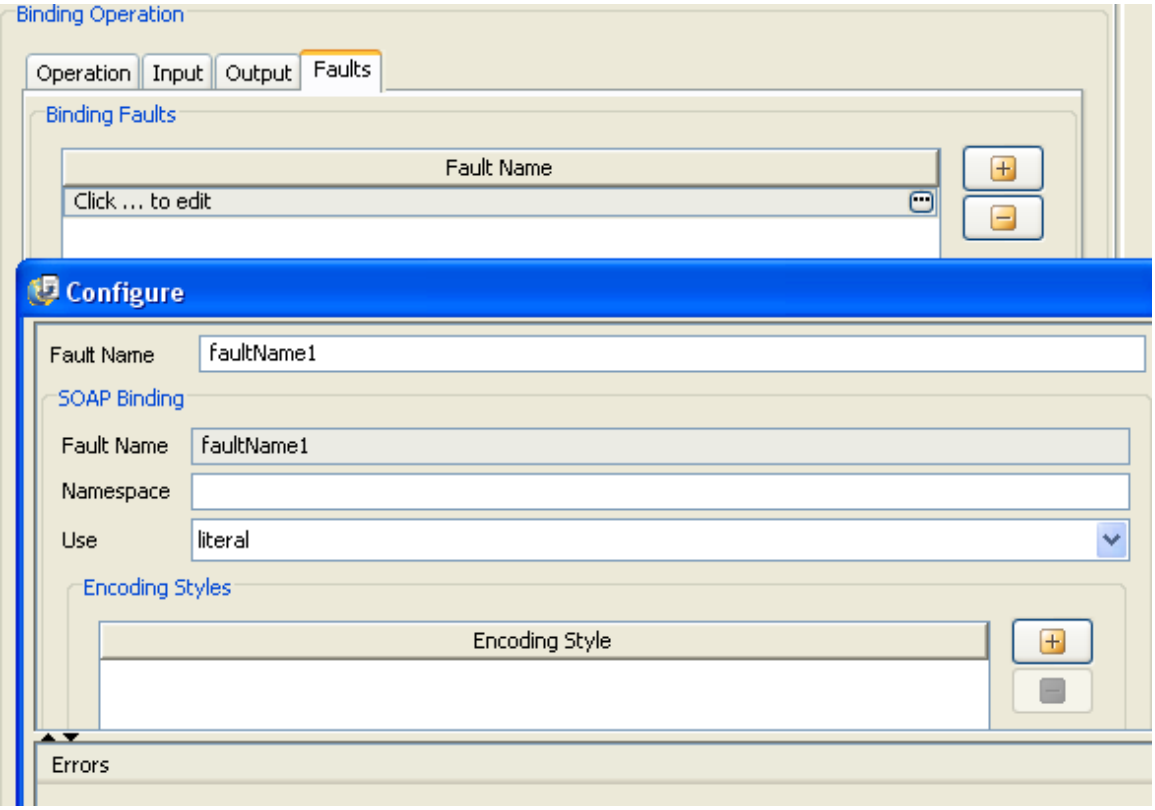


Figure 4.511: Binding Faults configuration

Fault Name: The name attribute of the fault element can be specified here.

SOAP Binding

Fault Name: The name attribute for soap:fault element. This is not editable and is same as the fault name.

Namespace: This can be used to specify the namespace corresponding to the SOAP body. This namespace should not be present in the namespaces defined in the definitions panel.

Use: The use attribute indicates whether the message parts are encoded using some encoding rules, or whether the parts define the concrete schema of the message.

Encoding Styles: The value of the encodingStyle attribute is a list of URIs. The URIs can be added using the table as shown in **Figure 4.511**. The URI represents encodings used within the message, in order from most restrictive to least restrictive

4.14.1.1.3.6 Services

The service for the web service can be added by selecting the option **Add Service** from the context menu which appears on right clicking on the **Services** node. This will add a service node as a child node of the services node. On selecting the added node, the panel as shown in **Figure 4.512** is displayed in the right side pane. **Only one service per instance is supported now.**

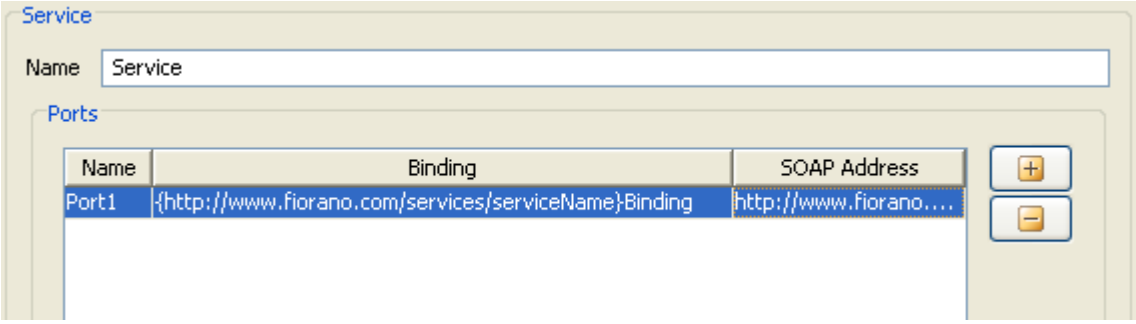


Figure 4.512: Service and ports configuration

Name

The name of the service can be specified here. The service name must be unique for WSStub instances configured on same peer server.

Ports

A port defines an individual endpoint by specifying a single address for a binding.

Name: The name attribute of the port can be specified here. It provides a unique name among all ports defined within the enclosing WSDL document.

Binding: The binding corresponding to port can be specified from the combo box.

SOAP Address: The SOAP address binding is used to give a port an address (a URI). The URI scheme specified for the address must correspond to the transport specified by the soap:binding.

4.14.1.1.4 WSDL

This is a read only view of the WSDL that is generated from the configuration done using basic/advanced views.

Figure 4.513: WSDL view

4.14.1.1.5 Common Panels

4.14.1.1.5.1 Errors

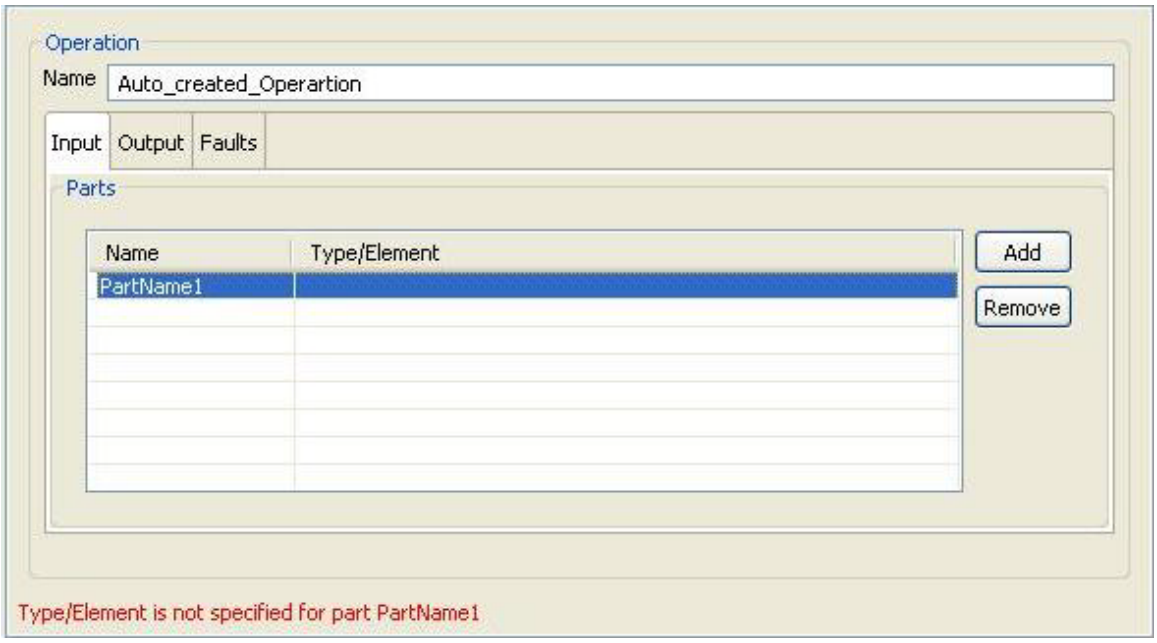


Figure 4.514: Errors Panel

The validation for a particular panel is done when the user makes changes corresponding to the panel. Validation errors corresponding to the particular panel are shown as above.

4.14.1.1.6 WS Standards

This page is used to define WS Standards like WS-Addressing, WS-Reliable Messaging and WS-Security.

4.14.1.1.6.1 WS Security

Request Security

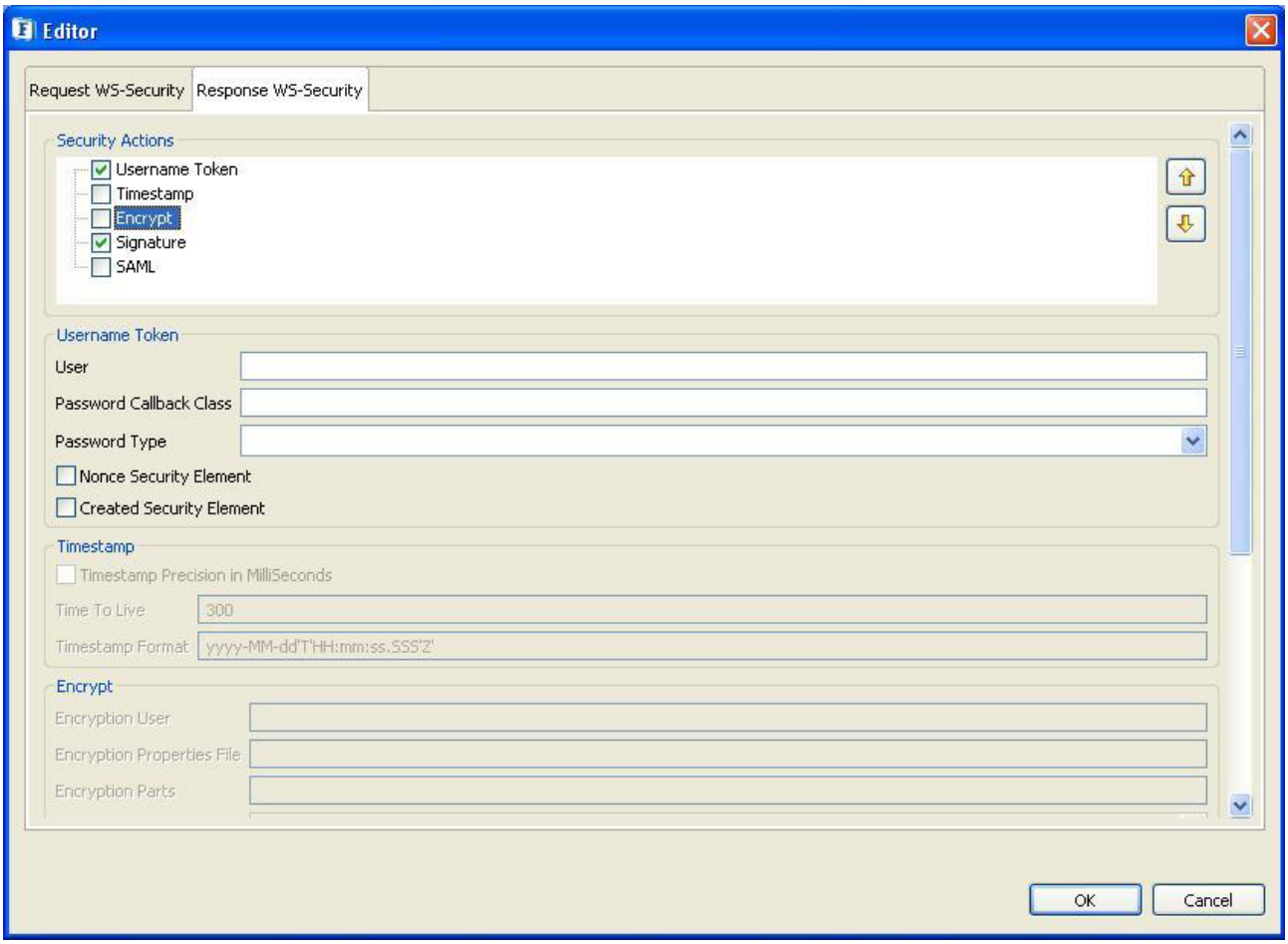


Figure 4.516: WS Request Security

Security actions

Ignore Order

If this is set, Order of Security actions will be ignored for the in coming request.

If this is not set, the order of a security function determines when this function will be applied when multiple security functions are being used. The order of the actions can be increased and decreased by using the and respectively. To enable an action/security function the check box against it must be selected. In **Figure 4.516** Username Token and Timestamp functions are enabled and the Username Token action will be applied followed by Timestamp action.

The different actions and the properties to be set for using them are explained below.

Username Token

Determines whether the server should perform Username token identification or not.

Password Callback Class

This is needed by the security functions to get the password and to verify the username/password pair. The password callback class should implement org.apache.ws.security.WSPasswordCallback class. This Password Callback class should be the fully qualified name of the class. The jar which contains the password callback class must be placed in '%FIORANO_HOME%\esb\server\jetty\fps\webapps\bcwsgateway\WEB-INF\lib'.

Is Password Required

This property should set to false if the Username security token is used without a password. No need to provide Password callback class if this property is set to no.

Timestamp

Determines whether the soap request has timestamp in security header or not. In this case, the message is valid up to 5 minutes or 300 seconds after the creation of the message.

Timestamp Precision in MilliSeconds

If this is set, timestamps will have precision in milliseconds. Otherwise it will be seconds.

Encrypt

Determines whether the soap request (or some parts of it) is encrypted or not.

Encryption Properties File Name

The name of the crypto property file to use for decryption of the soap request. If this parameter is not specified and if both the "Signature Properties filename" and "Signature" are set to yes, then the decryption function uses signature property file. Otherwise the handler throws an AxisFault. The properties file must be placed in '%FIORANO_HOME%\esb\server\jetty\fps\webapps\bcwsgateway\WEB-INF\classes'

An example of a properties file content:

```
org.apache.ws.security.crypto.provider=org.apache.ws.security.components.crypto.Merlin
org.apache.ws.security.crypto.merlin.file=C:\\Documents and
Settings\\Fiorano\\Desktop\\fiorano.ks
org.apache.ws.security.crypto.merlin.keystore.type=jks
org.apache.ws.security.crypto.merlin.keystore.password=fioranopass
org.apache.ws.security.crypto.merlin.keystore.alias=fiorano
org.apache.ws.security.crypto.merlin.alias.password=fioranopass
```

The properties descriptions:

- **org.apache.ws.security.crypto.provider**: Implementation class for security provider. Fiorano internally uses bouncycastle, to use the same this property must be set to "org.apache.ws.security.components.crypto.Merlin". To use other providers, the provider jar must be placed in '%FIORANO_HOME%\esb\server\jetty\fps\webapps\bcwsgateway\WEB-INF\lib' and fully qualified name of appropriate provider class has to be set for this property.
- **org.apache.ws.security.crypto.merlin.file**: The path to the keystore file.
- **org.apache.ws.security.crypto.merlin.keystore.type**: The keystore type, for example JKS for the Java key store. Other keystore type, such as pkcs12 are also possible but depend on the actual Crypto implementation.
- **org.apache.ws.security.crypto.merlin.keystore.password**: The password to read the keystore. If this property is not set, then the pwcallbackproperty must be defined.
- **org.apache.ws.security.crypto.merlin.keystore.alias**: The alias name under which the private key/certificate stored in keystore
- **org.apache.ws.security.crypto.merlin.alias.password**: Password for private key/certificate inside keystore stored under given alias (not used for encryption)

Signature

Determines whether the soap request (or some parts of it) is signed or not.

Signature Properties File Name

The name of the crypto property file to use for SOAP Signature. Please see the description of "Encryption Properties filename" for the details of the properties file.

SAML

Determines whether the server should perform SAML Token identification or not.

Signed SAML

If Signed SAML is used at client side for request, then the client performs two actions inserting a SAML Token (unsigned) and an associated Signature. So define both the actions SAML Unsigned and Signature here to resolve these security headers.

Response Security

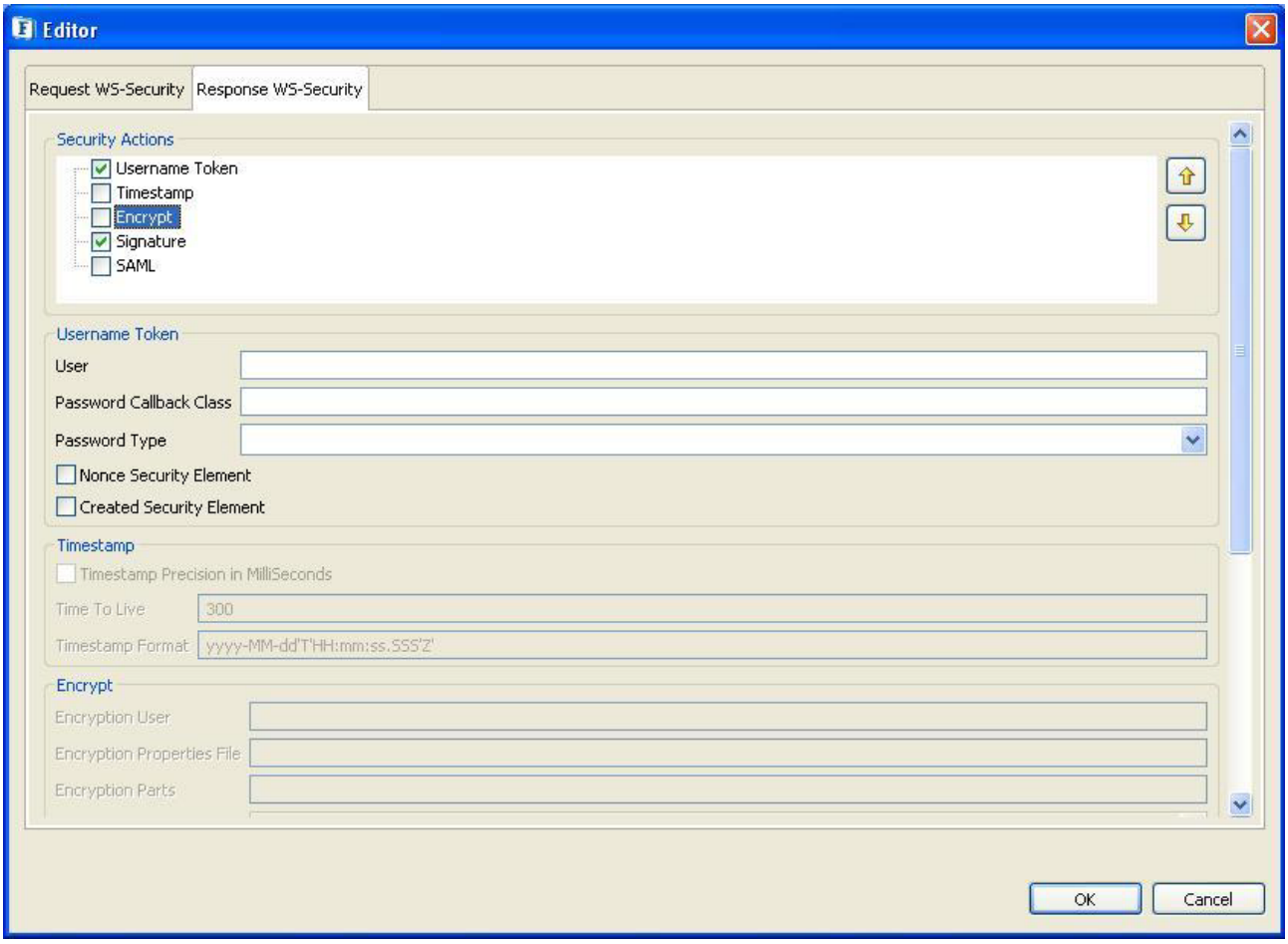


Figure 4.517: WS Response security

Order of actions

The order of a security function determines when this function will be applied when multiple security functions are being used. The order of the actions can be increased and decreased by using the ⇧ and ⇩ respectively. To enable an action/security function the check box against it must be selected. In the **Figure 4.517** Username Token, Timestamp and encrypt functions are enabled and they will be applied in the same order.

The different actions and the properties to be set for using them are explained below.

Username Token

Determines whether the server should perform Username Token Identification on response or not. If this property is set, the response from server contains Username and Password security headers.

User

This property is used as username for the UsernameToken security function. It is also used as the alias name in the keystore to get user's certificate or private key to perform signing for the Signature security function. It is also used as the fallback for the encryption security function in case of "Encryption User" is not specified and "Encryption" in response is enabled.

Password Callback Class

This is needed by the security functions to get the password and to verify the username/password pair. The password callback class should implement org.apache.ws.security.WSPasswordCallback class. This Password Callback class should be the fully qualified name of the class. The jar which contains the password callback class must be placed in '%FIORANO_HOME%\esb\server\jetty\fps\webapps\bcwsgateway\WEB-INF\lib'.

Password Type

Password type: The Password type can be either PasswordText or PasswordDigest

PasswordText

Password is sent in raw text format with in the security header of the soap response.

PasswordDigest

Password is sent in digest format with in the security header of the soap response.

PasswordNone

No password will be send in the security header. This option is useful when user wants to specify the username without any password.

Nonce Security Element

Specifies whether to use nonce element in the security header or not. When UsernameToken security function is used, then nonce security element can be employed to prevent message replay attacks. A nonce is a random value that the client creates to include in each UsernameToken that it sends. Although using a nonce is an effective countermeasure against replay attacks, it requires server to maintain a cache of used nonce’s, consuming server resources.

Created Security Element

Specifies whether to use Created element in the security header or not. This element denotes the time of creation of a nonce. Combining a nonce with a creation timestamp has the advantage of allowing a server to limit the cache of nonce’s to a "freshness" time period, establishing an upper bound on resource requirements.

Timestamp

Specifies whether the soap response from server should contain timestamp header or not.

Timestamp Precision in MilliSeconds

If this is set, timestamps will have precision in milliseconds. Otherwise it will be seconds.

Time To Live

The time difference between creation and expiry time in the WSS Timestamp. This should be specified in seconds.

Encrypt

Determines whether the server should encrypt the soap response (or some parts of it) or not.

Encryption User

Username for the encryption function. The encryption function uses the public key of this user's certificate. If this parameter is not set, then the encryption function falls back to the "User" parameter to get the certificate. The encrypt function will not authenticate user. So there is no need to set any password call back class for encrypt.

Encryption Properties File

The name of the crypto property file to use for SOAP Encryption in response. If this parameter is not specified and if both "Signature Properties filename" and "Signature" in response are set, then the encryption function uses signature property file. Otherwise the handler throws an AxisFault. The properties file must be placed in '%FIORANO_HOME%\esb\server\jetty\fps\webapps\bcwsgateway\WEB-INF\classes'. Please see the description of "Encryption Properties filename" in request security for the details of the crypto properties file.

Encryption Parts

Parameter specifies which parts of the response shall be encrypted. The value of this parameter is a list of semi-colon separated element names that identify the elements to encrypt. An encryption mode specifies and a namespace identification, each inside a pair of curly brackets, may precede each element name. The encryption mode is either {Content} or {Element}. 'Element' encryption mode will encrypt the entire element including start and end tags. 'Content' encrypt mode will encrypt only the content of the specified element. The default encryption mode is 'Content'.

For example if we set "{Element}{http://example.org/paymentv2}CreditCard;{}{}UserName" list to this property, then the first entry of the list identifies the element CreditCard in the namespace http://example.org/paymentv2, and will encrypt the entire element. In the second entry the encryption modifier and the namespace identifier are omitted. In this case the encryption mode defaults to Content and the namespace is set to the SOAP namespace.

The element name, the namespace identifier, and the encryption modifier are case sensitive. To specify an element without a namespace use the string Null as the namespace name (this is a case sensitive string). If no list is specified, the handler encrypts the SOAP Body in Content mode by default.

Encryption Key Identifier

Select the key identifier type to use.

DirectReference

The security function takes the signing certificate, converts it to a BinarySecurityToken, puts it in the security header. Thus the whole signing certificate is transferred.

X509KeyIdentifier

The encryption method uses the public key associated with this certificate to encrypt the symmetric key used to encrypt data. The certificate is converted into a KeyIdentifier token and sent to the server. Thus the complete certificate data is transferred.

SKIKeyIdentifier

The security function uses SKIKeyIdentifier.

IssuerSerial

The encryption method uses the public key associated with this certificate to encrypt the symmetric key used to encrypt data. The issuer name and the serial number of the signing certificate are sent to the server.

Signature

Determines whether the soap response (or some parts of it) should be signed or not.

Signature User

This name is used as the alias name in the keystore to get user's certificate and private key to perform signing. If this parameter is not set, then the signature function falls back to the "User" parameter to get the certificate. Password for the user to get certificates from the keystore should be provided in the Password Callback class.

Signature Properties File

The name of the crypto property file to use for SOAP Signature. Please see the description of "Encryption Properties filename" for the details of the properties file.

Signature Parts

Parameter specifies which parts of the request shall be signed. Please see the description of "Encryption Parts" for the syntax.

Signature Key Identifier

Select the key identifier type to use. Please see the description of "Encryption Key Identifier" for the descriptions of key identifiers.

Signature Algorithm

Parameter specifies signature algorithm to be used. If algorithm is not specified then the algorithm http://www.w3.org/2000/09/xmldsig#rsa-sha1 will be used by default.

Canonicalization Method

Parameter specifies canonicalization method to be used in the process of signining the request. If no method is specified then the method http://www.w3.org/2001/10/xml-exc-c14n# will be used by default.

SAML

Determines whether the server should perform SAML Token Identification or not.

Signed SAML

Specifies whether to use signed SAML or unsigned SAML. If Signed SAML is used, then server performs two actions inserting a SAML Token (unsigned) and an associated Signature. So define both the actions SAML Unsigned and Signature at client side to resolve these security headers. If Signed SAML is used, the signature properties should be specified in this panel with out selecting the property "Signature WS-Security".

SAML Properties File

The name of the SAML properties file. This file must be placed in '%FIORANO_HOME%\esb\server\jetty\fps\webapps\bcwsgateway\WEB-INF\lib'.

An example of properties file content:

```
org.apache.ws.security.saml.issuerClass=org.apache.ws.security.saml.SAMLIssuerImpl
org.apache.ws.security.saml.issuer.cryptoProp.file=crypto_wsc.properties
org.apache.ws.security.saml.issuer.key.name=fiorano
org.apache.ws.security.saml.issuer.key.password=fioranopass
org.apache.ws.security.saml.issuer=fiorano
org.apache.ws.security.saml.subjectNameId.name=uid=mule,ou=people,ou=saml-
demo,o=example.com
org.apache.ws.security.saml.subjectNameId.qualifier=www.example.com
org.apache.ws.security.saml.authenticationMethod=password
#org.apache.ws.security.saml.confirmationMethod=senderVouches
org.apache.ws.security.saml.confirmationMethod=keyHolder
```

4.14.1.1.7 Transport Security

4.14.1.1.7.1 SSL Configuration

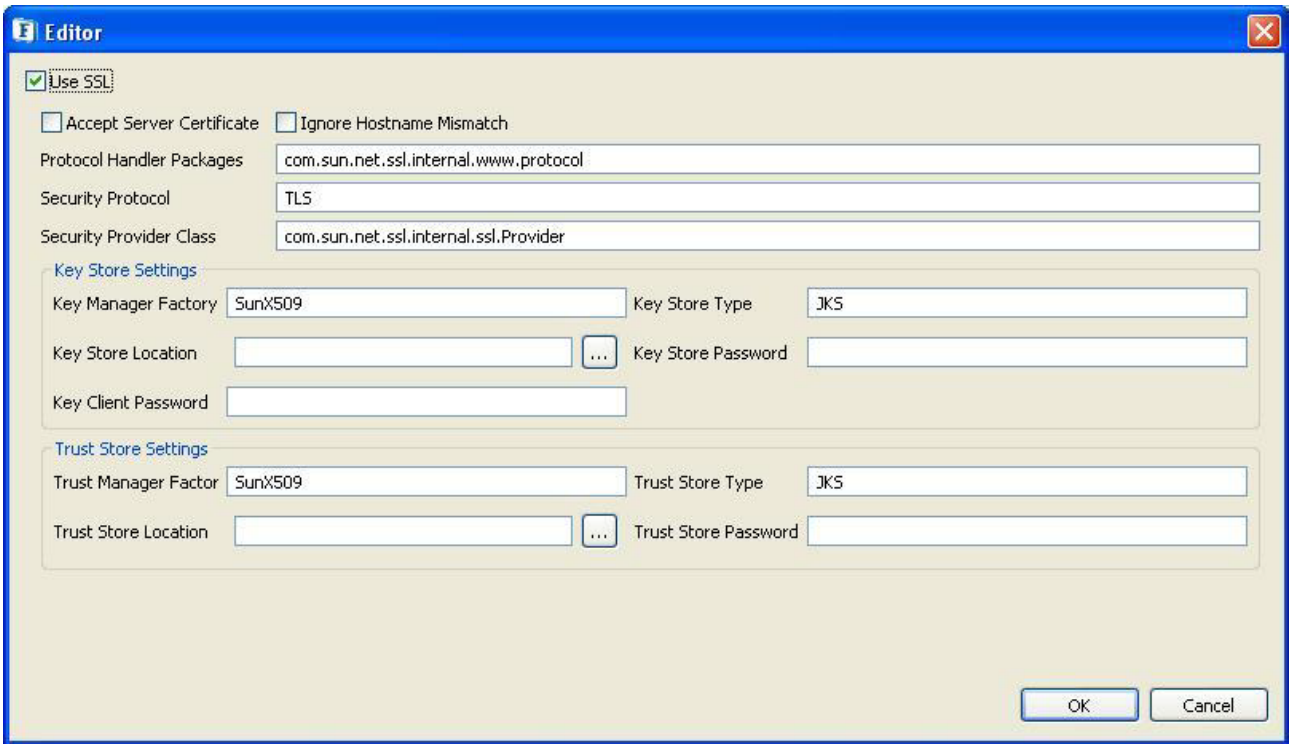


Figure 4.518: SSL Configuration

Use SSL

Select this option to enable SSL Settings. Rest of the properties in this editor are enabled and configurable only when this property is checked.

Accept Server Certificate

When accessing https URLs, this property determines whether the server certificates should be accepted or not. If selected, the certificate will be accepted without any validation, otherwise exception is sent to ON_EXCEPTION port.

Ignore Hostname Mismatch

If this option is selected the certificate will be accepted even if hostname in the certificate does not match with the hostname in the request URL. If its not selected exception is thrown.

Protocol Handler Packages

Determines protocol handler packages property.

Security Protocol

Determines Security protocol

Security Provider Class

Determines Security provider class.

Key Manager Factory Type

Algorithm for the Key Manager Factory.

Key Store Type

Type of the Key Store whose location is specified by **Key Store Location** should be specified in the fileld.

Key Store Location

Location of the key store file can be provided using the file dialog that opens up on clicking the ellipsis . The KeyStore is used by the component for client authentication.

Key Store Password

Password of the specified key store can be specified in the field.

Key Store Client Key

Determines Key Store Client Key

Trust Manager Factory Type

Algorithm for the Trust Manager Factory.

Trust Store Location

Location of the trust store file should be specified. TrustStore is a file where digital certificates of trusted sites are stored and retrieved for authentication during an SSL connection. TrustStore is used to authenticate a server in SSL authentication.

Trust Store Type

Type of Trust Store whose location is specified by property **Trust Store Location.**

Trust Store Password

Password of the specified trust store should be specified in the field.

4.14.1.1.7.2 HTTP Authentication

Figure 4.519: HTTP Authentication

Use HTTP Authentication

HTTP authentication support for the Web Services can be provided by selecting this property.

Type

The type of HTTP Authentication can be selected for the drop down. At present only **BASIC** authentication is supported.

Username

Username to be used by the WSStub component while deploying the Web Service. The entry for this user name has to be present in Realm.properties file. If this user name is not present, the component will not launch.

Password

Password to be used by the WSStub component while deploying the Web Service. If this password is not present in Realm.properties file the component will not launch.

Using Basic Authentication

Before enabling this property in WSStub, basic authentication has to be enabled in FPS Jetty Server and in bcwsgateway web application (FIORANO_HOME/esb/server/jetty/fps/webapps/bcwsgateway). Procedure to enable basic authentication in Jetty Server and in the web app is explained below.

Enabling Basic Authentication in Jetty Server

1. Stop the FPS Server if it is running.
2. Open FPS profile in Fiorano Studio and navigate to FPS→Fiorano→Esb→Jetty.

3. Select the Jetty mbean. In the properties window, set the property Basic Authentication to yes and give the fully qualified path of Realm.properties file against the property Realm Properties. More information on Realms and Realm.properties file content is discussed in the next section.
4. Save the profile.

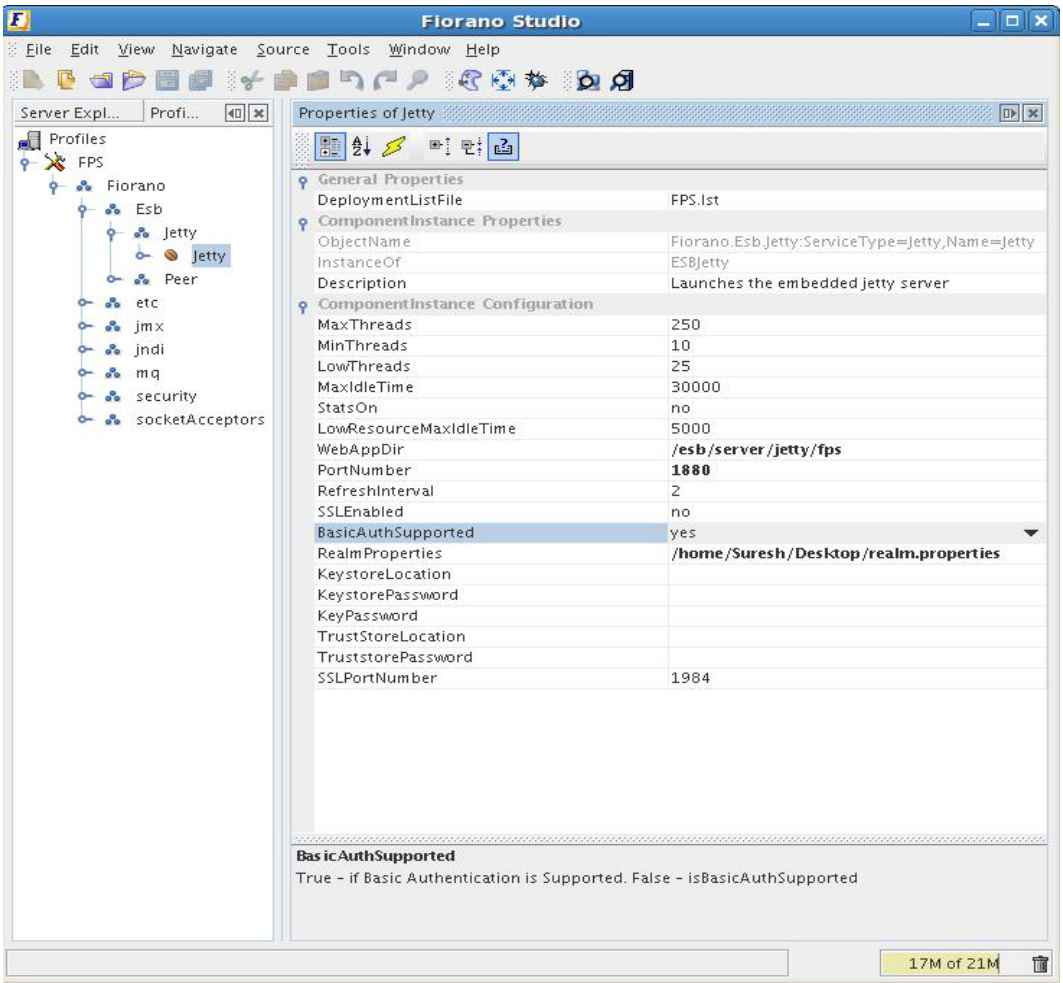


Figure 4.520: Enabling Basic Authentication in FPS Jetty server

Enabling Basic Authentication in webapp

1. Open Web.xml file located at FIORANO_HOME/esb/server/jetty/fps/webapps/bcwsgateway/WEB-INF.
2. Uncomment the security-constraint and login-config elements and save the file.
3. Delete the Peer Server cached profile from FES runtime by deleting the folder FIORANO_HOME/runtimedata/EnterpriseServers/<Profile Name>/FES/peers/<peer profile>.
4. Start the FPS server.

The basic authentication is enabled in FPS Jetty Server and in the webapp. Now the authentication can be enabled in WSStub by setting the value of Enable authentication property to yes. Provide a set of username and password which is present in Realm.properties file.

Note: To confirm the authentication, when the WSStub component is launched right-click the component and select View WDSL option. Web browser prompts the user to enter the username and password to display the WSDL.

When the authentication is enabled in WSStub, client can access the Web Service only if it uses the basic authentication with correct credentials.

UserName

Username to be used by the WSStub component while deploying the Web Service. The entry for this user name has to be present in Realm.properties file. If this user name is not present, the component will not launch.

Password

Password to be used by the WSStub component while deploying the Web Service. If this password is not present in Realm.properties file the component will not launch.

Realms

Security realms allow securing the web applications against unauthorized access. Protection is based on authentication (identifying who is requesting access to the webapp) and access control (restricting what can be accessed and how it is accessed within the webapp).

A webapp statically declares its security requirements in the web.xml file. Authentication is controlled by the <login-config> element. Access controls are specified by <security-constraint> and <security-role-ref> elements. When a request is received, the web container checks if the user performing the request is authenticated and if the user has a role assignment that permits access to the requested resource. Access to the resource is allowed only when the user is authorized and has the required permissions.

A realm has a unique name, and is composed of a set of users. Each user has authentication information (Example: a password) and a set of roles associated.

More than one realm can be configured depending on the needs.

Fiorano uses a simple realm whose authentication and authorization information is stored in a properties file. Each line in the file contains a username, a password, and 0 (zero) or more role assignments. The format is:

```
username: password[,rolename ...]
```

where:

- username is the user's unique identity
- password is the user's password
- rolename is the user's role

Sample content of Realm.properties file is shown below:

- admin: admin,admin
- user: password,user
- guest: guest,read-only

4.14.1.1.8 Miscellaneous Configuration

4.14.1.1.8.1 FES Connection Configuration

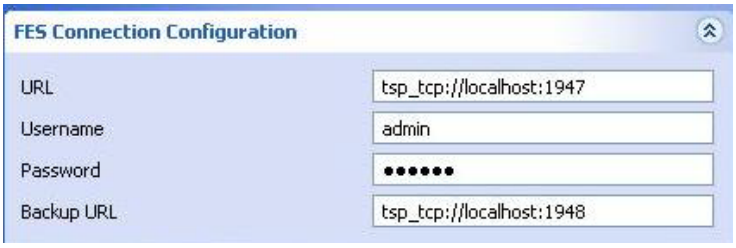


Figure 4.521: FES Connection Configuration

FES URL

The URL of Enterprise Server to which the Peer Server on which the component is running is connected.

Username

User name that should be used to connect to the enterprise server.

Password

Password that should be used to connect to the enterprise server.

FES Backup URL

The alternate URL that should be tried for connecting to the Enterprise Server if the Enterprise Server cannot be connected to using the URL mentioned against property **FES URL**.

Note: In case of Enterprise Servers in HA mode, this should point to Secondary Server URL if the primary is set against Server URL property and vice-versa.

4.14.1.1.8.2 Execution Configuration

Figure 4.522: Execution Configuration

Execution Timeout

WSStub component receives client request on its output port and sends the request message to the connected components. After the processing is completed by the connected components response message is sent to WSStub input port and from there it will be sent to the client.

WSStub component accepts the response if it reaches before the timeout period. If no response is received, on timeout Error message will be sent to the client saying No message received till the timeout period.

Note: Request will be processed by the connected components in the Event Process even after the timeout but the response is not sent back to the client by WSStub.

Execution Timeout of 0 indicates infinite timeout.

Request Timeout

This value is set as Time to live property on the request message. The request message will be discarded after the timeout.

0 indicates infinite time.

4.14.1.1.9 Additional Configuration

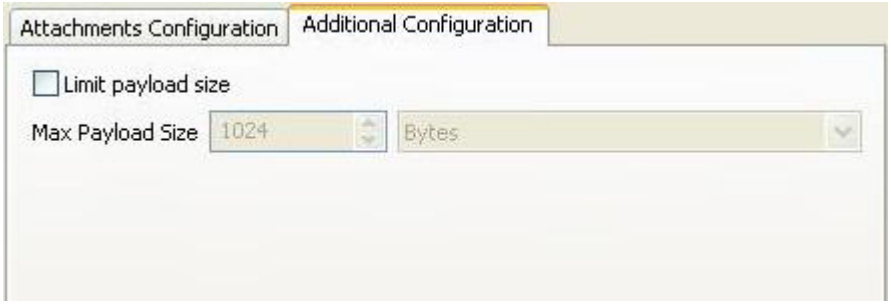


Figure: Additional Configuration

Limit payload size

Enable this checkbox to limit the payload size in server. If the request size is more than configured payload size then server throws soap fault saying soap message size is exceeded.

Max Payload Size

This property specifies the maximum payload size. If the client's request exceeds this value server will send a soap fault, if Limit Payload Size is selected.

4.14.1.1.10 Attachment Configuration

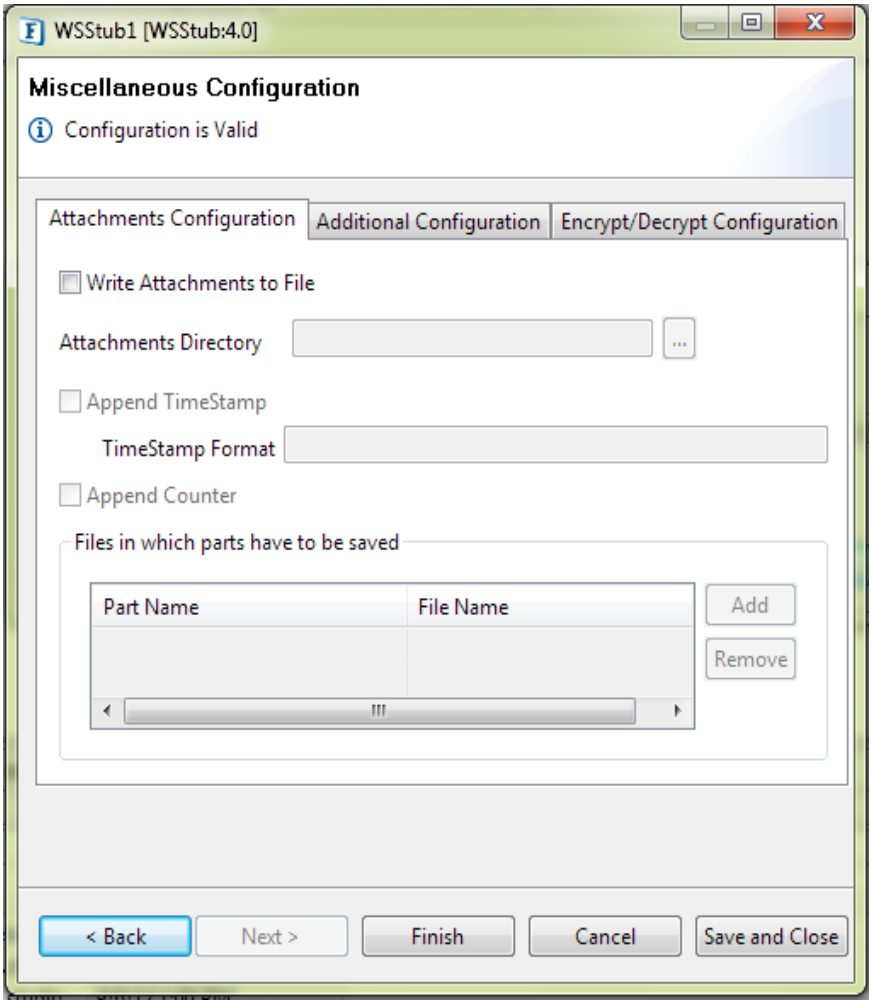


Figure: Attachments Configuration

Write Attachments to File

Select this option to write attachments to file, if not selected attachments will be sent to output port. While sending to output port, Attachments will be base64 encoded if they does not contain text content.

Attachments Directory

Provide the path of the Directory in which attachments needs to be saved.

Is Append TimeStamp

Select this option to append time stamp to the file name while writing attachments to file.

TimeStamp Format

Specify the time stamp format to be appended to the file name.

Is Append Counter

Select this option to append a counter to the file name in case a file with the same name exists. If **Is Append TimeStamp** is also enabled then time stamp will be appended before appending counter.

Example: fileName_0(with out time stamp), fileName_1313406444699_1(with time stamp).

Files in which parts have to be saved

This table will be updated automatically with the mime parts present in the wsdl. By default file name will be generated same as that of part name and can be changed if required.

Part Name

Unique name of the part in the message. Unique name will be generated by appending <SoapVersion>__<Opearation Name>__ to the partname. For example if the part Part1 in Opeartion Operation1 is a mime part, then the key populated in this table will be

```
SOAP11__Operation1__Part1 for SOAP1.1,
SOAP12__Operation__Part2 for SOAP1.2.
```

If a wsdl doesn't contain any defined mime parts and webservice need to save the attachments, then user should configure the table manually.

File Name

Specify the file name in which corresponding part message needs to be saved.

4.14.1.1.11 Encrypt/Decrypt Configuration

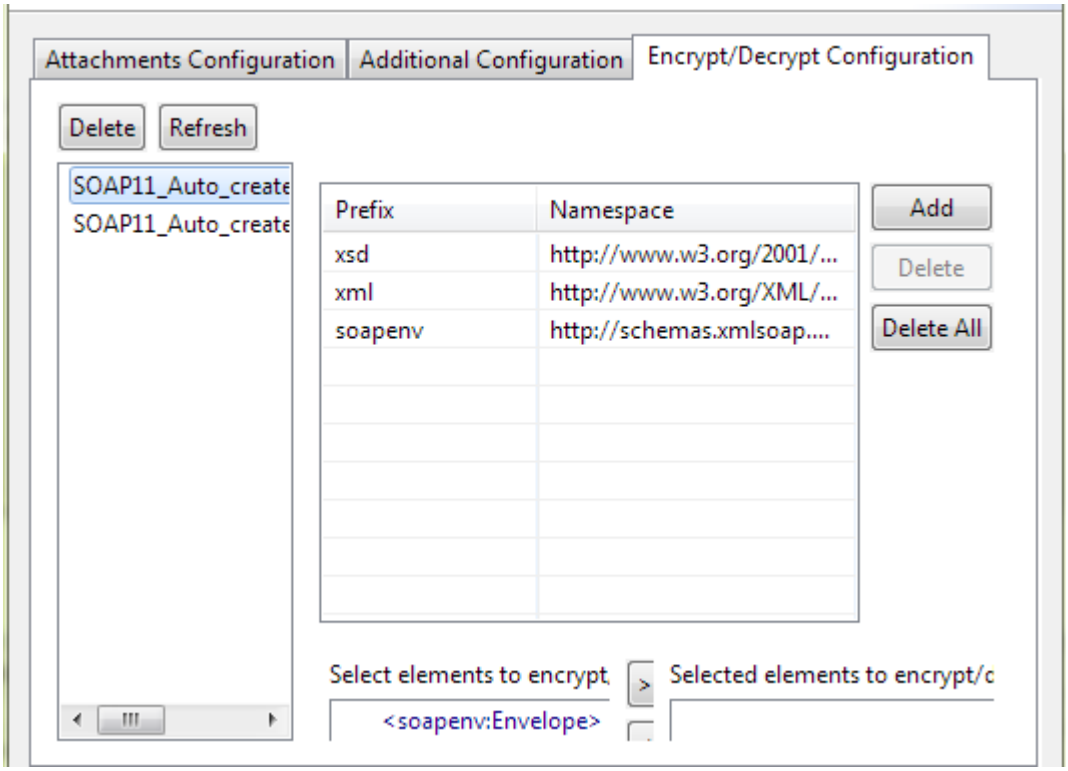


Figure: Encrypt/Decrypt Configuration

For details on this configuration please refer to Chapter 11 of Fiorano SOA User Guide.

4.14.1.1.12 Port Generation

The ports will be generated based on the configuration of WSDefinition as follows:

- For each operation which has SOAP binding defined request, response and error ports will be generated.
- The name of the ports will be prefixed by the type of SOAP binding i.e., SOAP11 or SOAP12.
- The prefix is followed by _<operation name>_ and the type of port.

Example:

For an operation with name operationName and SOAP11 binding, ports with following names will be generated:

```
Output port with name SOAP11_operationName_REQUEST
Input port with name SOAP11_operationName_RESPONSE
Input port with name SOAP11_operationName_FAILURE
```

4.14.1.1.12.1 Port Schema Structures

Request Schema

The following figure shows the sample request schema when Binding type is '**MIME MultiPartRelated**'. The mime parts will be present in the Attchments element as specified in the below figure.

Structure | Namespaces | Content

XML Item	Value Type
<soapenv:Envelope>	
<soapenv:Header>	
<soapenv:Body>	
<PartName2>	#string
<soapenv:Attachments>	
<PartName1>	
@type	#string
@savedToFile	yes | no
#string	

Figure: Request Schema

Type

Type specifies the Content type of the attachment.

SavedToFile

Whether the attachment was saved to file or not. If yes attatchment part element contains the file path to which it got saved else the entire attachment otherwise it contains the attachment content. Attachment part content will be base64 encoded if the attachment contains image or binary content.

Response Schema

The following figure shows the response schema elements when Binding type is '**MIME MultiPartRelated**'.

Structure | Namespaces | Content

XML Item	Value Type
<soapenv:Envelope>	
<soapenv:Header>	
<soapenv:Body>	
<PartName2>	#string
<soapenv:Attachments>	
<PartName1>	
@type	#string
@isURL	yes | no
#string	

Figure: Response Schema

type

Content type of the attachment.

IsURL

Specifies whether the attachment is an URL or not. If yes, attachment part element contains the URL of the attachment, otherwise the attachment content.

4.14.1.2 Functional Demonstration

Scenario 1

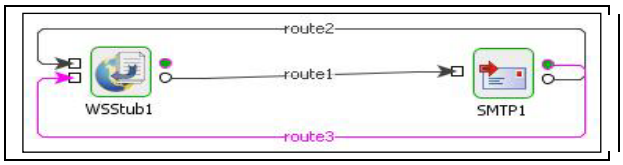


Figure 4.522: Scenario demonstration

Figure 4.522 demonstrates how WSStub can be used to deploy the Event Process as Web Service.

In the flow, WSStub is connected to an SMTP component. WSStub component sends the client requests to SMTP and gets the SMTP response which is sent back to the client. An operation is added in basic view and its input part type is set as root element of SMTP component's input port schema, output part type is set as root element of SMTP component's output port schema and a fault part is added and its part type is set as root element of error port schema of SMTP component.

When the Event Process is launched, it gets deployed as Web Service with service name and added operation name specified in the CPS. On right-clicking and selecting view wsdl, the user can view the wsdl for this WSStub in a web browser.

A WebserviceConsumer component can be configured to invoke this Web Service by providing the URL of this wsdl (in the Managed Connection Factory panel of WS consumer's CPS) and by selecting the Web Service Operation (In Interaction Configuration Panel).

Sample request and response to WebserviceConsumer component which accesses the deployed Web Service is shown below.

```
<ns1:Envelope xmlns:ns1="http://phanik:1880/bcwsgateway/services/simple/simpleOp
   <ns1:Header/>
   <ns1:Body>
      <ns2:Email xmlns:ns2="http://www.fiorano.com/fesb/activity/SMTP1/smtp/in">
         <To>ayrton@fiorano.com</To>
         <From>ayrton@fiorano.com</From>
         <Subject>WSStub test</Subject>
         <Body>
            <HtmlBody>Sample message</HtmlBody>
         </Body>
      </ns2:Email>
   </ns1:Body>
```

Figure 4.524: Sample Request

```
<?xml version="1.0" encoding="UTF-8"?><tnssimpleop:Envelope xmlns:tnssimpleo

<tnssimpleop:Header/>
<tnssimpleop:Body>
   <ns1:SMTP mlns:ns1="http://www.fiorano.com/fesb/activity/SMTP1/smtp/out">
<ns1:Result>Email sent successfully</ns1:Result>
</ns1:SMTP>
</tnssimpleop:Body>
</tnssimpleop:Envelope>
```

Figure 4.525: Sample Response

Scenario 2

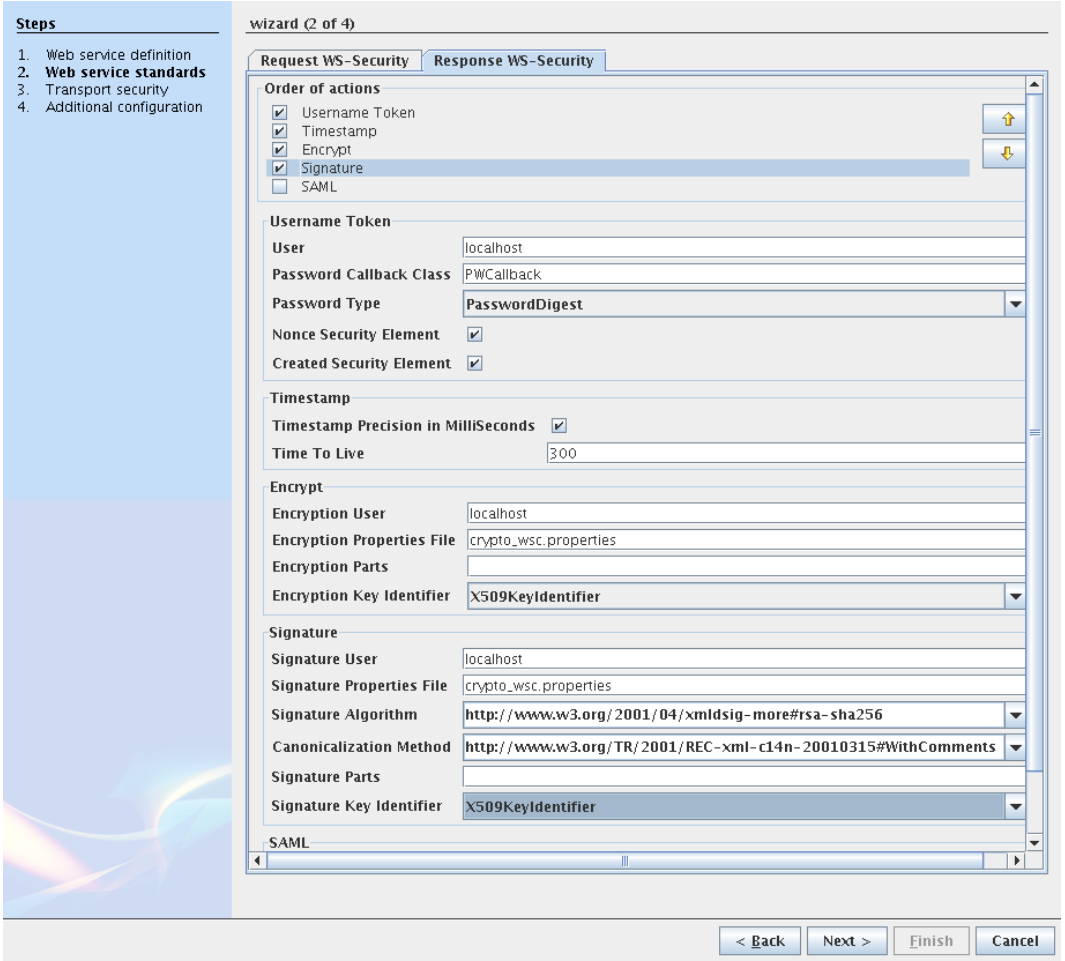


Figure: Sample WS-Security - Response configuration

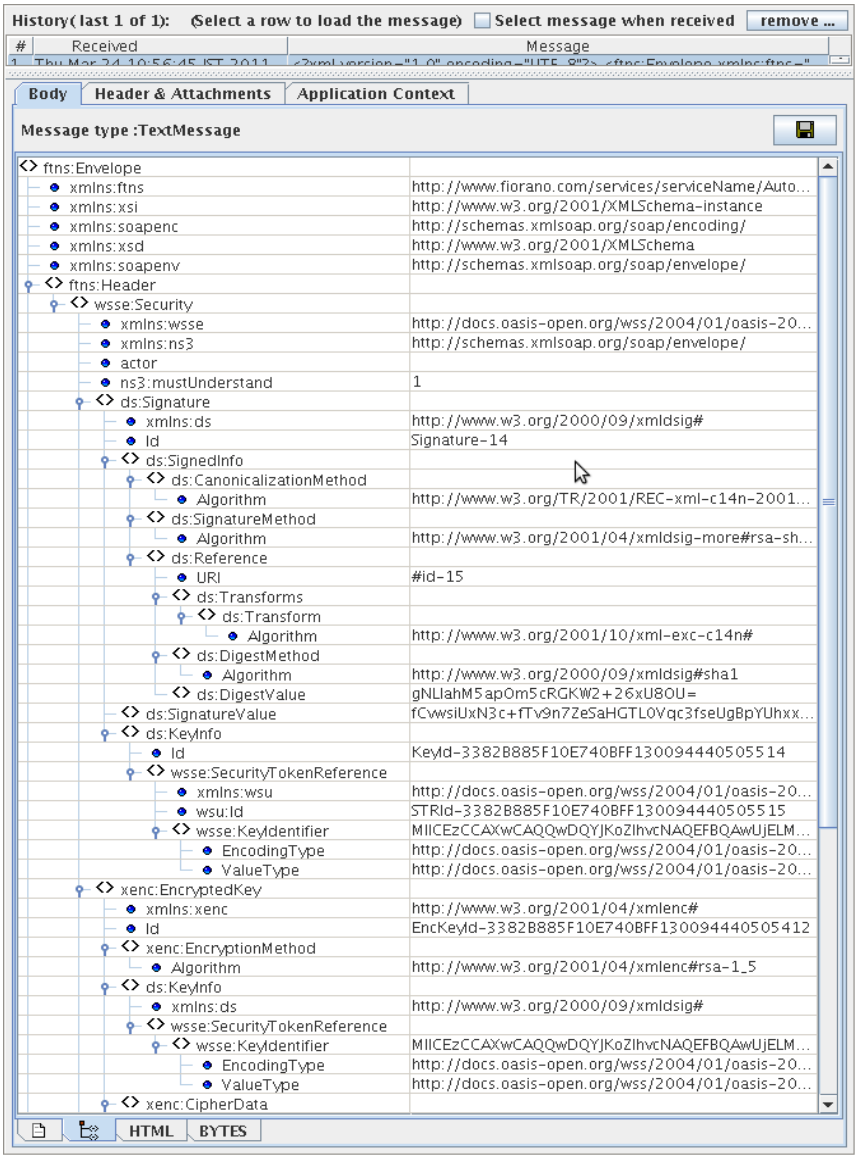


Figure: Sample Output in case of above configuration

4.14.1.3 Useful Tips

- For SSL to work properly the JDK version used should be greater than 1.5.0_08 and the jars present in JDK_HOME/jre/lib/ext should be copied to $FIORANO_HOME/esb/lib/ext.
- When WSStub component is configured to launch on HA (High Availability) Peer Servers
    - If both Primary and Secondary Servers are on the same machine

Initially, if WSStub is launched on the Primary Server, the generated WSDL URL contains Primary Server's jetty port number. In case of Primary Server failover, the Secondary Server becomes Active and relaunches the component. WSDL will be regenerated and if the Secondary Server uses a different jetty port then the WSDL URL is changed. The clients have to be reconfigured to use new URL in this case.

To avoid this situation, it is recommended to use same jetty ports for both Primary and Secondary Peer Servers.

Jetty service will be started only after the server is started successfully. In case of HA, only one server will be active at a given time and the Jetty Server will be running only in the active server and there will be no bind exceptions even if both the servers use same port number for Jetty.

- If both Primary and Secondary servers are on different machines

  In this case if the failover happens the hostname/IP address in the WSDL URL will be changed. So the clients have to be reconfigured accordingly.

  HTTP headers are received from the gateway as message properties with the header name prefixed with http_. For example, http_Content-Type.

  The service name must be unique for WSStub instances configured on same peer server.

- Sending Empty HTTP Response with 202 Status Code: WSStub Component can be used for an asynchronous request pattern where a HTTP Response with status code 202 and empty content is returned to inform the client that their invocation has been received. To achieve this, we need to send a JMS message to the response port of the WSStub with the string property http_statusCode set to 202.
- Enabling SOAP Compression in WSStub
  - Open Web.xml file located at FIORANO_HOME/esb/server/jetty/fps/webapps/bcwsgateway/WEB-INF.
  - Uncomment the filter and filter-mapping elements and save the file.
  - Copy the jars present at FIORANO_HOME/extlib/soapCompression to FIORANO_HOME/esb/server/jetty/fps/webapps/bcwsgateway/WEB-INF/lib.
  - Delete the Peer Server cached profile from FES runtime by deleting the folder FIORANO_HOME/runtimedata/EnterpriseServers/<Profile Name>/FES/peers/<peer profile>.
  - Start the FPS server.
- WSStub instances that are configured previously can be used as they were without opening the CPS. However, if the WSStub instance is reconfigured then there will be following changes
  - Port names are changed

    This would result in the routes leading to and from the component to disappear on closing the CPS.

- Schemas on ports are changed

  This would mean that the already defined transformations are no longer valid and will have to be redefined.

So it is recommended to save the transformations defined on the route externally and redefine the transformations after the component is configured.

- Web Service name changed
  - The webservice name will be the context name appended with suffix "Service" for the old WSStub components. If we deploy the same WSStub with out any configuration changes, then the web service name will be context name itself. No suffix will be appended. This will result to reconfigure the Web service consumer components configured to use this service should be reconfigured to fetch the service name.

4.14.2 Web Service Consumer (4.0)

This component invokes a web service (usually externally hosted on a third-party system) based on the configured WSDL.

Unlike most static web service client options (like Axis wsdl2java) which generates client stub code to invoke a given WSDL, this component employs a dynamic invocation mechanism to ensure that no code needs be written or deployed for invoking a component.

The incoming request parameters are automatically wrapped in a SOAP request packet (handling different types of SOAP headers for handling web service security, transactions and so on.) and sent to underlying transport (usually the response is sent back to the client

Points to note

- If the web service is secured using basic authentication, then the details of the basic authentication should be provided in the Call Properties property during execution time.
- When using WS-Security, the Password Callback class should be the fully qualified name of the class.
- The orders in which the WS-Security tokens are specified are important and should be the order in which they are specified at the web service.
- This component supports only WSDL files which are compliant to WS-I Basic Profile 1.0.
- To pass http headers to the web service, the input message should contain properties with the header name prefixed with http_. **Example**: http_Content-Type.

4.14.2.1 Connection Configuration

The web service connection details can be configured in the **Connection Properties** panel of CPS.

Attributes	
Use Operation Details From Input	no
Use Connection Details From Input	no
WSDL	http://www.webservicex.net/CurrencyConvertor.asmx?wsdl
HTTP Basic Authentication	**yes**
HTTP Username	user
HTTP Password	******
Connection Pool Params	Click here to edit...
Proxy Settings	Click ... to edit
SSL Security	**Click ... to edit**

Figure 4.526: Sample web service connection configuration

- **Operation Details From Input:** If the property "Use Operation Details From Input" is set to true, then the properties Use Connection Details From Input, WSDL, HTTP Basic Authentication, HTTP Username, HTTPPassword will be disabled in Managed Connection Factory Panel, and the properties WSDL Service, WSDL Port, End Point Address, WSDL Operation, Input Parameter, Output Parameter, SOAPBody Namespace will be disabled in Interaction Configurations panel. The required properties need to be provided in the input request. The sample input and output messages are shown in Scenario 2 of Functional Demonstration section.
- **Use Connection Details From Input**: If this property is set to true, then the "Endpoint Address" can be specified from the input also.
- **WSDL**: This property specifies the WSDL location. WSDL can be specified in the form of URL or File or it can be selected from UDDI regestry
- **HTTP Basic Authentication**: This property specifies whether http authentication is required or not to connect to web server.
- **HTTP Username**: User Name to connect to the web server.
- **HTTP Password**: Password to connect to the web server.

Server connection can be tested from within the CPS by clicking on **Test** in the connection properties panel.

Result of Operation (java.lang.String)

XML

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:soap="http://schema
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://www.webserviceX.NET/">
      <s:element name="ConversionRate">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="FromCurrency" type="tns:Currency" />
            <s:element minOccurs="1" maxOccurs="1" name="ToCurrency" type="tns:Currency" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:simpleType name="Currency">
        <s:restriction base="s:string">
          <s:enumeration value="AFA" />
          <s:enumeration value="ALL" />
          <s:enumeration value="DZD" />
          <s:enumeration value="ARS" />
          <s:enumeration value="AWG" />
          <s:enumeration value="AUD" />
          <s:enumeration value="BSD" />
          <s:enumeration value="BHD" />
          <s:enumeration value="BDT" />
```

Figure 4.527: Sample connection test result showing the WSDL loaded

The service operation can be chosen in the Interaction panel

General	
WSDL Service	{http://www.webserviceX.NET/}CurrencyConvertor
WSDL Port	CurrencyConvertorSoap
Endpoint Address	http://www.webservicex.net/CurrencyConvertor.asmx
WebService Operation	ConversionRate
Input Parameter	input
Output Parameter	output
Validate Input	no
Cleanup resources (excluding co...	yes
Target Namespace	http://www.webserviceX.NET/
Monitoring configuration	disabled: 5000 milli seconds

General Properties

- **WSDL Service**: This property specifies the WSDL Service name. Once the WSDL Opration is selected, this properety will be automatically populated
- **WSDL Port**: This property specifies the WSDL Port. Once the WSDL Opration is selected, this properety will be automatically populated
- **Endpoint Address**: This property specifies the Endpoint Address for the Web service opeartion. Once the WSDL Opration is selected, this properety will be automatically populated

- **WSDL Operation**: This property specifies the Web service opeartion. The wsdl operations specified in the WSDL provided in Managed Connection Factory panel will be shown by clicking the eclipse. Once the operation gets selected, the properties WSDL Service, WSDL Port, Endpoint Address, Input parameter and Output parameter will be populated by the respective values.
- **Input Parameter**: This property specifies the input message name for the selected WSDL Operation. Once the WSDL Opration is selected, this properety will be automatically populated
- **Output Parameter**: This property specifies the output message name for the selected WSDL Operation. Once the WSDL Opration is selected, this properety will be automatically populated

Call and Addressing

- **SOAPBody Namespace**: This property specifies the SOAP Body namespace. Once the WSDL Opration is selected, this properety will be automatically populated
- **Call Properties**: Advanced properties which can be used to optimize and change the behavior SOAP Invocation Call. The description for the axis call properties can be find at http://ws.apache.org/axis/java/apiDocs/org/apache/axis/client/Call.html
- **Enable WS-Adressing**: If this property is set, it enables the support for WS-Addressing headers. The input and and output schema contains WS-Addressing headers.

⊟ General	
WebService Operation	getRate
⊟ Call and Addressing	
Call Properties	click ... to see the details.
Enable WS-Addressing	no
⊟ Security - Request	
UsernameToken WS-Security (Request)	no
Order of UsernameToken (Request)	1
User	null
Password Callback class (Request)	null
Password type	PasswordText
Nonce Security element	no
Created Security element	no
Timestamp WS-Security (Request)	no
Precision in Milliseconds (Request)	no
Order of Timestamp (Request)	2
Encryption WS-Security (Request)	no
Order of Encrypt (Request)	3
Encryption User	null
Encryption Properties filename (Request)	null
Encryption Parts	null
Encryption Key Identifier	null
Signature WS-Security (Request)	no
Order of Signature (Request)	4
Signature Properties filename (Request)	null
Signature Parts	null
Signature Key Identifier	null
SAML WS-Security (Request)	no
Order of SAML (Request)	5
Signed SAML (Request)	no
SAML Properties filename	null
⊞ Security - Response	
⊟ Reliable Messaging	
Client port of WS-ReliableMessaging	9090
Enable WS-ReliableMessaging	no
⊟ Soap Compression	
Compress SOAP Request	no
Compress SOAP Response	no

Figure 4.528: The web service operation to invoke and other ws-* standards to use along with the request

Security - Request Properties

UsernameToken WS-Security (Request): If the web service performs UsernameToken identification for the request, then this property should be enabled. Username and password values are added to the message headers.

Order of UsernameToken (Request): Determines the order of the UsernameToken security function. The order of a security function determines when this function will be applied when multiple security functions are being used.

User: This property is used as username for the UsernameToken security function. It is also used as the alias name in the keystore to get user's certificate or private key to perform signing for the Signature security function. It is also used as the fallback for the encryption security function in case of "Encryption User" is null and "Encryption WS-Security (Request)" is set to yes.

Password Callback class (Request): This is needed by the security functions to get the password and to verify the username/password pair. The password callback class should implement org.apache.ws.security.WSPasswordCallback class. This Password Callback class should be the fully qualified name of the class. The jar which contains the password callback class should be added as a resource to the component.

Password type: The Password type can be either PasswordText or PasswordDigest.

- **PasswordText**: Password is sent in raw text format with in the security header of the soap request.
- **PasswordDigest**: Password is sent in digest format with in the security header of the soap request.
- **PasswordNone:** No password will be send in the security header. This option is useful when user wants to specify the username without any password.

Nonce Security element: Specifies whether to use nonce element in the security header or not. When UsernameToken security function is used, then nonce security element can be employed to prevent message replay attacks. A nonce is a random value that the client creates to include in each UsernameToken that it sends. Although using a nonce is an effective countermeasure against replay attacks, it requires server to maintain a cache of used nonces, consuming server resources.

Created Security element: Specifies whether to use Created element in the security header or not. This element denotes the time of creation of a nonce. Combining a nonce with a creation timestamp has the advantage of allowing a server to limit the cache of nonces to a "freshness" time period, establishing an upper bound on resource requirements.

Timestamp WS-Security (Request): If this property is set, timestamp will be added as security header in the soap request. In this case, the message is valid up to 5 minutes or 300 seconds after the creation of the message.

Precision in Milliseconds (Request): If this is set, timestamps will have precision in milliseconds. Otherwise it will be seconds.

Time To Live: The time difference between creation and expiry time in the WSS Timestamp. This should be specified in seconds.

Order of Timestamp (Request): Specifies the order of the Timestamp security function. The order of a security function determines when this function will be applied when multiple security functions are being used.

Encryption WS-Security (Request): This property can be set to perform encryption on the entire soap message or some parts of the soap message.

Order of Encrypt (Request): Specifies the order of the Encrypt security function. The order of a security function determines when this function will be applied when multiple security functions are being used.

Encryption User: Username for the encryption function. The encryption function uses the public key of this user's certificate. If this parameter is not set, then the encryption function falls back to the "User" parameter to get the certificate. The encrypt function will not authenticate user. So there is no need to set any password call back class for encrypt.

Encryption Properties filename (Request): The name of the crypto property file to use for SOAP Encryption. If this parameter is not specified and if both "Signature Properties filename (Request)" and "Signature WS-Security (Request)" are set, then the encryption function uses signature property file. Otherwise the handler throws an AxisFault.

The sample properties file content:

```
org.apache.ws.security.crypto.provider=org.apache.ws.security.components.crypto.Merlin
org.apache.ws.security.crypto.merlin.file=C:\\Documents and
Settings\\Fiorano\\Desktop\\fiorano.ks
org.apache.ws.security.crypto.merlin.keystore.type=jks
org.apache.ws.security.crypto.merlin.keystore.password=fioranopass
org.apache.ws.security.crypto.merlin.keystore.alias=fiorano
org.apache.ws.security.crypto.merlin.alias.password=fioranopass
```

The properties descriptions:

org.apache.ws.security.crypto.provider: implementation class for security provider. Fiorano internally uses bouncycastle, to use the same this property must be set to "org.apache.ws.security.components.crypto.Merlin". To use other providers, the provider jar has to be added as a resource for this component and fully qualified name of appropriate provider class has to be set for this property.

- **org.apache.ws.security.crypto.merlin.file**: The path to the keystore file.
- **org.apache.ws.security.crypto.merlin.keystore.type**: The keystore type, for example JKS for the Java key store. Other keystore type, such as pkcs12 are also possible but depend on the actual Crypto implementation.
- **org.apache.ws.security.crypto.merlin.keystore.password**: The password to read the keystore. If this property is not set, then the pwcallbackproperty must be defined.
- **org.apache.ws.security.crypto.merlin.keystore.alias**: The alias name under which the private key/certificate stored in keystore
- **org.apache.ws.security.crypto.merlin.alias.password**: Password for private key/certificate inside keystore stored under given alias (not used for encryption)

Encryption Parts: Parameter specifies which parts of the request shall be encrypted. The value of this parameter is a list of semi-colon separated element names that identify the elements to encrypt. An encryption mode specifier and a namespace identification, each inside a pair of curly brackets, may preceed each element name. The encryption mode specifier is either {Content} or {Element}. 'Element' encryption mode will encrypt the entire element including start and end tags. 'Content' encrypt mode will encrypt only the content of the specifioed element. The default encryption mode is 'Content'.
For example if we set "{Element}{http://example.org/paymentv2}CreditCard;{}{}UserName" list to this property, then the first entry of the list identifies the element CreditCard in the namespace http://example.org/paymentv2, and will encrypt the entire element. In the second entry the encryption modifier and the namespace identifier are omitted. In this case the encryption mode defaults to Content and the namespace is set to the SOAP namespace.
The element name, the namespace identifier, and the encryption modifier are case sensitive. To specify an element without a namespace use the string Null as the namespace name (this is a case sensitive string) If no list is specified, the handler encrypts the SOAP Body in Content mode by default.

Encryption Key Identifier: Select the key identifier type to use.

- **DirectReference**: The security function takes the signing certificate, converts it to a BinarySecurityToken, and then place it in the security header. Thus the whole signing certificate is transfered.
- **X509KeyIdentifier**: The encryption method uses the public key associated with this certificate to encrypt the symmetric key used to encrypt data. The certificate is converted into a KeyIdentfier token and sent to the server. Thus the complete certificate data is transfered.
- **SKIKeyIdentifier**: The security function uses SKIKeyIdentifier.
- **IssuerSerial**: The encryption method uses the public key associated with this certificate to encrypt the symmetric key used to encrypt data. The issuer name and the serial number of the signing certificate are sent to the server.

Signature WS-Security (Request): If this security function is selected the digest of the message is created and encrypted before sending. The property "User" must be specified to get the private key/certificate of respective user from the keystore for signing.

Order of Signature (Request): Specifies the order of the Signature security function. The order of a security function determines when this function will be applied when multiple security functions are being used.

Signature User: This name is used as the alias name in the keystore to get user's certificate and private key to perform signing. If this parameter is not set, then the signature function falls back to the "User" parameter to get the certificate. Password for the user to get certificates from the keystore should be provided in the Password Callback class.

Signature Properties filename (Request): The name of the crypto property file to use for SOAP Signature. Please see the description of "Encryption Properties filename" for the details of the properties file.

Signature Parts: Parameter specifies which parts of the request shall be signed. Please see the description of "Encryption Parts" for the syntax.

Signature Key Identifier: Select the key identifier type to use. Please see the description of "Encryption Key Identifier" for the descriptions of key identifiers.

SAML WS-Security (Request): Select this property to perform SAML Token Identification.

Order of SAML (Request): Specifies the order of the SAML security function. The order of a security function determines when this function will be applied when multiple security functions are being used.

Signed SAML (Request): Specifies whether to use signed SAML or unsigned SAML. If Signed SAML is used, then client performs two actions inserting a SAML Token (unsigned) and an associated Signature. So define both the actions SAML Unsigned and Signature at server to resolve these security headers. If Signed SAML is used, the signature properties should be specified with out selecting the property "Signature WS-Security (Request)".

SAML Properties filename (Request): The name of the SAML properties file. This file should be added as resource to the component.

The example properties file content:

```
org.apache.ws.security.saml.issuerClass=org.apache.ws.security.saml.SAMLIssuerImpl
org.apache.ws.security.saml.issuer.cryptoProp.file=crypto_wsc.properties
org.apache.ws.security.saml.issuer.key.name=fiorano
org.apache.ws.security.saml.issuer.key.password=fioranopass
org.apache.ws.security.saml.issuer=fiorano
org.apache.ws.security.saml.subjectNameId.name=uid=mule,ou=people,ou=saml-
demo,o=example.com
org.apache.ws.security.saml.subjectNameId.qualifier=www.example.com
org.apache.ws.security.saml.authenticationMethod=password
#org.apache.ws.security.saml.confirmationMethod=senderVouches
org.apache.ws.security.saml.confirmationMethod=keyHolder
```

Security - response Properties

- **Ignore Order** If this is set, Order of Security actions will be ignored for the in coming response.
- **UsernameToken WS-Security (response)**: Determines whether the response from the server contains Username token headers or not.
- **Order of UsernameToken (response)**: Determines the order of the Username Token security function. The order of a security function determines when this function will be applied when multiple security functions are being used.
- **Password Callback class (response)**: This is needed by the security functions to get the password and to verify the username/password pair. The password callback class should implement org.apache.ws.security.WSPasswordCallback class. This Password Callback class should be the fully qualified name of the class. The jar which contains the password callback class must be added as a resource to the component.
- **Is Password Required:** This property should set to false if the Username security token is used without a password. No need to provide Password callback class if this property is set to no.
- **Timestamp WS-Security (response)**: Specifies whether the soap response contains timestamp headers or not.

- **Precision in Milliseconds (response)**: If this is set, timestamps will have precision in milliseconds. Otherwise it will be seconds.
- **Order of Timestamp (response)**: Specifies the order of the Timestamp security function. The order of a security function determines when this function will be applied when multiple security functions are being used.
- **Encryption WS-Security (response)**: Specifies whether the soap response or some parts of the soap response are encrypted or not. If this property is set then the client validates the user, so password callback class should be specified.
- **Order of Encrypt (response)**: Specifies the order of the encrypt security function. The order of a security function determines when this function will be applied when multiple security functions are being used.
- **Encryption Properties filename (response)**: The name of the crypto property file to use for decryption of the soap response. If this parameter is not specified and if both the "Signature Properties filename (response)" and "Signature WS-Security (response)" are set to yes, then the decryption function uses signature property file. Otherwise the handler throws an AxisFault. Please see the description of "Encryption Properties filename (Request)" for the details of the crypto properties file.
- **Signature WS-Security (response)**: Specifies whether the soap response or some parts of the soap response are signed or not.
- **Order of Signature (response)**: Specifies the order of the Signature security function. The order of a security function determines when this function will be applied when multiple security functions are being used.
- **Signature Properties filename (response)**: The name of the crypto property file to use for SOAP Signature. Please see the description of "Encryption Properties filename (Request)" for the details of the properties file.
- **SAML WS-Security (response)**: Specifies whether the soap response uses SAML Token Identification or not.
- **Order of SAML (response)**: Specifies the order of the SAML security function. The order of a security function determines when this function will be applied when multiple security functions are being used.
- **Signed SAML (response)**: If Signed SAML is used at server side for response, then the server performs two actions inserting a SAML Token (unsigned) and an associated Signature. So define both the actions SAML Unsigned and Signature here to resolve these security headers.

4.14.2.1.1 Using Named Configurations

Changes are done in eStudio w.r.t to Named Configurations. Proxy Settings and SSL Configuration Panels are modified to use Named Configurations.

The configuration can be tested when you click on the **Test** option in the interaction properties panel.

Figure 4.529: Sample input

Figure 4.530: Sample output

4.14.2.2 Input and Output

4.14.2.2.1 Input Schema

The input schema is auto generated based on the configuration provided.

When the property "Use Operation Details From Input" is set to true, we should provide the WSDL service, operation details in the input. If this property set to true, the input schema would be

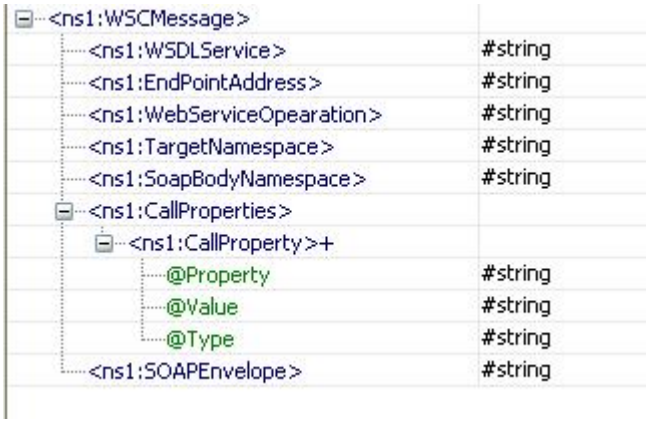


Figure: Input schema with Operation properties

When the property "Use Connection Details From Input" is chosen, an additional element ConnectionProperties is added to the input schema, as shown in the figure. We can provide End Point Address under this element.

XML Item	Value Type
<ns1:Envelope>	
<ConnectionProperties>?	
<EndPointAddress>?	#string
<ns1:Header>	
<ns1:Body>	
<ns2:ConversionRate>	
<ns2:FromCurrency>	AFA | ALL | DZD | ARS | AWG | AUD | B...
<ns2:ToCurrency>	AFA | ALL | DZD | ARS | AWG | AUD | B...

Figure: Input schema with Connectionproperties

4.14.2.2.2 Output Schema

The output schema is auto generated based on the configuration provided. For the configuration shown above, the schema would be

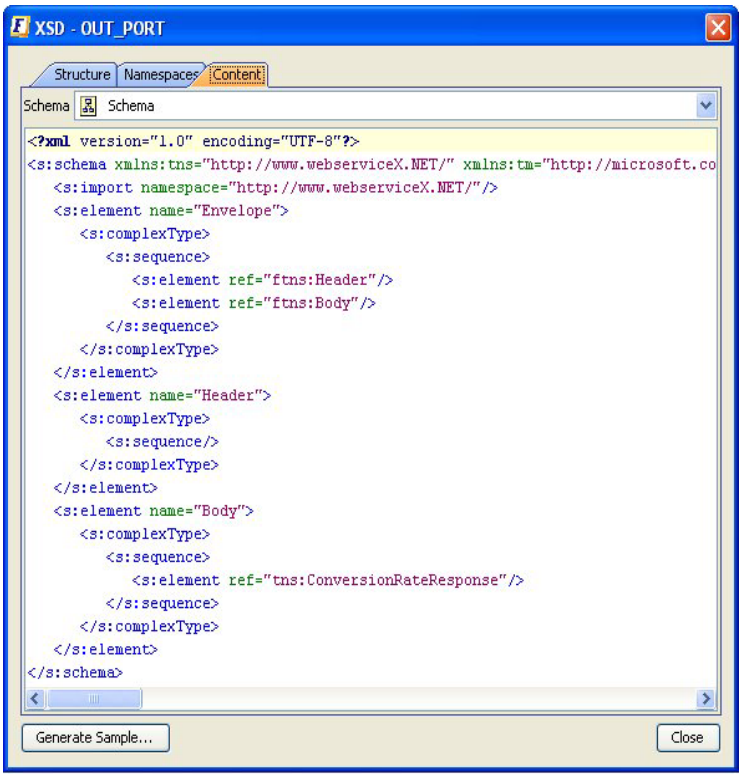


Figure 4.532: Output Schema Content

4.14.2.3 Accessing Share Point Web Services

Sample configuration to access Share Point Web Services Using Web Service Consumer is explained below:

1. Used wsdl is present at http://www.wssdemo.com/_vti_bin/lists.asmx?wsdl.
2. To access sharepoint webservices, you need the authentication details of the share point webserver.In the MCF Panel, HTTP Basic Authentication should be set to **Yes** and HTTPUsername and HTTPPassword should be provided.Sample username "**demouser**" and password "**Templates**" is used in the below configuration.

Attributes	
Use Operation Details From Input	no
Use Connection Details From Input	no
WSDL	http://www.webservicex.net/CurrencyConvertor.asmx?wsdl
HTTP Basic Authentication	**yes**
HTTP Username	user
HTTP Password	******
Connection Pool Params	Click here to edit...
Proxy Settings	Click ... to edit
SSL Security	**Click ... to edit**

Figure 4.533: web service connection configuration to access share point web services

3. In Interaction Configurations panel, click on the ellipsis button against "WebService Operation" property to select the WebService operation as shown below:

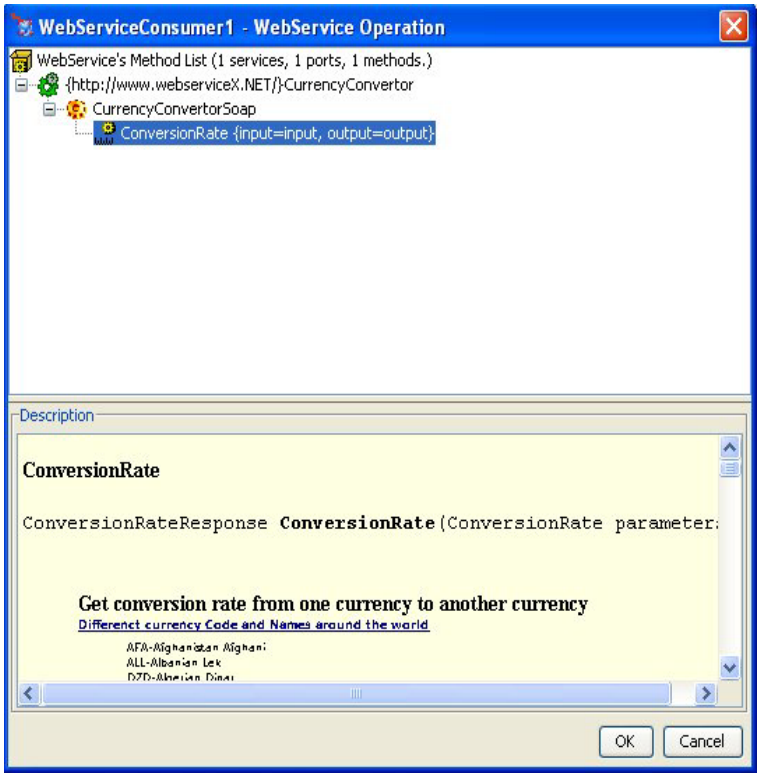


Figure 4.534: selecting the WebService operation

4. After selecting the operation, click on ellipsis button against the **CALL Properties** property to launch the **Advanced Properties** dialog to add the username and password properties.
5. To add the Username, click on add button, select javax.xml.rpc.security.auth.username and give value as **demouser**. To add a password, click on **Add** button, select javax.xml.rpc.security.auth.password and give value as **Templates**. The properties provided here is set on the SOAP Invocation Call.

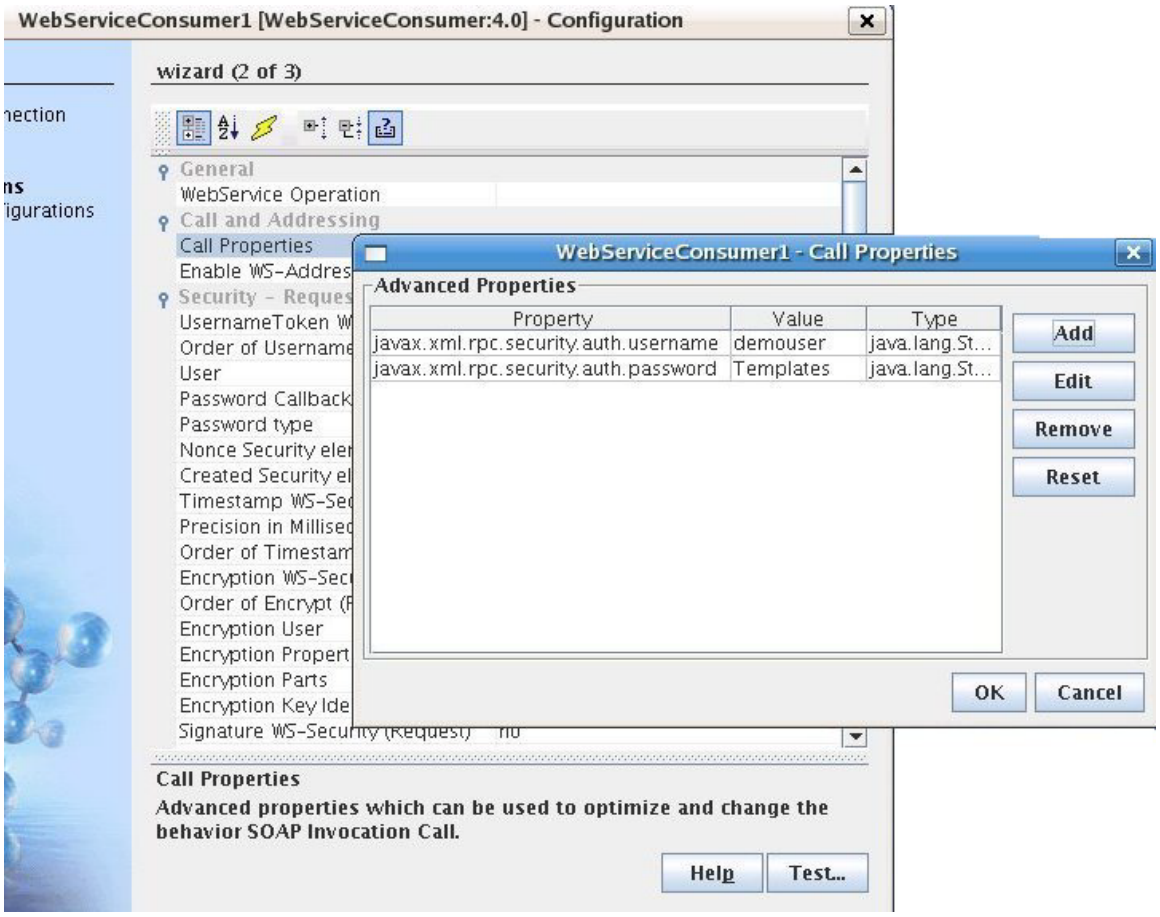


Figure 4.535: WebServiceConsumer – Call Properties

6. To test the configuration, click test -> **execute** button. The test returns output as shown in Figure 4.536.

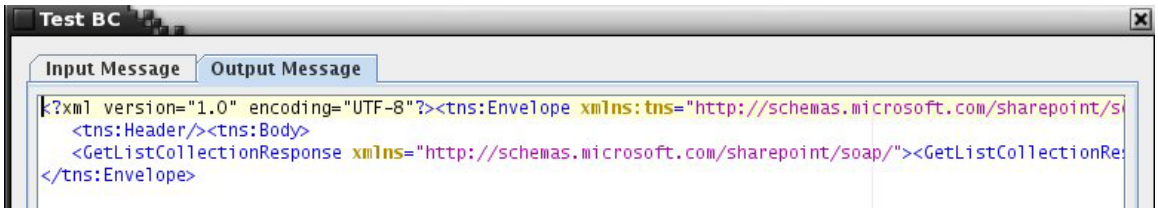


Figure 4.536: Output Message

4.14.2.4 Functional Demonstration

4.14.2.4.1 Scenario 1

Invoking a web service operation using a WSDL from the following URL
http://www.webservicex.net/CurrencyConvertor.asmx?WSDL

Configure the Web Service Consumer component as described in *Configuration and Testing* section and use feeder and display component to send sample input and check the response respectively.

```
<ns1:Envelope xmlns:ns1="http://www.webserviceX.NET/ConversionRate/Input">
   <ns1:Header/>
   <ns1:Body>
      <ns2:ConversionRate xmlns:ns2="http://www.webserviceX.NET/">
         <ns2:FromCurrency>NAD</ns2:FromCurrency>
         <ns2:ToCurrency>WST</ns2:ToCurrency>
      </ns2:ConversionRate>
   </ns1:Body>
</ns1:Envelope>
```

Figure 4.538: Demonstrating Scenario 1 with sample input

Sample Input

```
<?xml version="1.0" encoding="UTF-8"?><tns:Envelope xmlns:tns="http://www.webserviceX.NET/ConversionRate/Output" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   <tns:Header/><tns:Body>
   <ConversionRateResponse xmlns="http://www.webserviceX.NET/"><ConversionRateResult>0.295</ConversionRateResult></ConversionRateResponse></tns:Body>
</tns:Envelope>
```

Sample Output

Figure 4.539: Demonstrating Scenario 1 with sample input and output

4.14.2.4.2 Scenario 2

Invoking a web service operation from the following URL, by setting the property "Use Operation Details From Input" to true.

http://www.webservicex.net/CurrencyConvertor.asmx?WSDL

Configure the Web Service Consumer component as described in Chapter 2 and use feeder and display component to send sample input and check the response respectively.

Sample Input

Input Message | Output Message

```
<nsl:WSCMessage xmlns:ns1="http://www.fiorano.com/fesb/activity/WSInvoker/">
   <nsl:WSDLService>CurrencyConvertor</nsl:WSDLService>
   <nsl:EndPointAddress>http://www.webservicex.net/CurrencyConvertor.asmx</nsl:EndPointAddress>
   <nsl:WebServiceOpearation>ConversionRate</nsl:WebServiceOpearation>
   <nsl:TargetNamespace>http://www.webserviceX.NET/</nsl:TargetNamespace>
   <nsl:SoapBodyNamespace>http://schemas.xmlsoap.org/wsdl/soap/</nsl:SoapBodyNamespace>
   <nsl:CallProperties>
      <nsl:CallProperty Property="TargetNamespace.Prefix" Value="tns" Type="java.lang.String"/>
      <nsl:CallProperty Property="javax.xml.rpc.soap.operation.style" Value="document" Type="java.lang.String"/>
      <nsl:CallProperty Property="SOAP.version" Value="http://schemas.xmlsoap.org/wsdl/soap/" Type="java.lang.String"/>
      <nsl:CallProperty Property="javax.xml.rpc.soap.http.soapaction.uri" Value="http://www.webserviceX.NET/ConversionRate"
                                                                                Type="java.lang.String"/>
   </nsl:CallProperties>
   <nsl:SOAPEnvelope><![CDATA[<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
                                             xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <soap:Body>
   <ns2:ConversionRate xmlns:ns2="http://www.webserviceX.NET/">
         <ns2:FromCurrency>EGP</ns2:FromCurrency>
         <ns2:ToCurrency>TND</ns2:ToCurrency>
      </ns2:ConversionRate>
          </soap:Body></soap:Envelope>]]></nsl:SOAPEnvelope>
</nsl:WSCMessage>
```

Sample Output

Input Message | Output Message

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
<ConversionRateResponse xmlns="http://www.webserviceX.NET/">
    <ConversionRateResult>0.2536</ConversionRateResult>
</ConversionRateResponse></soap:Body></soap:Envelope>
```

Figure 540: Demonstrating Scenario 2 with sample input and output

4.14.2.4.3 Scenario 3

Using WS-Security

Configure the Web Service Consumer component as described in Security - Request and Response sections.

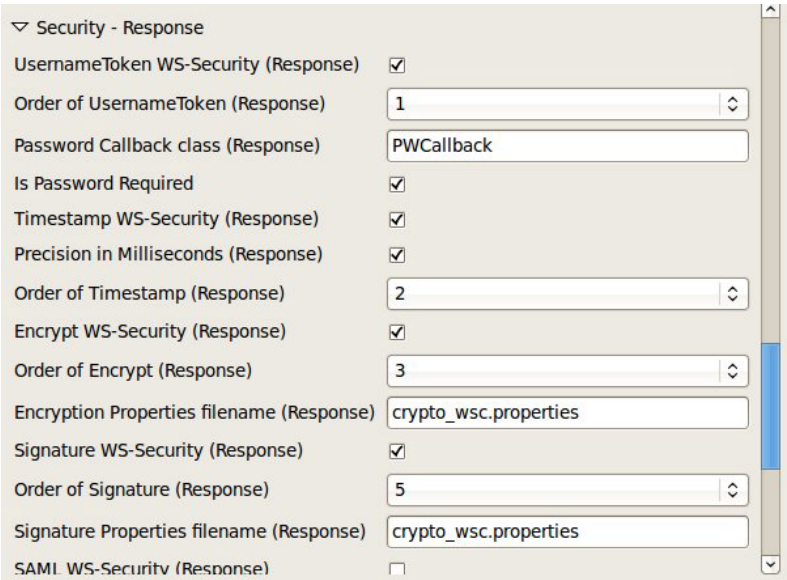


Figure: Sample WS-Security - Response configuration

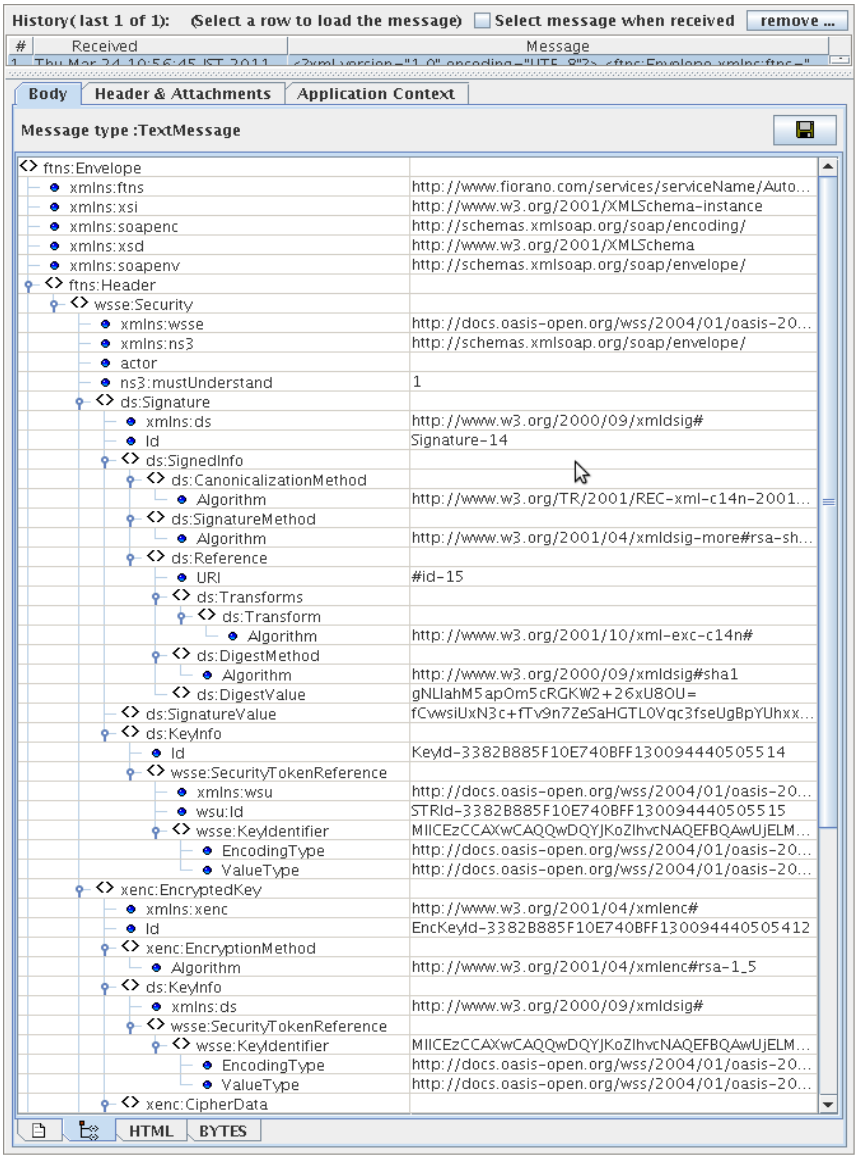


Figure: Sample Output in case of above configuration

4.14.2.5 Use Case Scenario

In a Salesforce Integration scenario, Salesforce updates are performed based on the details in the database.

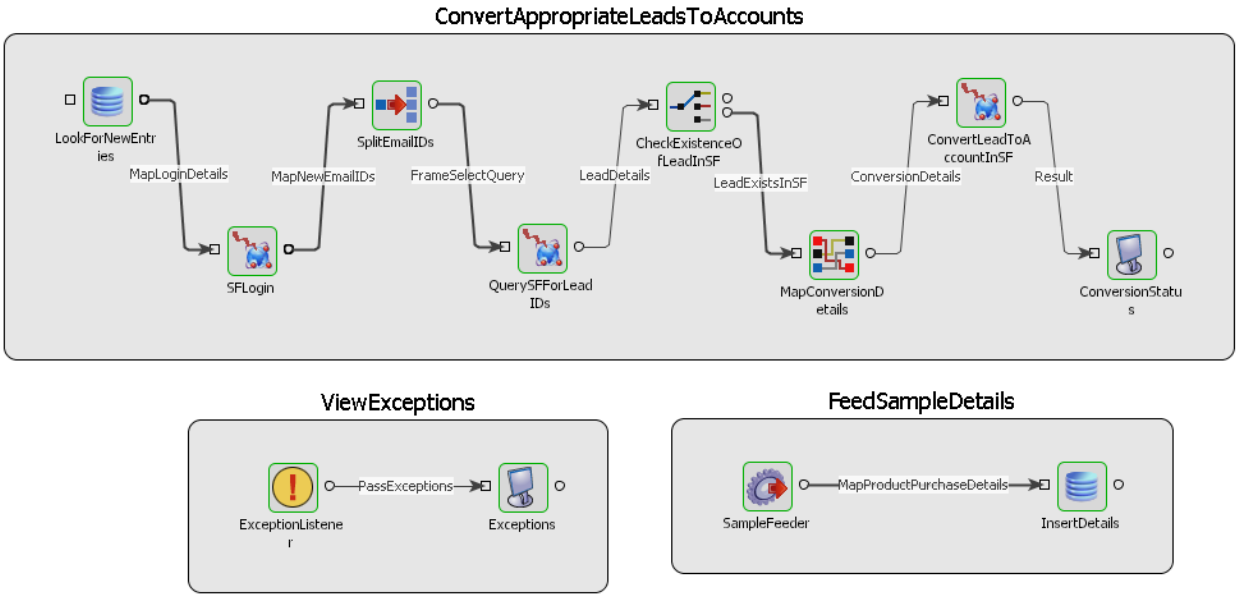


Figure 4.541: Salesforce Integration Scenario

The event process demonstrating this scenario is bundled with the installer.

Documentation of the scenario and instructions to run the flow can be found in the **Help** tab of flow when open in Studio.

4.14.2.6 Useful Tips

- If the web service is secured using basic authentication, then the details of the basic authentication should be provided in the Call Properties property during execution time.
- When using WS-Security, the Password Callback class should be the fully qualified name of the class.
- The orders in which the WS-Security tokens are specified are important and should be the order in which they are specified at the web service.
- This component supports only WSDL files which are compliant to WS-I Basic Profile 1.0.
- To pass http headers to the web service, the input message should contain properties with the header name prefixed with http_. For example, http_Content-Type.

4.14.3 Web Service Consumer (5.0)

This component invokes a web service (usually externally hosted on a third-party system) based on the configured WSDL. This component uses **axis2** API to invoke the web services.

Unlike most static web service client options (like Axis wsdl2java) which generates client stub code for invocation of a given WSDL, this component employs a dynamic invocation mechanism to ensure that no code needs be written or deployed for invoking a component.

The incoming request parameters are automatically wrapped in a SOAP request packet (handling different types of SOAP headers for handling web service security, transactions, and so on) and sent to underlying transport (usually the response is sent back to the client).

4.14.3.1 Configuration and Testing

The following properties can be configured in the Custom Property Sheet of the component.

General	
Use Operation Details From Input	
Use Connection Details From Input	no
Monitoring configuration	disabled: 5000 milli seconds
Error handling configuration	com.fiorano.edbc.framework.service.configuration.Conn...
Load WSDL From	URL
WSDL URL	http://www.webservicex.net/CurrencyConvertor.asmx?...
WebService Operation	ConversionRate
WSDL Service	{http://www.webserviceX.NET/}CurrencyConvertor
WSDL Port	CurrencyConvertorSoap
Endpoint Address	http://www.webservicex.net/CurrencyConvertor.asmx
Action	http://www.webserviceX.NET/ConversionRate
Socket Timeout	60000 milli-seconds
Connection Timeout	60000 milli-seconds
SOAPBody Namespace	http://schemas.xmlsoap.org/wsdl/soap/
Input Parameter	input
Output Parameter	output
HTTP Header protocol	HTTP/1.0
Advanced	
Authentication Type	None
Proxy Settings	Click ... to edit
SSL Security	**Click ... to edit**

Figure 4.542: Custom Property Sheet

4.14.3.1.1 General

Use Operation Details from Input

If this property is set to true, component will use the webservice operation details from the input request. If this property is set to true the properties Use Connection Details From Input, Load WSDL From, WSDL URL,WSDL File, WebService Operation, WSDL Service, WSDL Port, Endpoint Address, Action, SOAPBody Namespace, Input parameter, Output Parameter will be disabled.

Use Connection Details from Input

If this property is set to true, then the property "End Point Address" can be provided dynamically in the input request. If this property is selected and the "End Point Address" is not given in the input request, then the value given in this panel for that property will be used.

Error Handling Configuration

Click on the ellipsis button against Error Handling Configuration property to configure Error Handling properties.

Figure 4.543: Error Handling Configuration

The remedial actions to be taken when a particular error occurs can be configured here.

The default actions configured are:

- Log to Error logs
- Send to error port

Load WSDL From

Source for WSDL can be configured here. It can be a File or a valid WSDL URL.

When File is selected, the WSDL file location can be configured.

Figure 4.544: Specifying WSDL File

When URL is selected, the WSDL URL can be configured.

Figure 4.545: Specifying WSDL URL

WebService Operation

The web service operation to be invoked can be configured here.

Click on the ellipsis button against this property to select the operation from the available list of operations.

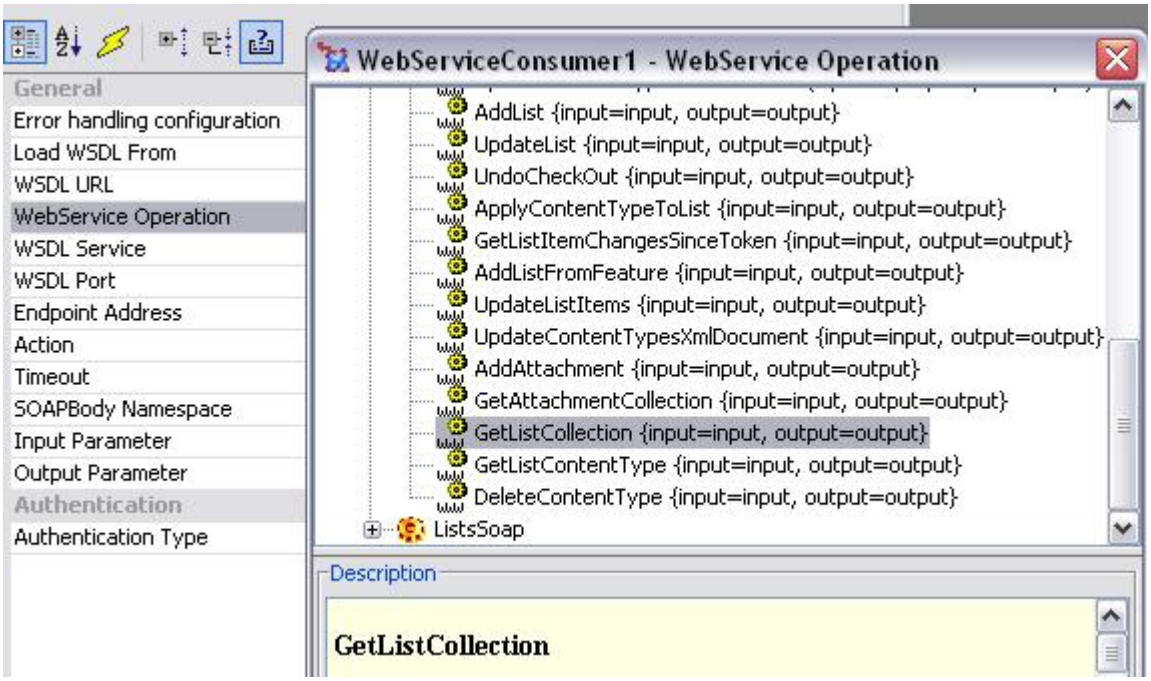


Figure 4.546: WSDL operations

Note: When the operation is selected, the parameters WSDL service, WSDL Port, Endpoint Address, Action, Soap Body NameSpace, Input Parameter and Output Parameter is populated automatically.

The Operation can also be typed manually in offline mode but all the other properties have to be populated manually.

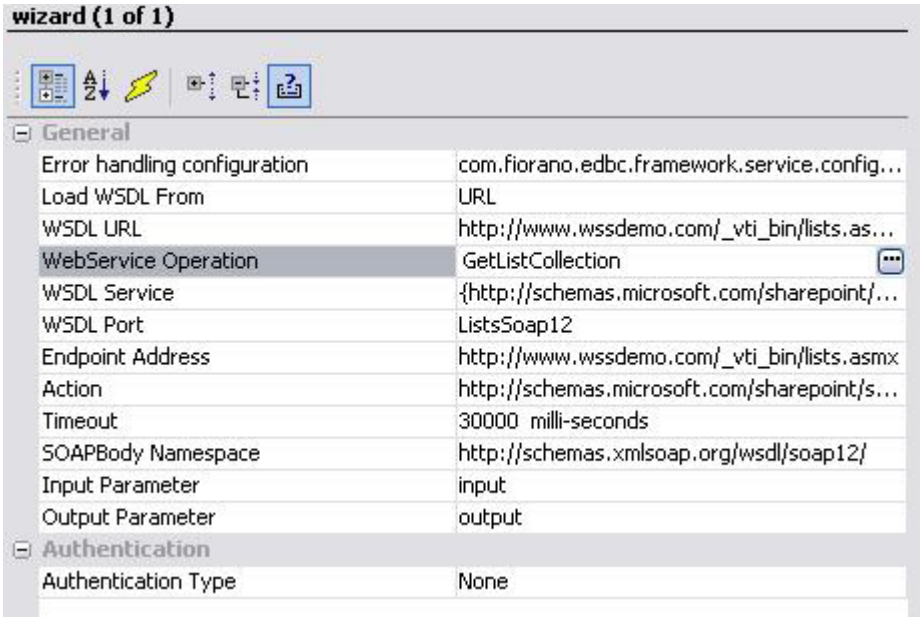


Figure 4.547: CPS after selecting WSDL operation

WSDLService: The service element defines the ports supported by the Web service. For each of the supported protocols, there is one port element. The service element is a collection of ports.

WSDL Port: Name of a single endpoint defined as a combination of a binding and a network address.

Endpoint Address: Provides a unique network address that a client uses to communicate with a service endpoint.

Soap Action: The SOAP Action field can be used to indicate the intent of the SOAP HTTP request. The value is a URI identifying the intent.

Timeout: The client's execute call will time out after waiting this amount of time.

SOAP Body Namespace: Namespace attribute of soap:body element. This is usually specified in case of RPC encoded web services.

Input Parameter: Input parameter name of the selected Operation. This is used to identify correct operation in the case of overloaded operations in the WSDL.

Output Parameter: Output parameter name of the selected Operation. This is used to identify correct operation in the case of overloaded operations in the WSDL.

4.14.3.1.2 Authentication Type

The type of Authentication to be used to connect to the web server. "None" specifies no authentication is required.

Other supported authentications include **Basic**, **Digest** and **NTLM**.

When Authentication Type is **Basic**, the following fields have to be specified.

Authentication	
Authentication Type	**Basic**
HTTP User Name	demouser
HTTP Password	Templates

Figure 4.548: Basic Authentication Type

- **HTTP User Name:** User Name to connect to the web server.
- **Password:** Password for the username mentioned.

Note: To access a share point web service, basic authentication is required.

When Authentication Type is **NTLM**, the following fields have to be populated

Authentication	
Authentication Type	NTLM
HTTP User Name	null
HTTP Password	null
Host Name	null
Host Port	0
Realm for Authentication	null

Figure 4.549: NTLM Authentication Type

- **HTTP User Name:** User Name to connect to the web server.
- **Password:** Password for the username mentioned.
- **Host Name:** Host that needed to be authenticated with.
- **Host Port:** Port of the host that needed to be authenticated with.
- **Realm for Authentication:** Realm for authentication scope.

When Authentication Type is **Digest**, the following fields have to be specified.

Authentication	
Authentication Type	**Digest**
HTTP User Name	null
HTTP Password	null
Host Name	null
Host Port	0
Host Domain Name	null

Figure 4.550: Digest Authentication Type

- **HTTP User Name:** User Name to connect to the webserver.
- **Password:** Password for the username mentioned.
- **Host Name:** Host that needed to be authenticated with.
- **Host Port:** Port of the host that needed to be authenticated with.
- **Host Domain Name:** Domain name needed by NTCredentials for NT Domain.

4.14.3.1.3 Using Named Configurations

Changes are done in eStudio w.r.t to Named Configurations. SSL Security details, Proxy Settings and HTTP Authentication Panels are modified to use Named Configurations.

4.14.3.2 Input and Output

4.14.3.2.1 Input Schema

The input schema is auto generated based on the configuration provided. For the configuration shown above, the schema would be

XSD - IN_PORT

Structure | Namespaces | Content

Schema: Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<s:schema xmlns:tns="http://schemas.microsoft.com/sharepoint/soap/" xm
   <s:import namespace="http://schemas.microsoft.com/sharepoint/soap/"
   <s:import namespace="http://microsoft.com/wsdl/types/"/>
   <s:import namespace="http://www.w3.org/2001/XMLSchema"/>
   <s:element name="Envelope">
      <s:complexType>
         <s:sequence>
            <s:element ref="ftns:Header"/>
            <s:element ref="ftns:Body"/>
         </s:sequence>
      </s:complexType>
   </s:element>
   <s:element name="Header">
      <s:complexType>
         <s:sequence/>
      </s:complexType>
   </s:element>
   <s:element name="Body">
      <s:complexType>
         <s:sequence>
            <s:element ref="tns:GetListCollection"/>
         </s:sequence>
      </s:complexType>
   </s:element>
</s:schema>
```

Figure 4.551: Input Schema

When the property Use Connection details from input is chosen, an additional element ConnectionFactorySettings is added to the input schema, as shown in the figure. Properties that are used to create the connection are present under this element.

XML Item	Value Type
<ns1:Envelope>	
<ConnectionProperties>?	
<EndPointAddress>?	#string
<ns1:Header>	
<ns1:Body>	
<ns2:ConversionRate>	
<ns2:FromCurrency>	AFA | ALL | DZD | ARS | AWG | AUD | B...
<ns2:ToCurrency>	AFA | ALL | DZD | ARS | AWG | AUD | B...

Figure: Input schema with ConnectionProperties

When the property "Use Operation Details From Input" is chosen, the wsdl operation details are added to the input schema, as shown in the following figure.

<ns1:WSCMessage>	
<ns1:WebServiceOpearation>	#string
<ns1:WSDLService>	#string
<ns1:EndPointAddress>	#string
<ns1:SoapActionURI>?	#string
<ns1:TargetNamespace>	#string
<ns1:SoapBodyNamespace>	#string
<ns1:SOAPEnvelope>	#string

Figure: Input schema with Operation Details

4.14.3.2.2 Output Schema

The output schema is auto generated based on the configuration provided. For the configuration shown above, the schema would be

XSD - OUT_PORT

Structure | Namespaces | Content

Schema: Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<s:schema xmlns:tns="http://schemas.microsoft.com/sharepoint/soap/"
   <s:import namespace="http://schemas.microsoft.com/sharepoint/soap
   <s:import namespace="http://microsoft.com/wsdl/types/"/>
   <s:import namespace="http://www.w3.org/2001/XMLSchema"/>
   <s:element name="Envelope">
      <s:complexType>
         <s:sequence>
            <s:element ref="ftns:Header"/>
            <s:element ref="ftns:Body"/>
         </s:sequence>
      </s:complexType>
   </s:element>
   <s:element name="Header">
      <s:complexType>
         <s:sequence/>
      </s:complexType>
   </s:element>
   <s:element name="Body">
      <s:complexType>
         <s:sequence>
            <s:element ref="tns:GetListCollectionResponse"/>
         </s:sequence>
      </s:complexType>
   </s:element>
</s:schema>
```

Figure 4.552: Output Schema

4.14.3.3 Functional Demonstration

4.14.3.3.1 Scenario 1

Invoking a web service operation using a WSDL from the following URL

http://www.webservicex.net/CurrencyConvertor.asmx?WSDL

Configure the Web Service Consumer component as described and use Feeder and Display components to send sample input and check the response respectively.

Figure 4.553: Feeder and Display Components

Sample Input

```
1 <ns1:Envelope xmlns:ns1="http://www.webserviceX.NET/ConversionRate/Input">
2     <ns1:Header/>
3     <ns1:Body>
4         <ns2:ConversionRate xmlns:ns2="http://www.webserviceX.NET/">
5             <ns2:FromCurrency>USD</ns2:FromCurrency>
6             <ns2:ToCurrency>INR</ns2:ToCurrency>
7         </ns2:ConversionRate>
8     </ns1:Body>
9 </ns1:Envelope>
```

Figure 4.554: Demonstrating scenario with sample input

Sample Output

```
<responsens:Envelope xmlns:responsens="http://www.webserviceX.NET/Co
<responsens:Body>
<ConversionRateResponse xmlns="http://www.webserviceX.NET/">
<ConversionRateResult>39.95</ConversionRateResult>
</ConversionRateResponse>
</responsens:Body>
</responsens:Envelope>
```

Figure 4.555: Demonstrating scenario with sample input and output

4.14.3.3.2 Scenario 2

Invoking a web service operation using a WSDL from the following URL, by setting the property "Use Operation Details From Input" to true

http://www.webservicex.net/CurrencyConvertor.asmx?WSDL

Configure the Web Service Consumer component as described and use Feeder and Display components to send sample input and check the response respectively.

Sample Input

```
<nsl:WSCMessage xmlns:ns1="http://www.fiorano.com/fesb/activity/WSInvoker/">
   <nsl:WebServiceOpearation>ConversionRate</nsl:WebServiceOpearation>
   <nsl:WSDLService>CurrencyConvertor</nsl:WSDLService>
   <nsl:EndPointAddress>http://www.webservicex.net/CurrencyConvertor.asmx</nsl:EndPointAddress>
   <nsl:SoapActionURI>http://www.webserviceX.NET/ConversionRate</nsl:SoapActionURI>
   <nsl:TargetNamespace>http://www.webserviceX.NET/</nsl:TargetNamespace>
   <nsl:SoapBodyNamespace>http://schemas.xmlsoap.org/wsdl/soap/</nsl:SoapBodyNamespace>
   <nsl:SOAPEnvelope><![CDATA[<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
                                xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <soap:Body>
   <ns2:ConversionRate xmlns:ns2="http://www.webserviceX.NET/">
         <ns2:FromCurrency>EGP</ns2:FromCurrency>
         <ns2:ToCurrency>TND</ns2:ToCurrency>
      </ns2:ConversionRate>
          </soap:Body></soap:Envelope>]]></nsl:SOAPEnvelope>
</nsl:WSCMessage>
```

Sample Output

```
<?xml version='1.0' encoding='utf-8'?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body><ConversionRateResponse xmlns="http://www.webserviceX.NET/">
<ConversionRateResult>0.2536</ConversionRateResult></ConversionRateResponse>
</soap:Body></soap:Envelope>
```

Figure 4.556: Demonstrating scenario with sample input and output

4.15 Finance

4.15.1 FIXAcceptor

The Financial Information eXchange ("FIX") Protocol is a series of messaging specifications for the electronic communication of trade-related messages. FIX Protocol helps in an automated, electronic, transparent, reliable, cost efficient and timely communication between counterparties in financial services sector.

FIXAcceptor is a FIX engine which can be used by sell side firms such as brokers or dealers to receive FIX messages like order placements from buy side firms such as financial institutions. FIXAcceptor supports data persistence for storing session state and messages. It supports FIX Protocol versions FIX.4.0 - FIX.4.4 and FIX.5.0 as well as Transport Independence version FIXT.1.1. Apart from FIX Session protocol for session layer communication, FIXAcceptor can also be used to communicate over Java Messaging Service.

4.15.1.1 Configuration

4.15.1.1.1 Common Session Configuration

Session details that are common across all FIX sessions should be configured here.

General Configuration

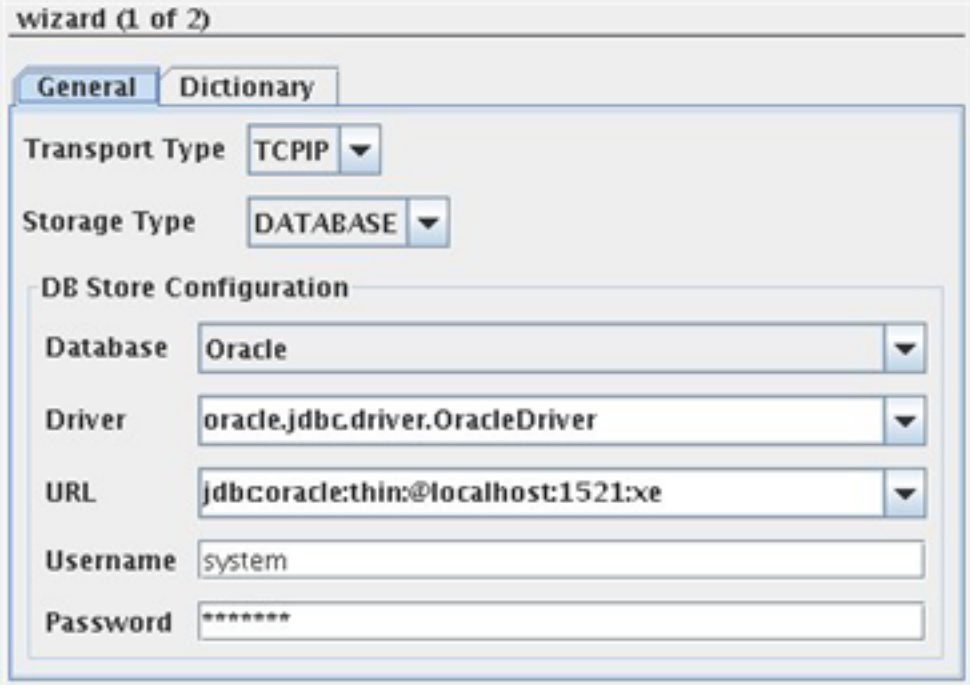


Figure 1: General Configuration

- **Transport Type** – The transport type used by session layer communication. Transport Type can be either TCPIP or JMS.
    - **TCPIP** - Session communication is done using FIX Session Protocol i.e., using socket layer communication.
    - **JMS** - Session communication is done using Java Messaging Service.
- **Storage Type** – The storage type used for storing sessions' and messages' details. Storage Type can be either FILE or DATABASE.
    - **FILE** - Details regarding sessions and messages are stored in files.
    - **DATABASE** - Details regarding sessions and messages are stored in a database.

When storage type is DATABASE, the database details panel is shown.

DB Store Configuration

Database - The database that is used to store details regarding sessions and messages. One of the databases present in the drop down can be selected. The default driver and url values are shown which may be edited as required.

- **Driver** - The driver class using which the component connects to database. The resource file containing the driver class should be added as a resource to FIXLib service in System Lib category.
- **URL** - The database URL to which the component connects to.
- **Username** - The database username with which the component connects to database.
- **Password** - The database password with which the component connects to database.

Note: Please refer to property JdbcSessionIdDefaultPropertyValue in Properties section when configuring for Oracle database.

4.15.1.1.2 Dictionary

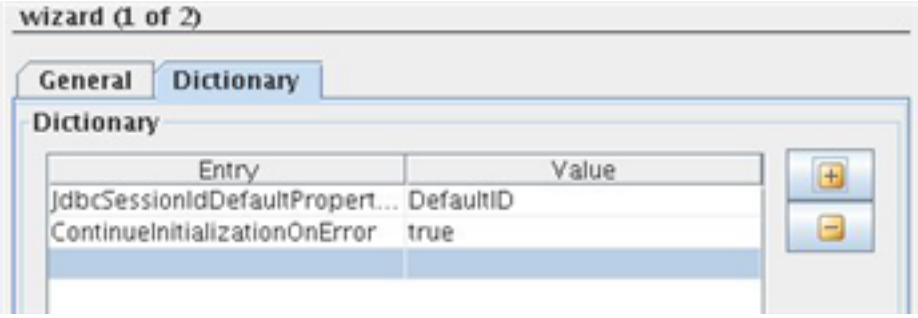


Figure 2: Common Properties Configuration

Properties which are common across all FIX sessions can be specified in common session dictionary. The properties which may be provided here are explained in Properties section.

4.15.1.1.3 Session Configuration

A single FIXAcceptor instance can be configured to create several FIX sessions. FIX Session details that are specific to each individual FIX session should be configured by adding a session using + on left hand side.

4.15.1.1.3.1 General Configuration

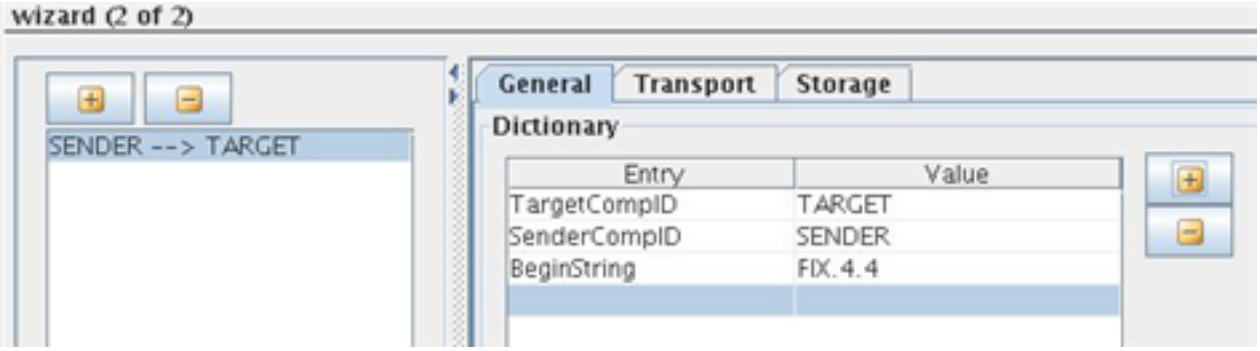


Figure 3: General Session Configuration

Properties related to a specific session such as TargetCompID, SenderCompID, BeginString are provided here. The properties which may be provided here are explained in Properties section.

4.15.1.1.3.2 Transport Configuration

Transport Configuration is shown based on the transport type selected in the previous panel. In case of JMS transport type, transport configuration panel is not shown.

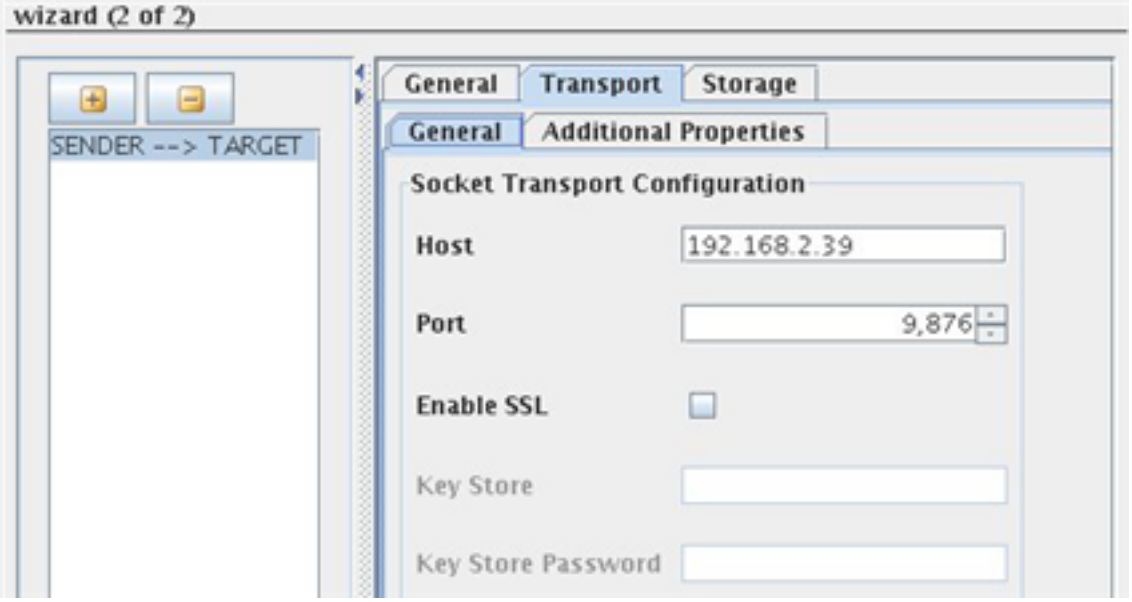


Figure 4: Transport Configuration

General

Socket Transport Configuration

Socket configuration details in case TCPIP transport type can be provided here.

- **Host** - The hostname of the system on which the acceptor is running to which the initiator will connect to.
- **Port** - The port on which the acceptor is running to which the initiator will connect to. Note: The various sessions configured in a FIXAcceptor instance should run on different ports and same host.
- **Enable SSL** - Whether to use SSL while creating socket connections between initiator and acceptor.
- **Keystore** - Keystore used for creating secure connection between initiator and acceptor.
- **Keystore Password** - Password of the keystore specified above.

Additional Properties

Additional properties that are used while creating socket connections must be provided in Dictionary properties in previous panel, that is, the Common Session Configuration panel. The properties which may be there are explained in Properties section. The additional properties table here can be left empty.

4.15.1.1.3.3 Storage Configuration

Details regarding FIX sessions and messages can be persisted for further use such as in case of resend requests. This can be achieved by using either file or database storage. Storage Configuration is shown based on the storage type selected in the previous panel.

Enable Storage - All the sessions that are created and messages that are processed by FIX engine will be stored for further use. If set to false, no messages will be persisted. This will force the engine to always send GapFills instead of resending messages. It can be set to false if resending a message is not required such as in case of market data streams.

File Store Configuration

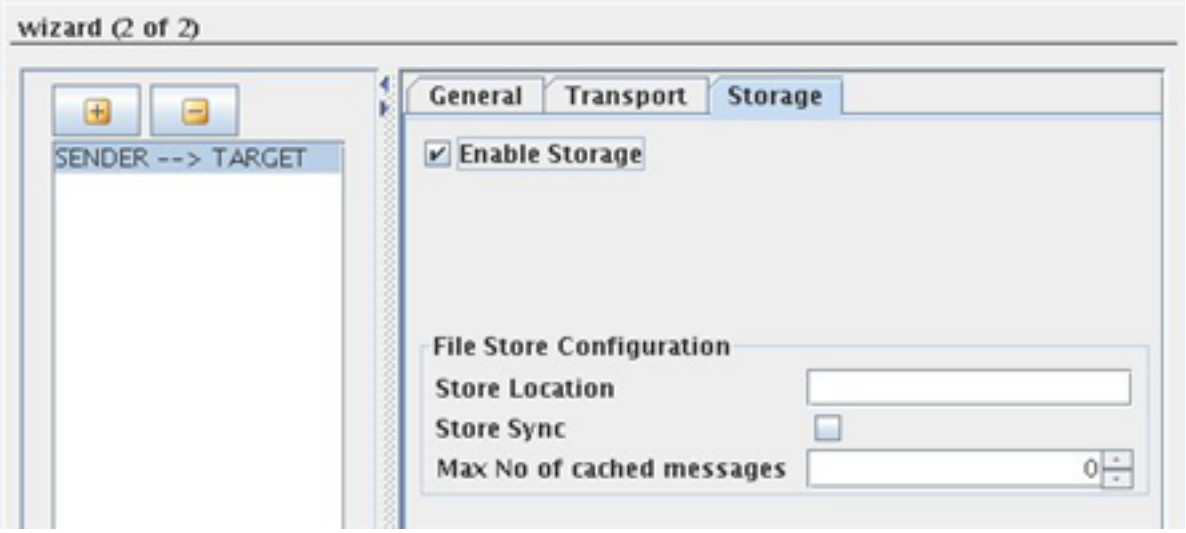


Figure 5: File Store Configuration

- **Store Location** - Directory to store sequence number, message and session files. The value should be a valid directory and must have write access.
- **Store Sync** - Boolean option for controlling whether the FileStore syncs to the hard drive on every write. It's safer to synchronize, but it's also much slower (100x or more slower in some cases).
- **Max. No. of cached messages** - Maximum number of message index entries to cache in memory. The value should be an Integer. A zero will not cache any entries.

Database Store Configuration

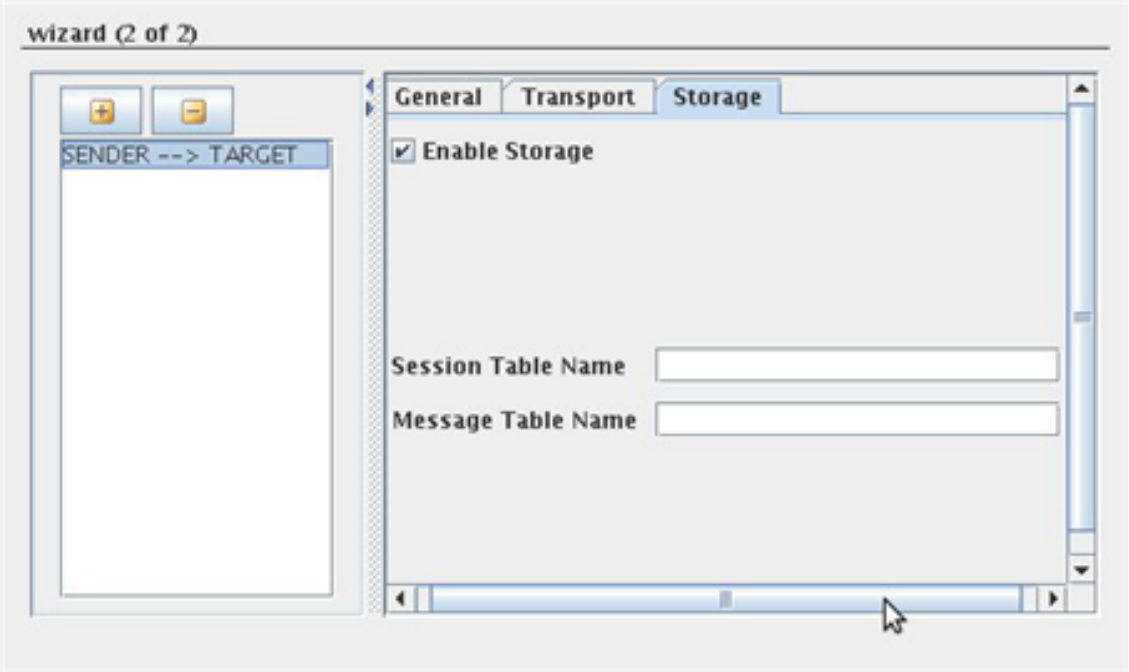


Figure 6: Database Store Configuration

- Sessions Table Name - Valid SQL table name for sessions table. If a table by this name does not already exist when the component is launched, the component will create a table with the name specified with following columns -

```
beginstring VARCHAR(8) NOT NULL,  sendercompid VARCHAR(64)
NOT NULL,
sendersubid VARCHAR(64) NOT NULL,  senderlocid VARCHAR(64)
NOT NULL,
targetcompid VARCHAR(64) NOT NULL, targetsubid VARCHAR(64)
NOT NULL,
targetlocid VARCHAR(64) NOT NULL, session_qualifier
VARCHAR(64) NOT NULL,
creation_time TIMESTAMP NOT NULL, incoming_seqnum INT NOT
NULL,
outgoing_seqnum INT NOT NULL
```

  and primary key -

```
PRIMARY KEY (beginstring, sendercompid, sendersubid,
senderlocid, targetcompid, targetsubid, targetlocid,
session_qualifier)
```

- Messages Table Name - Valid SQL table name for messages table. If a table by this name does not already exist when the component is launched, the component will create a table with the name specified with following columns -

```
beginstring VARCHAR(8) NOT NULL, sendercompid VARCHAR(64)
NOT NULL,
sendersubid VARCHAR(64) NOT NULL,  senderlocid VARCHAR(64)
NOT NULL,          targetcompid VARCHAR(64) NOT NULL,
targetsubid VARCHAR(64) NOT NULL,           targetlocid
VARCHAR(64) NOT NULL,  session_qualifier VARCHAR(64) NOT
NULL,           msgseqnum INT NOT NULL, message
VARCHAR(4000) NOT NULL
```

  and primary key -

```
PRIMARY KEY (beginstring, sendercompid, sendersubid,
senderlocid, targetcompid, targetsubid, targetlocid,
session_qualifier, msgseqnum)
```

4.15.1.2 Properties

The properties that can be specified in all properties tables across the configuration are mentioned below. All the Boolean type properties' value should be specified as either 'true' or 'false'.

4.15.1.2.1 Session Properties

- **BeginString** - Version of FIX this session should use. Valid values are FIXT.1.1, FIX.4.4, FIX.4.3, FIX.4.2, FIX.4.1, FIX.4.0. Each session should have a BeginString specified in its properties.
- **SenderCompID** - Sender's compID as associated with this FIX session, a case-sensitive alpha-numeric string. SenderCompID is a required property for each session.

- **SenderSubID** - Sender's subID as associated with this FIX session, a case-sensitive alpha-numeric string.
- **SenderLocationID** - Sender's locationID as associated with this FIX session, a case-sensitive alpha-numeric string.
- **TargetCompID** - counterparty's compID as associated with this FIX session, a case-sensitive alpha-numeric string. TargetCompID is a required property for each session.
- **TargetSubID** - counterparty's subID as associated with this FIX session, a case-sensitive alpha-numeric string.
- **TargetLocationID** - counterparty's locationID as associated with this FIX session, a case-sensitive alpha-numeric string.
- **SessionQualifier** - Additional qualifier to disambiguate otherwise identical sessions.
- **DefaultApplVerID** - This must and should be specified only for FIXT 1.1 (and newer). Ignored for earlier transport versions. Specifies the default application version ID for the session. This can either be the ApplVerID enum (see the ApplVerID field in FIX specification) or the beginString for the default version.

For example:

Enum. FIX 5.0 over FIXT 1.1

DefaultApplVerID=7

BeginString: FIX 5.0 over FIXT 1.1

DefaultApplVerID=FIX.5.0

BeginString: FIX 4.2 over FIXT 1.1

DefaultApplVerID=FIX.4.2

- **TimeZone** - Time zone for this session; if specified, the session start and end will be converted from this zone to UTC. Time zone ID (America/New_York, Asia/Tokyo, Europe/London, etc.)
- **StartTime** - Time of day that this FIX session becomes activated. Time should be specified in the format of HH:MM:SS [timezone]. The time zone is optional. The TimeZone setting will be used, if set, or UTC will be used by default. The timezone string should be one that the Java TimeZone class can resolve. For example, "15:00:00 US/Central". StartTime is a required property for each session.
- **EndTime** - Time of day that this FIX session becomes deactivated. Time should be specified in the format of HH:MM:SS [timezone]. The time zone is optional. The TimeZone setting will be used, if set, or UTC will be used by default. The timezone string should be one that the Java TimeZone class can resolve. For example, "09:00:00 US/Eastern". EndTime is a required property for each session.
- **StartDay** - For week long sessions, the starting day of week for the session. Use in combination with StartTime. Day of week in the default locale (e.g. Monday, mon, lundi, lun. etc.)
- **EndDay** - For week long sessions, the ending day of week for the session. Use in combination with EndTime. Day of week in the default locale (e.g. Monday, mon, lundi, lun. etc.)

- **MillisecondsInTimeStamp(Boolean)** - Boolean to determine if milliseconds should be added to timestamps. Only available for FIX.4.2 and greater. Default value is true.
- **ClosedResendInterval(Boolean)** - Boolean to determine whether to use actual end of sequence gap for resend requests rather than using "infinity" as the end sequence of the gap. Not recommended by the FIX specification, but needed for some counter parties. Default value is false.
- **RefreshOnLogon(Boolean)** – Boolean to determine whether to refresh the session state when a logon is received. This allows a simple form of failover when the message store data is persistent. The option will be ignored for message stores that are not persistent (e.g., MemoryStore). Default value is false.

4.15.1.2.2 Validation Properties

- **ValidateFieldsOutOfOrder(Boolean)** - If set to false, fields that are out of order (i.e. body fields in the header, or header fields in the body) will not be rejected. Useful for connecting to systems which do not properly order fields. Default value is true.
- **ValidateFieldsHaveValues(Boolean)** - If set to false, fields without values (empty) will not be rejected. Useful for connecting to systems which improperly send empty tags. Default value is true.
- **ValidateUserDefinedFields(Boolean)** - If set to false, user defined fields will not be rejected if they are not defined in the data dictionary, or are present in messages they do not belong to. Default value is false.
- **AllowUnknownMsgFields(Boolean)** - Allow unknown fields in messages. This is intended for unknown fields with tags < 5000 (not user defined fields). Use ValidateUserDefinedFields for controlling validation of tags >= 5000. Default value is false.
- **CheckCompID(Boolean)** - If set to true, messages must be received from the counterparty with the correct SenderCompID and TargetCompID. Some systems will send different CompIDs by design, so this must be set to false. Default value is true.
- **CheckLatency(Boolean)** - If set to true, messages must be received from the counterparty within a defined number of seconds (see MaxLatency). It is useful to turn this off if a system uses local time for it's timestamps instead of GMT. Default value is true.
- **MaxLatency(Boolean)** - If CheckLatency is set to true, this defines the number of seconds latency allowed for a message to be processed. Default is 120. It should be a valid positive integer.

4.15.1.2.3 Socket Properties

FIXAcceptor socket properties must be set in common session dictionary.

- **SocketKeepAlive(Boolean)** - When the keep alive option is set for a TCP socket and no data has been exchanged across the socket in either direction for 2 hours (NOTE: the actual value is implementation dependent), TCP automatically sends a keep alive probe to the peer. This probe is a TCP segment to which the peer must respond. One of the following three responses is expected -

1. The peer responds with the expected ACK. The application is not notified (since everything is OK). TCP will send another probe following another 2 hours of inactivity.
2. The peer responds with an RST, which tells the local TCP that the peer host has crashed and rebooted. The socket is closed.
3. There is no response from the peer. The socket is closed. The purpose of this option is to detect if the peer host crashes.

- **SocketOobInline(Boolean)** - When the OOBINLINE option is set, any TCP urgent data received on the socket will be received through the socket input stream. When the option is disabled (which is the default) urgent data is silently discarded.
- **SocketReceiveBufferSize** - Set a hint the size of the underlying buffers used by the platform for incoming network I/O. When used in set, this is a suggestion to the kernel from the application about the size of buffers to use for the data to be received over the socket. A valid Integer value should be specified.
- **SocketReuseAddress(Boolean)** - Sets SO_REUSEADDR for a socket. This is used only for MulticastSockets in java, and it is set by default for MulticastSockets.
- **SocketSendBufferSize** - Set a hint the size of the underlying buffers used by the platform for outgoing network I/O. When used in set, this is a suggestion to the kernel from the application about the size of buffers to use for the data to be sent over the socket. A valid Integer value should be specified.
- **SocketLinger** - Specify a linger-on-close timeout. This option disables/enables immediate return from a close() of a TCP Socket. Enabling this option with a non-zero Integer timeout means that a close() will block pending the transmission and acknowledgement of all data written to the peer, at which point the socket is closed gracefully. Upon reaching the linger timeout, the socket is closed forcefully, with a TCP RST. Enabling the option with a timeout of zero does a forceful close immediately. If the specified timeout value exceeds 65,535 it will be reduced to 65,535. It should be an Integer value.
- **SocketTcpNoDelay(Boolean)** - Disable Nagle's algorithm for this connection. Written data to the network is not buffered pending acknowledgement of previously written data. Default value is true.
- **SocketTrafficClass** - Sets traffic class or type-of-service octet in the IP header for packets sent from this Socket. As the underlying network implementation may ignore this value applications should consider it a hint.

  The tc must be in the range 0 <= tc <= 255 or an IllegalArgumentException will be thrown.

  Notes: For Internet Protocol v4 the value consists of an octet with precedence and TOS fields as detailed in RFC 1349. The TOS field is bitset created by bitwise-or'ing values such the following -

  IPTOS_LOWCOST (0x02)

  IPTOS_RELIABILITY (0x04)

  IPTOS_THROUGHPUT (0x08)

  IPTOS_LOWDELAY (0x10)

The last low order bit is always ignored as this corresponds to the MBZ (must be zero) bit.

Setting bits in the precedence field may result in a SocketException indicating that the operation is not permitted.

The property takes an integer value or a set of string options separated by "|" (e.g., "IPTOS_LOWCOST|IPTOS_LOWDELAY").

- **SocketSynchronousWrites(Boolean)** - Write messages synchronously. This is not generally recommended as it may result in performance degradation. The MINA communication layer is asynchronous by design, but this option will override that behavior if needed. Default value is false.
- **SocketSynchronousWriteTimeout** - The time in milliseconds to wait for a write to complete. An Integer value should be specified. Defaults to 30000 ms if SocketSynchronousWrites is true.

4.15.1.2.4 Storage Properties

- **JdbcSessionIdDefaultPropertyValue** - The default value for Session ID bean properties (namely BeginString, SenderCompID, SenderSubID, SenderLocationID, TargetCompID, TargetSubID, TargetLocationID) is an empty string. Oracle treats this as a SQL NULL and that causes problems. This configuration setting allows to set the default value for unspecified Session ID properties. This property takes any nonempty string value.

4.15.1.2.5 Miscellaneous Properties

- **ResetOnLogon(Boolean)** - Determines if sequence numbers should be reset before sending/receiving a logon request. Default value is false.
- **ResetOnLogout(Boolean)** - Determines if sequence numbers should be reset to 1 after a normal logout termination. Default value is false.
- **ResetOnDisconnect(Boolean)** - Determines if sequence numbers should be reset to 1 after an abnormal termination. Default value is false.
- **ForceResync(Boolean)** - When logout response contains "expecting XXX", the sequence numbers of forcibly resynchronized. Default value is false.
- **NonStopSession(Boolean)** - Session scheduling setting to specify that session never ends/resets. Default value is false.
- **ContinueInitializationOnError(Boolean)** - Continue initializing sessions if an error occurs. Default value is false.
- **SendRedundantResendRequests(Boolean)** - Allows sending of redundant resend requests. Default value is false.

4.15.1.3 Functional Demonstration

Configure the FIXAcceptor as described in Configuration section and await session logon message from FIXInitiator.

4.15.1.3.1 Scenario 1

Configure FIXInitiator and FIXAcceptor with TCPIP transport type.

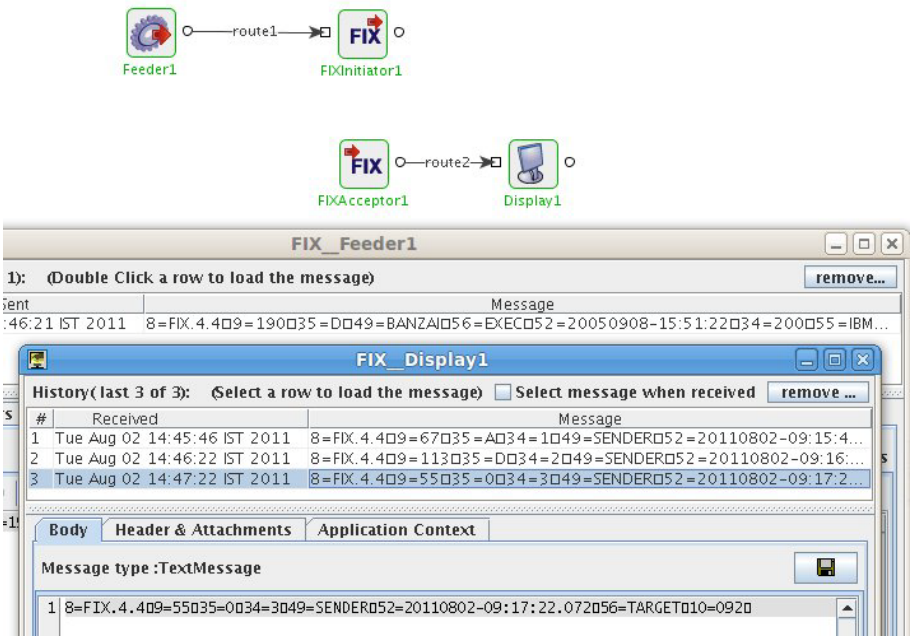


Figure 7: TCPIP Transport Type scenario

FIXInitiator sends Logon message (35=A) as soon as it is started to establish a session using FIX Session protocol. It then sends a NewOrderSingle message (35=D) for FIXAcceptor to execute the order. FIXAcceptor sends the NewOrderSingle message on to its outport for further processing. When FIXInitiator is not sending any messages like order requests to FIXAcceptor, it keeps pinging the FIXInitiator by sending Heartbeat messages (35=0) for keeping the session alive.

4.15.1.3.2 Scenario 2

Configure FIXInitiator and FIXAcceptor with JMS transport type.

Figure 8: JMS Transport Type scenario

Communication between FIXInitiator and FIXAcceptor is similar to Scenario 1 with the only difference being the medium. Here, the communication is done over JMS. FIXInitiator sends admin messages like logon, heartbeat and application messages like order requests to FIXAcceptor via JMS. FIXAcceptor also sends admin messages to FIXInitiator in the same way.

4.15.1.4 Useful Tips

After FIXAcceptor is started, it waits to listen for FIX Logon messages sent by counterparties and sends a reply on receiving those messages indicating whether or not a session is successfully initiated or not.

In case of JMS transport type, inorder for FIXAcceptor not to send application messages back to FIXInitiator and to limit the communication from FIXAcceptor to FIXInitiator to just admin messages, a JMS selector (ADMIN_MSG_PROPERTY IS NOT NULL) should be specified on route between outport of FIXAcceptor and inport of FIXInitiator (route3 as shown in the image above).

4.15.2 FIXInitiator

The Financial Information eXchange ("FIX") Protocol is a series of messaging specifications for the electronic communication of trade-related messages. FIX Protocol helps in an automated, electronic, transparent, reliable, cost efficient and timely communication between counterparties in financial services sector.

FIXInitiator is a FIX engine which can be used by buy side firms such as financial institutions to send FIX messages like order placements to sell side firms such as brokers or dealers. FIXInitiator supports data persistence for storing session state and messages. It supports FIX Protocol versions FIX.4.0 - FIX.4.4 and FIX.5.0 as well as Transport Independence version FIXT.1.1. Apart from FIX Session protocol for session layer communication, FIXInitiator can also be used to communicate over Java Messaging Service.

4.15.2.1 Configuration

4.15.2.1.1 Common Session Configuration

Session details that are common across all FIX sessions should be configured here.

General Configuration

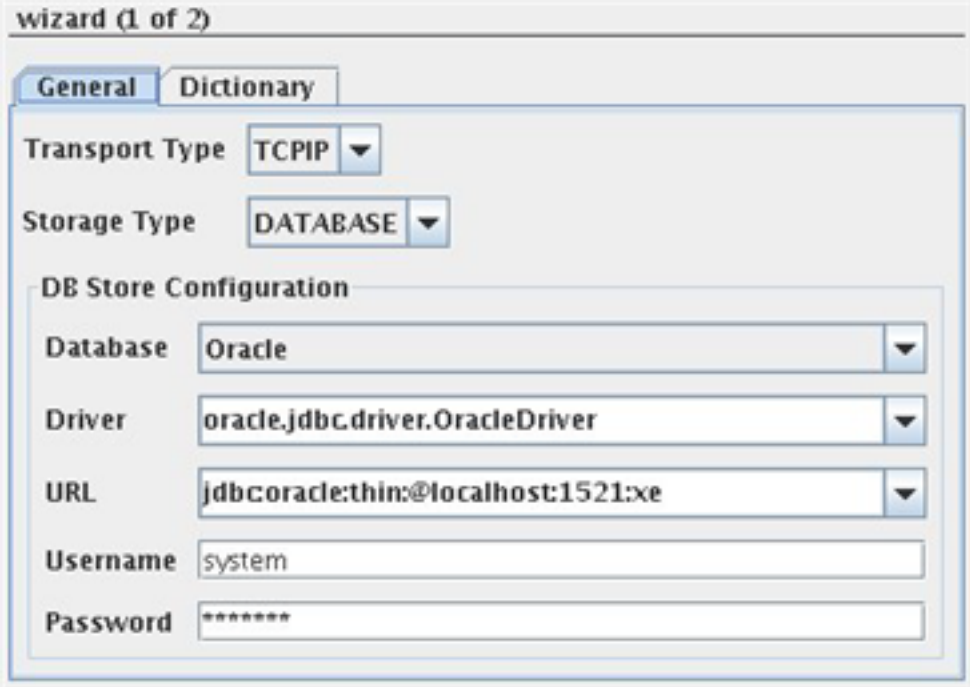


Figure 1: General Configuration

- **Transport Type** – The transport type used by session layer communication. Transport Type can be either TCPIP or JMS.
    - **TCPIP** - Session communication is done using FIX Session Protocol i.e., using socket layer communication.
    - **JMS** - Session communication is done using Java Messaging Service.
- **Storage Type** – The storage type used for storing sessions' and messages' details. Storage Type can be either FILE or DATABASE.
    - **FILE** - Details regarding sessions and messages are stored in files.
    - **DATABASE** - Details regarding sessions and messages are stored in a database.

When storage type is DATABASE, the database details panel is shown.

- **Database** - The database that is used to store details regarding sessions and messages. One of the databases present in the drop down can be selected. The default driver and url values are shown which may be edited as required.
- **Driver** - The driver class using which the component connects to database. The resource file containing the driver class should be added as a resource to FIXLib service in System Lib category.
- **URL** - The database URL to which the component connects to.
- **Username** - The database username with which the component connects to database.
- **Password** - The database password with which the component connects to database.

Note: Please refer to property JdbcSessionIdDefaultPropertyValue in Properties section when configuring for Oracle database.

4.15.2.1.2 Dictionary

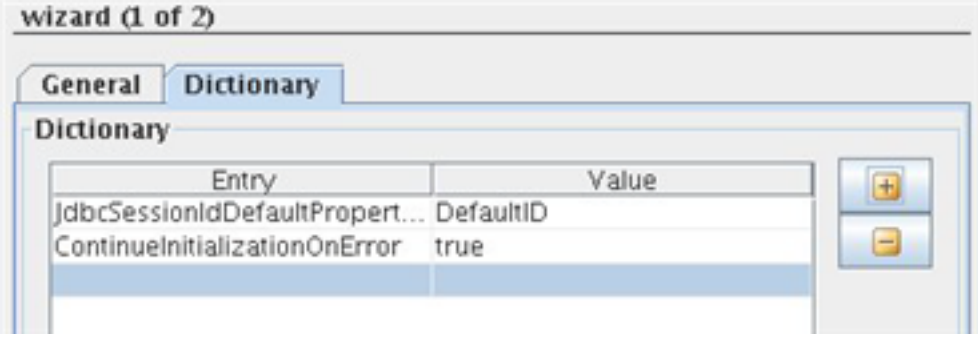


Figure 2: Common Properties Configuration

Properties which are common across all FIX sessions can be specified in common session dictionary. The properties which may be provided here are explained in Properties section.

4.15.2.1.3 Session Configuration

A single FIXInitiator instance can be configured to create several FIX sessions. Session details that are specific to each individual FIX session should be configured by adding a session using + on left hand side.

4.15.2.1.3.1 General Configuration

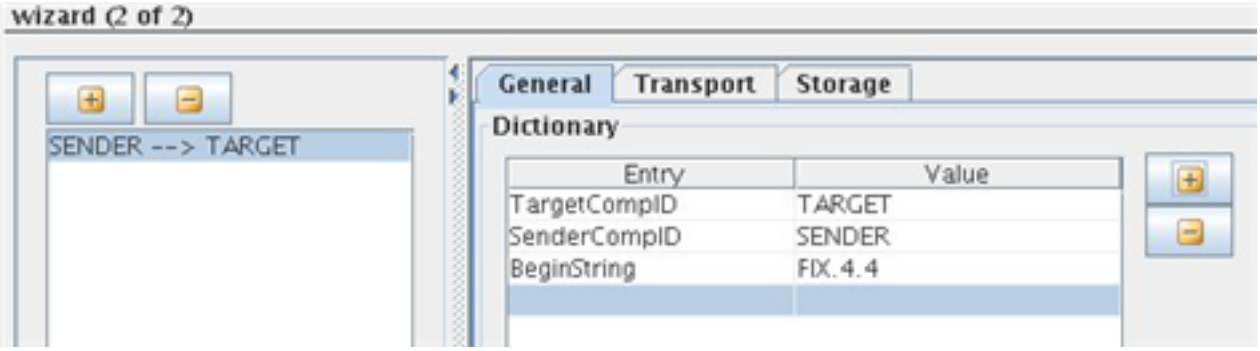


Figure 3: General Session Configuration

Properties related to a specific session such as BeginString, TargetCompID, SenderCompID are provided here. The properties which may be provided here are explained in Properties section.

4.15.2.1.0033.2 Transport Configuration

Transport Configuration is shown based on the transport type selected in the previous panel. In case of JMS transport type, transport configuration panel is not shown.

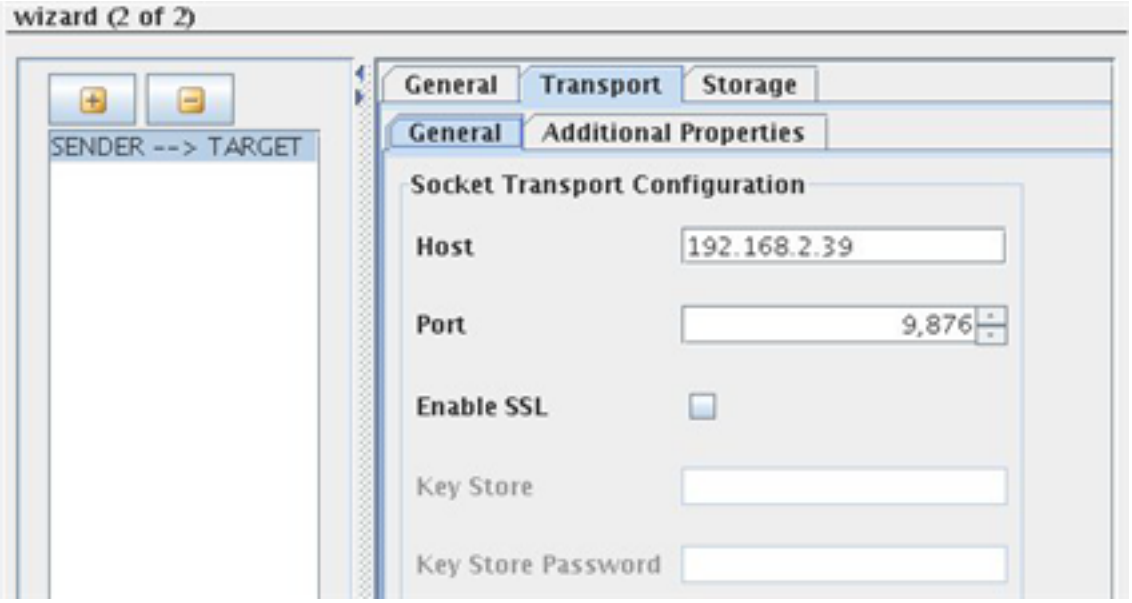


Figure 4: Transport Configuration

General

Socket Transport Configuration

Socket configuration details in case TCPIP transport type can be provided here.

- **Host** - The hostname of the system on which the acceptor is running to which the initiator will connect to.
- **Port** - The port on which the acceptor is running to which the initiator will connect to.
- **Enable SSL** - Whether to use SSL while creating socket connections between initiator and acceptor.
- **Keystore** - Keystore used for creating secure connection between initiator and acceptor.
- **Keystore Password** - Password of the keystore specified above.

Additional Properties

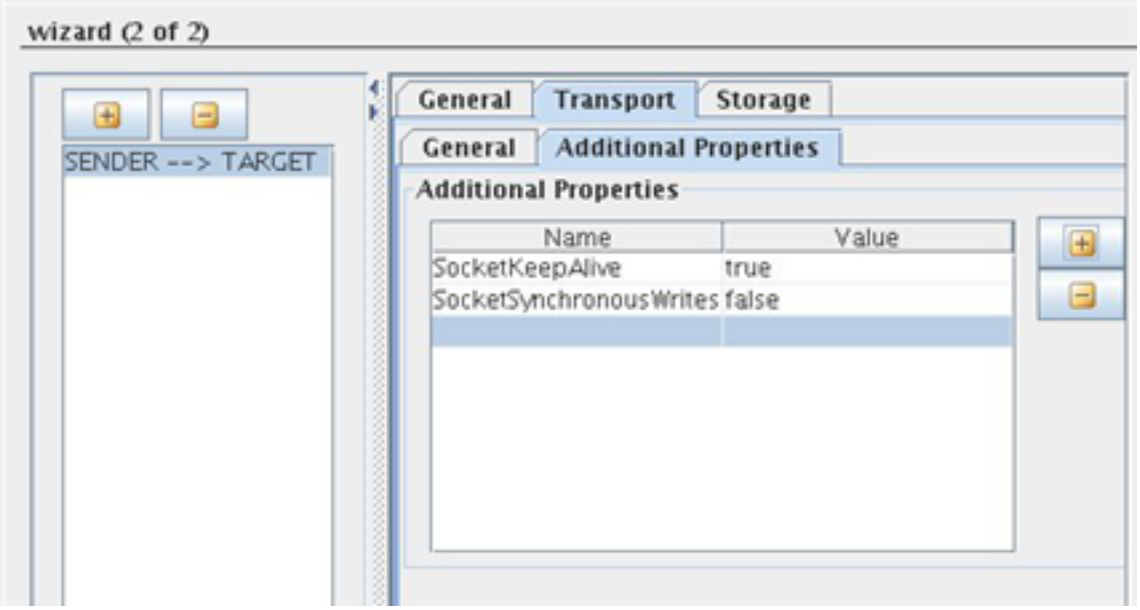


Figure 5: Additional Transport Configuration Properties

Additional properties that are used while creating socket connections. The properties which may be provided here are explained in Properties section.

4.15.2.1.3.3 Storage Configuration

Details regarding FIX sessions and messages can be persisted for further use such as in case of resend requests. This can be achieved by using either file or database storage. Storage Configuration is shown based on the storage type selected in the previous panel.

- **Enable Storage** - All the sessions that are created and messages that are processed by FIX engine will be stored for further use. If set to false, no messages will be persisted. This will force the engine to always send GapFills instead of resending messages. It can be set to false if resending a message is not required such as in case of market data streams.

File Store Configuration

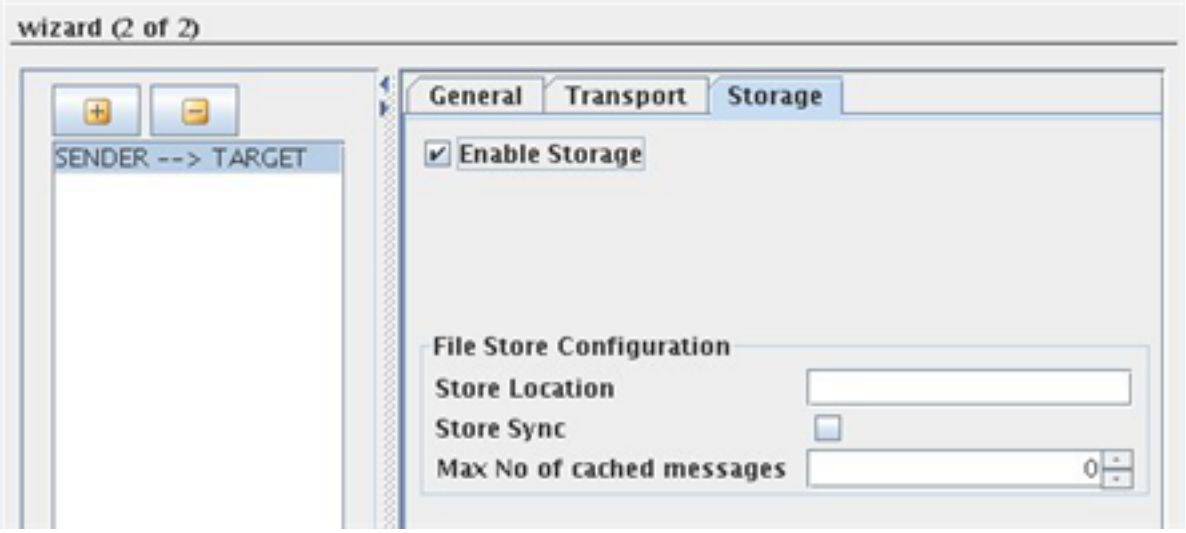


Figure 6: File Store Configuration

- **Store Location** - Directory to store sequence number, message and session files. The value should be a valid directory and must have write access.
- **Store Sync** - Boolean option for controlling whether the FileStore syncs to the hard drive on every write. It's safer to synchronize, but it's also much slower (100x or more slower in some cases).
- **Max. No. of cached messages** - Maximum number of message index entries to cache in memory. The value should be an Integer. A zero will not cache any entries.

Database Store Configuration

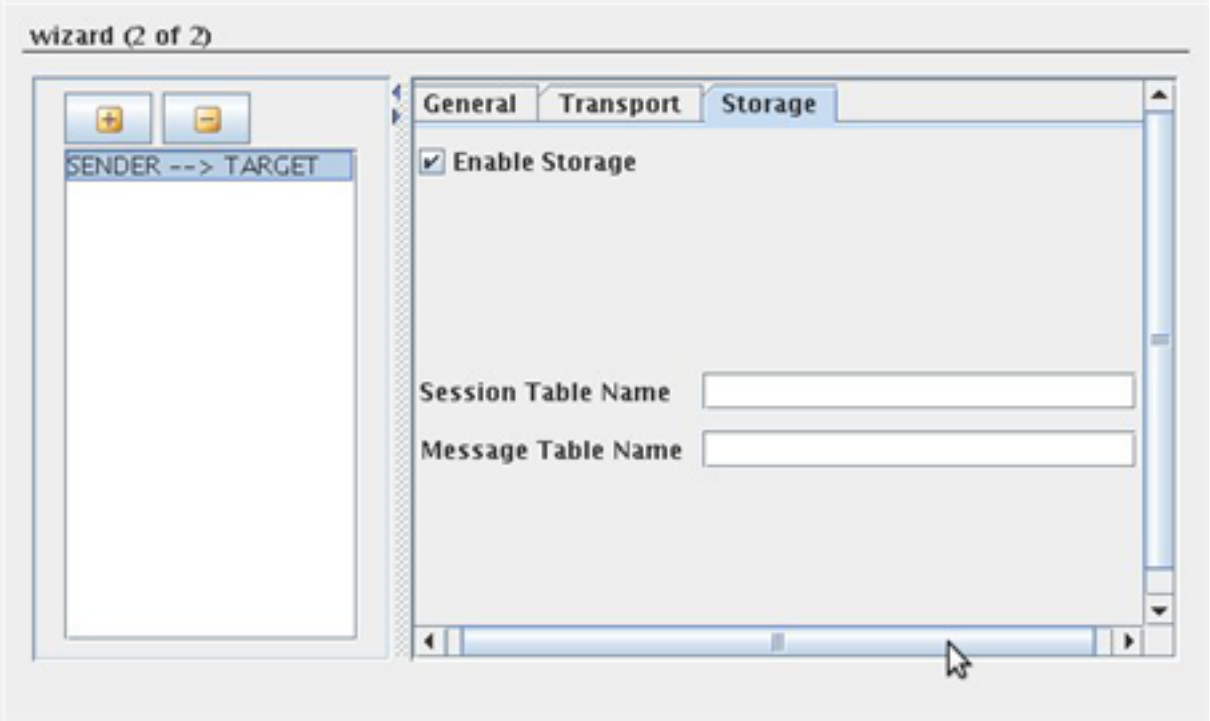


Figure 7: Database Store Configuration

- Sessions Table Name - Valid SQL table name for sessions table. If a table by this name does not already exist when the component is launched, the component will create a table with the name specified with following columns -

```
beginstring VARCHAR(8) NOT NULL,  sendercompid VARCHAR(64)
NOT NULL,
sendersubid VARCHAR(64) NOT NULL,  senderlocid VARCHAR(64)
NOT NULL,
targetcompid VARCHAR(64) NOT NULL,  targetsubid VARCHAR(64)
NOT NULL,
targetlocid VARCHAR(64) NOT NULL,  session_qualifier
VARCHAR(64) NOT NULL,           creation_time TIMESTAMP NOT
NULL,  incoming_seqnum INT NOT NULL,
outgoing_seqnum INT NOT NULL
```

  and primary key -

```
PRIMARY KEY (beginstring, sendercompid, sendersubid,
senderlocid, targetcompid, targetsubid, targetlocid,
session_qualifier)
```

- Messages Table Name - Valid SQL table name for messages table. If a table by this name does not already exist when the component is launched, the component will create a table with the name specified with following columns -

```
beginstring VARCHAR(8) NOT NULL,  sendercompid VARCHAR(64)
NOT NULL,
sendersubid VARCHAR(64) NOT NULL,  senderlocid VARCHAR(64)
NOT NULL,
targetcompid VARCHAR(64) NOT NULL,  targetsubid VARCHAR(64)
NOT NULL,
targetlocid VARCHAR(64) NOT NULL,  session_qualifier
VARCHAR(64) NOT NULL,           msgseqnum INT NOT NULL,
message VARCHAR(4000) NOT NULL
```

and primary key -

```
PRIMARY KEY (beginstring, sendercompid, sendersubid,
senderlocid, targetcompid, targetsubid, targetlocid,
session_qualifier, msgseqnum)
```

4.15.2.2 Properties

The properties that can be specified in all properties tables across the configuration are mentioned below. All the Boolean type properties' value should be specified as either 'true' or 'false'.

4.15.2.2.1 Session Properties

- **BeginString** - Version of FIX this session should use. Valid values are FIXT.1.1, FIX.4.4, FIX.4.3, FIX.4.2, FIX.4.1, FIX.4.0. Each session should have a BeginString specified in its properties.
- **SenderCompID** - Sender's compID as associated with this FIX session, a case-sensitive alpha-numeric string. SenderCompID is a required property for each session.
- **SenderSubID** - Sender's subID as associated with this FIX session, a case-sensitive alpha-numeric string.
- **SenderLocationID** - Sender's locationID as associated with this FIX session, a case-sensitive alpha-numeric string.
- **TargetCompID** - counterparty's compID as associated with this FIX session, a case-sensitive alpha-numeric string. TargetCompID is a required property for each session.
- **TargetSubID** - counterparty's subID as associated with this FIX session, a case-sensitive alpha-numeric string.
- **TargetLocationID** - counterparty's locationID as associated with this FIX session, a case-sensitive alpha-numeric string.
- **SessionQualifier** - Additional qualifier to disambiguate otherwise identical sessions.
- **DefaultApplVerID** - This must and should be specified only for FIXT 1.1 (and newer). Ignored for earlier transport versions. Specifies the default application version ID for the session. This can either be the ApplVerID enum (see the ApplVerID field in FIX specification) or the beginString for the default version.

For example:

Enum. FIX 5.0 over FIXT 1.1

DefaultApplVerID=7

BeginString: FIX 5.0 over FIXT 1.1

DefaultApplVerID=FIX.5.0

BeginString: FIX 4.2 over FIXT 1.1

DefaultApplVerID=FIX.4.2

- **HeartBtInt** - The interval, in seconds, at which the FIXInitiator sends heartbeat messages to a counter party. HeartbtInt is a required property for each session.
- **TimeZone** - Time zone for this session; if specified, the session start and end will be converted from this zone to UTC. Time zone ID (America/New_York, Asia/Tokyo, Europe/London, etc.)
- **StartTime** - Time of day that this FIX session becomes activated. Time should be specified in the format of HH:MM:SS [timezone]. The time zone is optional. The TimeZone setting will be used, if set, or UTC will be used by default. The timezone string should be one that the Java TimeZone class can resolve. For example, "15:00:00 US/Central". StartTime is a required property for each session.
- **EndTime** - Time of day that this FIX session becomes deactivated. Time should be specified in the format of HH:MM:SS [timezone]. The time zone is optional. The TimeZone setting will be used, if set, or UTC will be used by default. The timezone string should be one that the Java TimeZone class can resolve. For example, "09:00:00 US/Eastern". EndTime is a required property for each session.
- **StartDay** - For week long sessions, the starting day of week for the session. Use in combination with StartTime. Day of week in the default locale (e.g. Monday, mon, lundi, lun. etc.)
- **EndDay** - For week long sessions, the ending day of week for the session. Use in combination with EndTime. Day of week in the default locale (e.g. Monday, mon, lundi, lun. etc.)
- **MillisecondsInTimeStamp(Boolean)** - Boolean to determine if milliseconds should be added to timestamps. Only available for FIX.4.2 and greater. Default value is true.
- **ClosedResendInterval(Boolean)** - Boolean to determine whether to use actual end of sequence gap for resend requests rather than using "infinity" as the end sequence of the gap. Not recommended by the FIX specification, but needed for some counter parties. Default value is false.
- **ReconnectInterval** - Time between reconnection attempts in seconds.
- **LogonTimeout** - Number of seconds to wait for a logon response before disconnecting.
- **LogoutTimeout** - Number of seconds to wait for a logout response before disconnecting.

4.15.2.2.2 Validation Properties

- **ValidateFieldsOutOfOrder(Boolean)** - If set to false, fields that are out of order (i.e. body fields in the header, or header fields in the body) will not be rejected. Useful for connecting to systems which do not properly order fields. Default value is true.

- **ValidateFieldsHaveValues(Boolean)** - If set to false, fields without values (empty) will not be rejected. Useful for connecting to systems which improperly send empty tags. Default value is true.
- **ValidateUserDefinedFields(Boolean)** - If set to false, user defined fields will not be rejected if they are not defined in the data dictionary, or are present in messages they do not belong to. Default value is false.
- **AllowUnknownMsgFields(Boolean)** - Allow unknown fields in messages. This is intended for unknown fields with tags < 5000 (not user defined fields). Use ValidateUserDefinedFields for controlling validation of tags >= 5000. Default value is false.
- **CheckCompID(Boolean)** - If set to true, messages must be received from the counterparty with the correct SenderCompID and TargetCompID. Some systems will send different CompIDs by design, so this must be set to false. Default value is true.
- **CheckLatency(Boolean)** - If set to true, messages must be received from the counterparty within a defined number of seconds (see MaxLatency). It is useful to turn this off if a system uses local time for it's timestamps instead of GMT. Default value is true.
- **MaxLatency(Boolean)** - If CheckLatency is set to true, this defines the number of seconds latency allowed for a message to be processed. Default is 120. It should be a valid positive integer.

4.15.2.2.3 Socket Properties

FIXInitiator socket properties can be set in either common or per-session dictionary.

- **SocketKeepAlive(Boolean)** - When the keep alive option is set for a TCP socket and no data has been exchanged across the socket in either direction for 2 hours (NOTE: the actual value is implementation dependent), TCP automatically sends a keep alive probe to the peer. This probe is a TCP segment to which the peer must respond. One of the following three responses is expected -

  The peer responds with the expected ACK. The application is not notified (since everything is OK). TCP will send another probe following another 2 hours of inactivity.

  The peer responds with an RST, which tells the local TCP that the peer host has crashed and rebooted. The socket is closed.

  There is no response from the peer. The socket is closed. The purpose of this option is to detect if the peer host crashes.
- **SocketOobInline(Boolean)** - When the OOBINLINE option is set, any TCP urgent data received on the socket will be received through the socket input stream. When the option is disabled (which is the default) urgent data is silently discarded.
- **SocketReceiveBufferSize** - Set a hint the size of the underlying buffers used by the platform for incoming network I/O. When used in set, this is a suggestion to the kernel from the application about the size of buffers to use for the data to be received over the socket. A valid Integer value should be specified.
- **SocketReuseAddress(Boolean)** - Sets SO_REUSEADDR for a socket. This is used only for MulticastSockets in java, and it is set by default for MulticastSockets.

- **SocketSendBufferSize** - Set a hint the size of the underlying buffers used by the platform for outgoing network I/O. When used in set, this is a suggestion to the kernel from the application about the size of buffers to use for the data to be sent over the socket. A valid Integer value should be specified.
- **SocketLinger** - Specify a linger-on-close timeout. This option disables/enables immediate return from a close() of a TCP Socket. Enabling this option with a non-zero Integer timeout means that a close() will block pending the transmission and acknowledgement of all data written to the peer, at which point the socket is closed gracefully. Upon reaching the linger timeout, the socket is closed forcefully, with a TCP RST. Enabling the option with a timeout of zero does a forceful close immediately. If the specified timeout value exceeds 65,535 it will be reduced to 65,535. It should be an Integer value.
- **SocketTcpNoDelay(Boolean)** - Disable Nagle's algorithm for this connection. Written data to the network is not buffered pending acknowledgement of previously written data. Default value is true.
- **SocketTrafficClass** - Sets traffic class or type-of-service octet in the IP header for packets sent from this Socket. As the underlying network implementation may ignore this value applications should consider it a hint.

  The tc must be in the range 0 <= tc <= 255 or an IllegalArgumentException will be thrown.

  Notes: For Internet Protocol v4 the value consists of an octet with precedence and TOS fields as detailed in RFC 1349. The TOS field is bitset created by bitwise-or'ing values such the following -

  IPTOS_LOWCOST (0x02)

  IPTOS_RELIABILITY (0x04)

  IPTOS_THROUGHPUT (0x08)

  IPTOS_LOWDELAY (0x10)

  The last low order bit is always ignored as this corresponds to the MBZ (must be zero) bit.

  Setting bits in the precedence field may result in a SocketException indicating that the operation is not permitted.

  The property takes an integer value or a set of string options separated by "|" (e.g., "IPTOS_LOWCOST|IPTOS_LOWDELAY").
- **SocketSynchronousWrites(Boolean)** - Write messages synchronously. This is not generally recommended as it may result in performance degradation. The MINA communication layer is asynchronous by design, but this option will override that behavior if needed. Default value is false.
- **SocketSynchronousWriteTimeout** - The time in milliseconds to wait for a write to complete. An Integer value should be specified. Defaults to 30000 ms if SocketSynchronousWrites is true.

4.15.2.2.4 Storage Properties

JdbcSessionIdDefaultPropertyValue - The default value for Session ID bean properties (namely BeginString, SenderCompID, SenderSubID, SenderLocationID, TargetCompID, TargetSubID, TargetLocationID) is an empty string. Oracle treats this as a SQL NULL and that causes problems. This configuration setting allows to set the default value for unspecified Session ID properties. This property takes any nonempty string value.

4.15.2.2.5 Miscellaneous Properties

- **ResetOnLogon(Boolean)** - Determines if sequence numbers should be reset before sending/receiving a logon request. Default value is false.
- **ResetOnLogout(Boolean)** - Determines if sequence numbers should be reset to 1 after a normal logout termination. Default value is false.
- **ResetOnDisconnect(Boolean)** - Determines if sequence numbers should be reset to 1 after an abnormal termination. Default value is false.
- **ForceResync(Boolean)** - When logout response contains "expecting XXX", the sequence numbers of forcibly resynchronized. Default value is false.
- **NonStopSession(Boolean)** - Session scheduling setting to specify that session never ends/resets. Default value is false.
- **ContinueInitializationOnError(Boolean)** - Continue initializing sessions if an error occurs. Default value is false.
- **SendRedundantResendRequests(Boolean)** - Allows sending of redundant resend requests. Default value is false.

4.15.2.3 Functional Demonstration

Configure the FIXInitiator as described in Configuration section and send sample input to the component.

4.15.2.3.1 Scenario 1

Configure FIXInitiator and FIXAcceptor with TCPIP transport type.

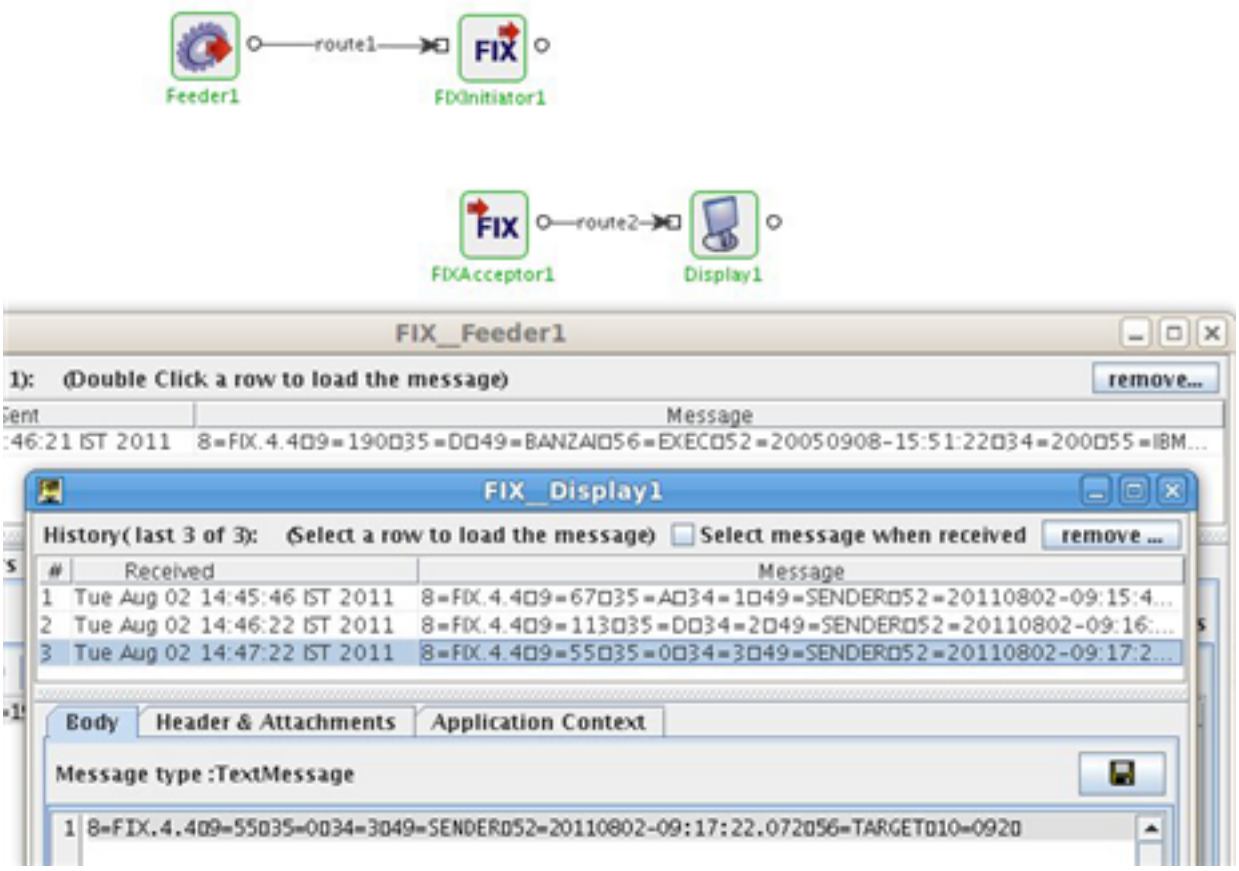


Figure 8: TCPIP Transport Type scenario

FIXInitiator sends Logon message (35=A) as soon as it is started to establish a session using FIX Session protocol. It then sends a NewOrderSingle message (35=D) for FIXAcceptor to execute the order. When FIXInitiator is not sending any messages like order requests to FIXAcceptor, it keeps pinging the FIXAcceptor by sending Heartbeat messages (35=0) for keeping the session alive.

4.15.2.3.2 Scenario 2

Configure FIXInitiator and FIXAcceptor with JMS transport type.

Figure 9: JMS Transport Type scenario

Communication between FIXInitiator and FIXAcceptor is similar to Scenario 1 with the only difference being the medium. Here, the communication is done over JMS. FIXInitiator sends admin messages like logon, heartbeat and application messages like order requests to FIXAcceptor via JMS. FIXAcceptor also sends admin messages to FIXInitiator in the same way.

4.15.2.4 Useful Tips

After FIXInitiator is started, it sends a FIX session Logon message for each session configured to a counter party and awaits a response. If a response is not received within time limit specified by LogonTimeout property, the session is disconnected.

In case of JMS transport type, inorder for FIXAcceptor not to send application messages back to FIXInitiator and to limit the communication from FIXAcceptor to FIXInitiator to just admin messages, a JMS selector (ADMIN_MSG_PROPERTY IS NOT NULL) should be specified on route between outport of FIXAcceptor and inport of FIXInitiator (route3 as shown in the image above).

Chapter 5: Event Processes

This chapter discuss about Event Processes. Event processes are composite applications created as event-driven assemblies of service components. They represent the orchestration of data flow across customized service-components distributed across the ESB network.

5.1 What are Event Processes?

Event processes in Fiorano are designed to connect disparate applications in a heterogeneous distributed SOA environment.

Event processes are designed using the Fiorano Studio. Fiorano Studio enables intuitive visual configuration of all the elements of an event process including the components of the process, the data flow or routes between components, deployment and profile information and the layout. The event-process meta-data contains all required information in XML format and is stored in the service repository managed by the Fiorano Enterprise Server.

5.2 Creating Event Processes

Fiorano's unique model for composition of event processes allows the logical process design to be mapped directly to physical service components distributed across the ESB network.

Event processes are designed by 'drag-drop-connect' of service components. The components are customized by configuration rather than custom code. The routes between components are drawn by visually connecting the component ports. Every component instance in the flow can be configured to be deployed on different nodes of the ESB network.

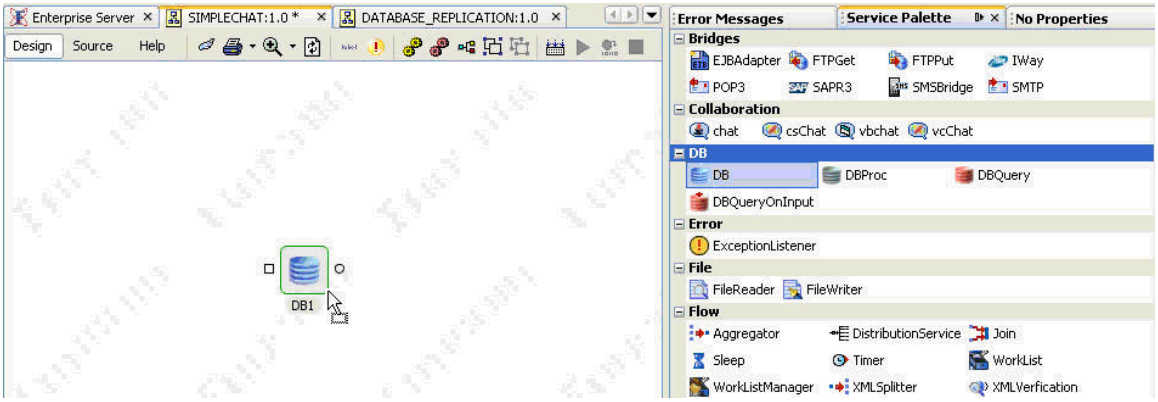


Figure 5.2.1: Drag-drop service component

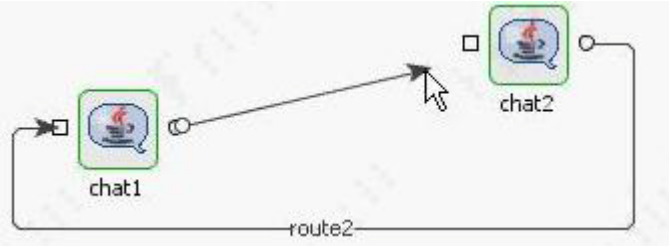


Figure 5.2.2: Orchestrating service components

The composition model of event processes is rooted on intuitive understanding and high levels of abstraction. The composition model along with the complementing service component architecture allows business users to design, deploy, and manage event driven business processes in a single view with no separation between the development (composition) and deployment steps as is customary in most SOA platform products.

5.2.1 Creating a New Event Process

1. Launch the Studio, and login to the Enterprise Server.
2. Browse the **Event Process Repository** as illustrated in Figure 5.2.3.
3. Right-click on **User Processes** and choose **Add Event Process** from the pop-up menu.

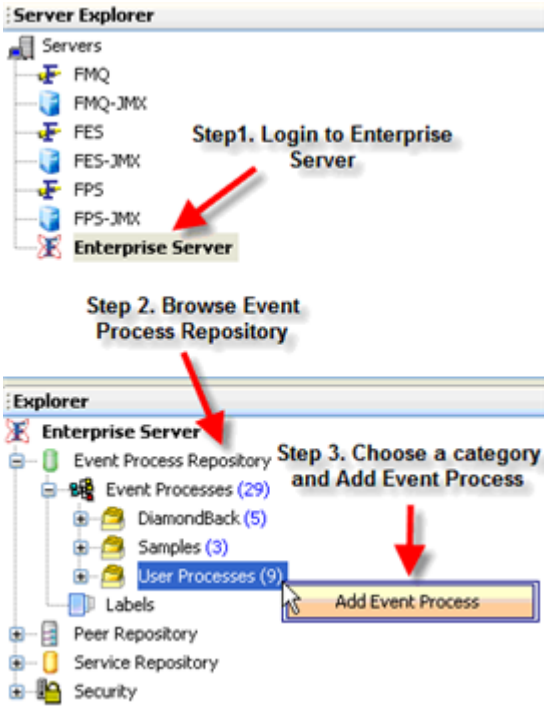


Figure 5.2.3: Creatind an Event Process

Using External Components

The Fiorano Studio allows users to compose an event process with service component references from external event processes. This enables inter process communication when designing modularized, interdependent event processes.

Figure 5.2.4 illustrates steps to use an external service component in the event process.

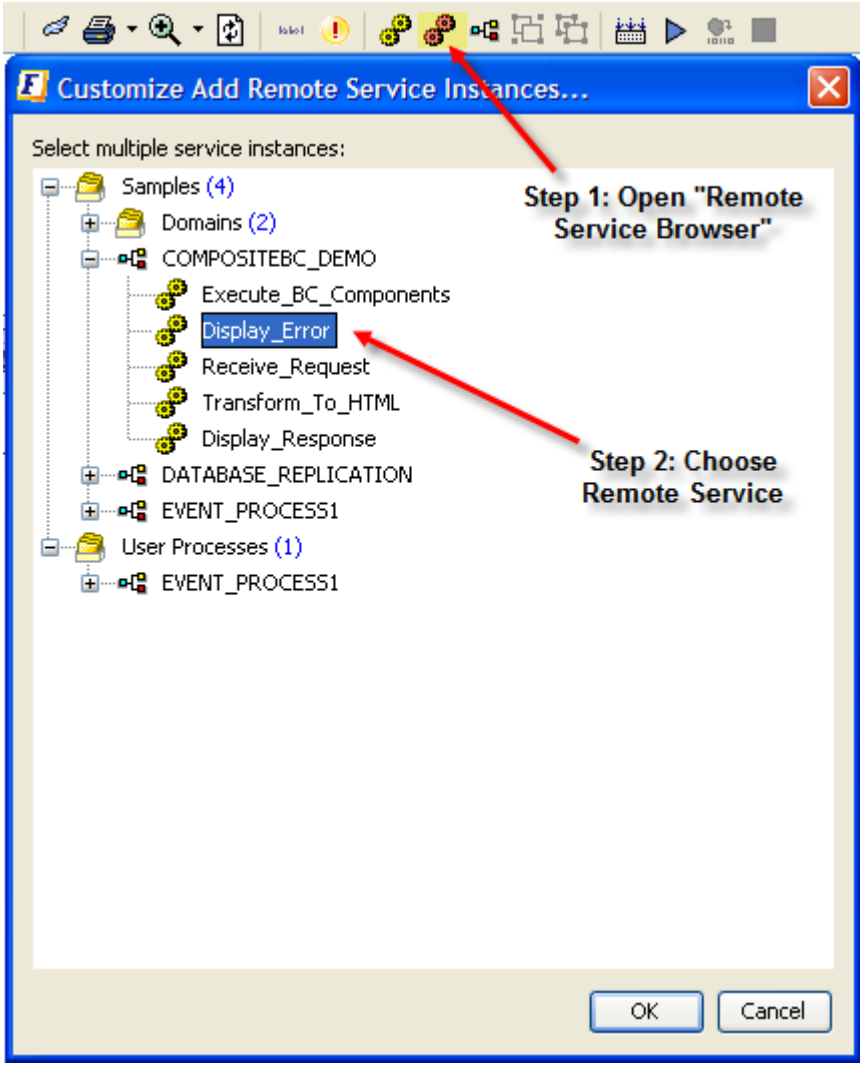


Figure 5.2.4: Add Remote Service Component in an Event Process

5.3 Configure Event Processes

Configuring an event process involves configuring the component instances, routes, and other deployment parameters of the process.

5.3.1 Configuring Components through Custom Property Sheet

Components instances can be customized using the associated custom property sheets for each component. To review the custom property sheet associated with any component, simply double click the component in the application easel.

5.3.2 Configuring Common Component Properties

Apart from the component specific properties that can be configured using the CPS, there are a set of common properties associated with the every component, Figure 5.3.1 illustrates the component properties detail.

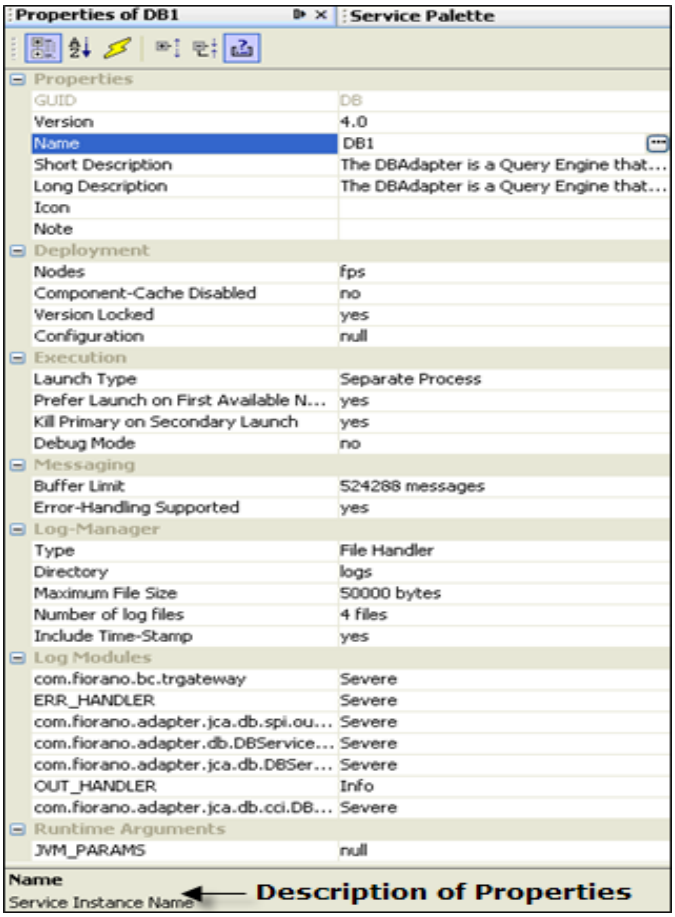


Figure 5.3.1: Component Properties

The description of individual properties is shown at the bottom of the panel.

5.3.3 Adding Additional Jars/Libraries to Components

Service Components can be upgraded to use new libraries during configuration and at runtime.

5.3.4 Setting up Component Port Properties

Component port properties can be configured by choosing the port and configuring the appropriate property in the Properties panel.

To configure the output port of a service component, simply click the output port in the application easel and then set the appropriate properties in the RHS properties pane, as shown in Figure 5.3.2.

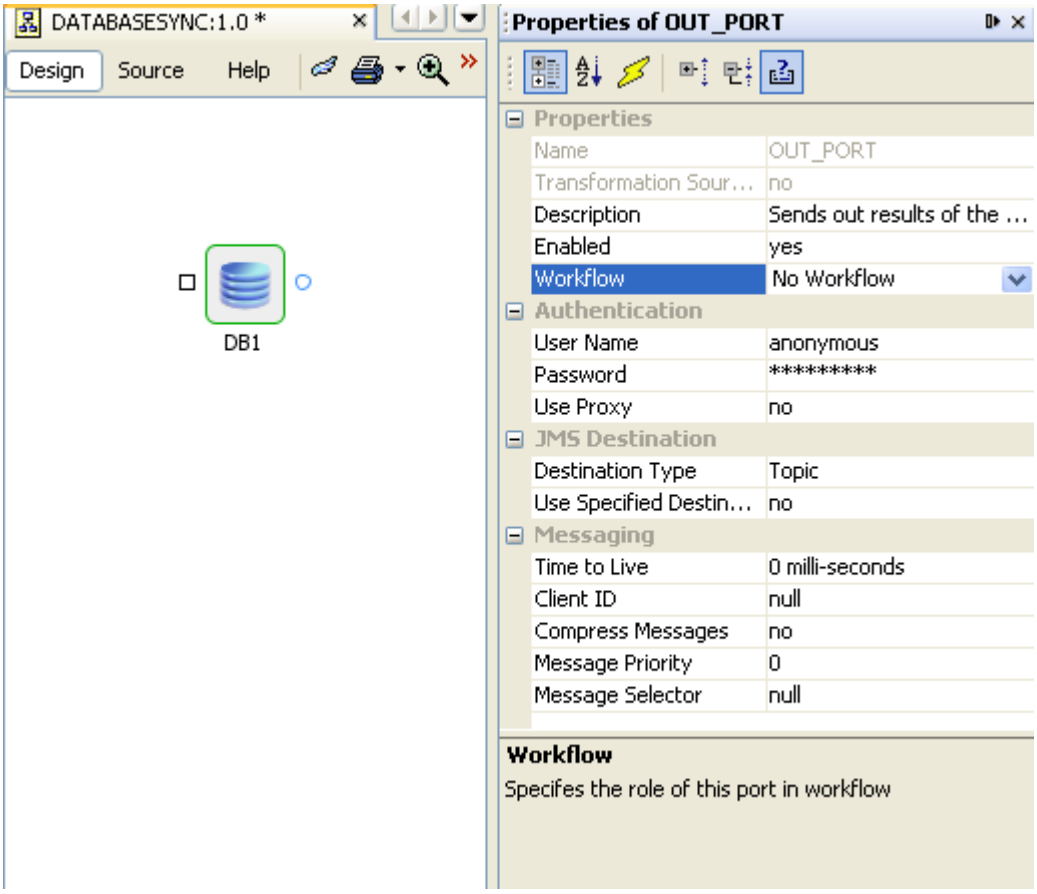


Figure 5.3.2: Configuring Output Ports

Similarly, to configure an input port of a service component instance, select the port in the application easel and set the property as illustrated in Figure 5.3.3.

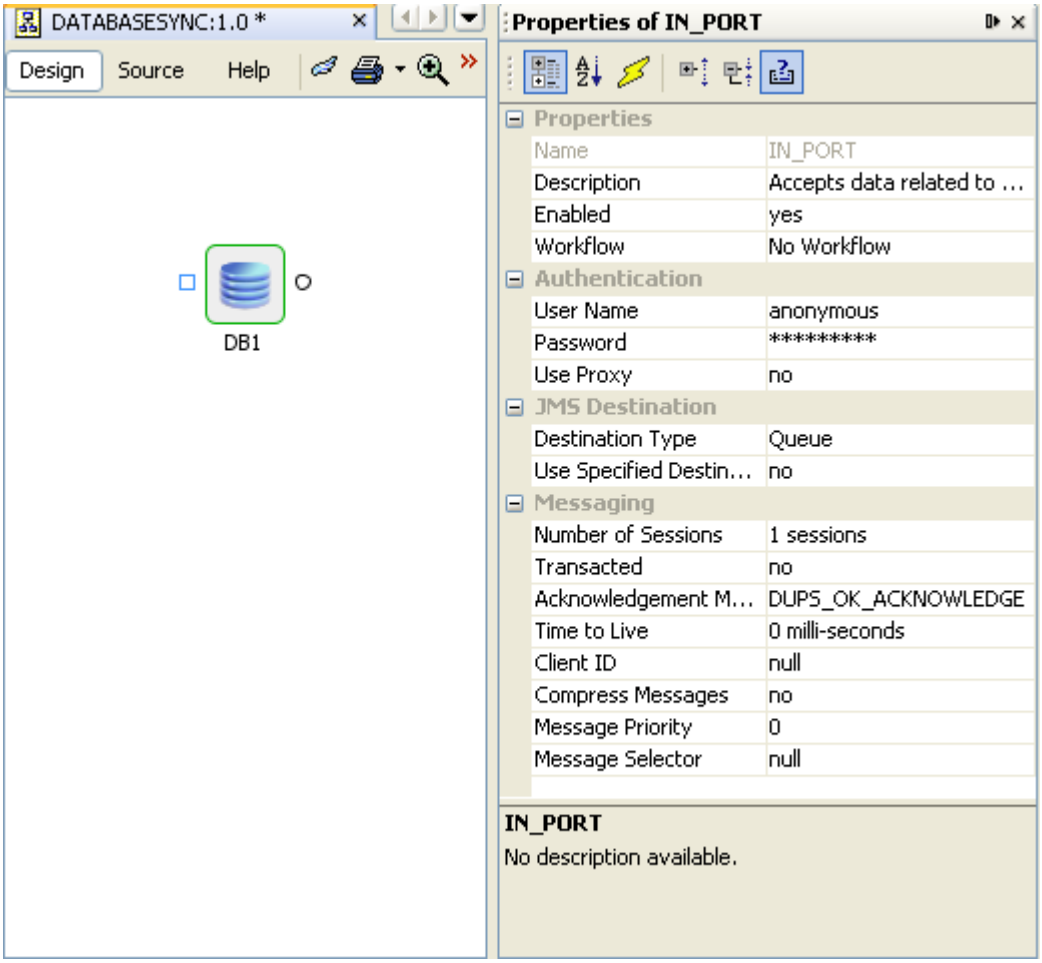


Figure 5.3.3: Configuring Input Port of a Service Component

5.3.5 Defining Data Transformation

XML to XML or native transformations can be achieved using Fiorano's transformation service component. This includes an associated transformation tool that allows users to configure complex data transformations visually without writing custom code.

5.3.6 Defining Exception Flows

Handling exceptions in a uniform way is one of the critical aspects of designing business processes on the Fiorano platform. This can be achieved using the following methods.

5.3.6.1 Using the Exception Listener Service Component:

This component serves as a global exception listener that subscribes to exception messages occurring in any service component of any running event process and handles them appropriately in the error handling event process.

Figures 5.11 and 5.12 illustrate how the exception listener is configured and used in a global exception handler event process.

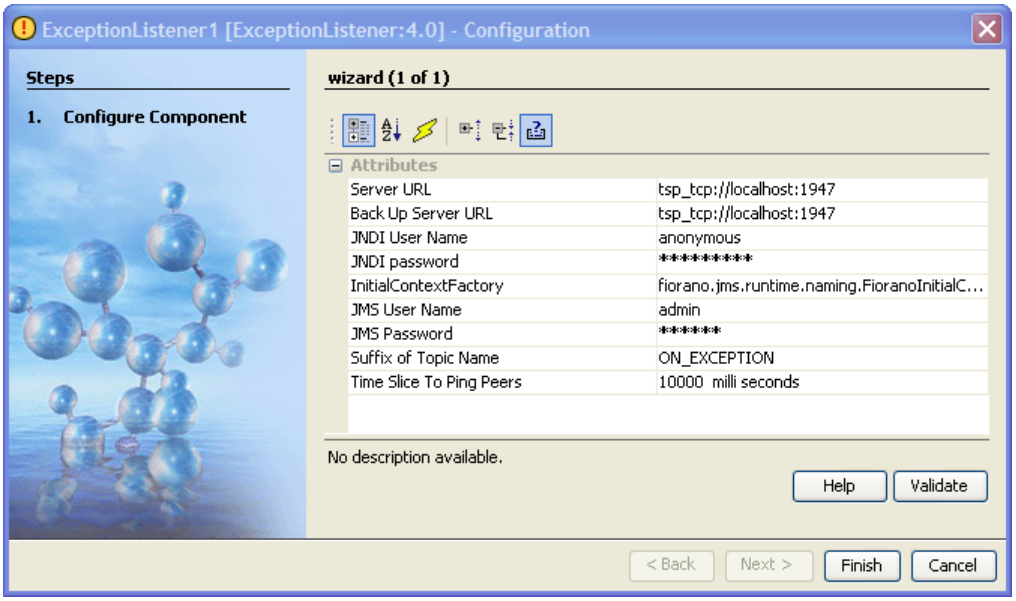


Figure 5.3.4: Configuring the Exception Listener

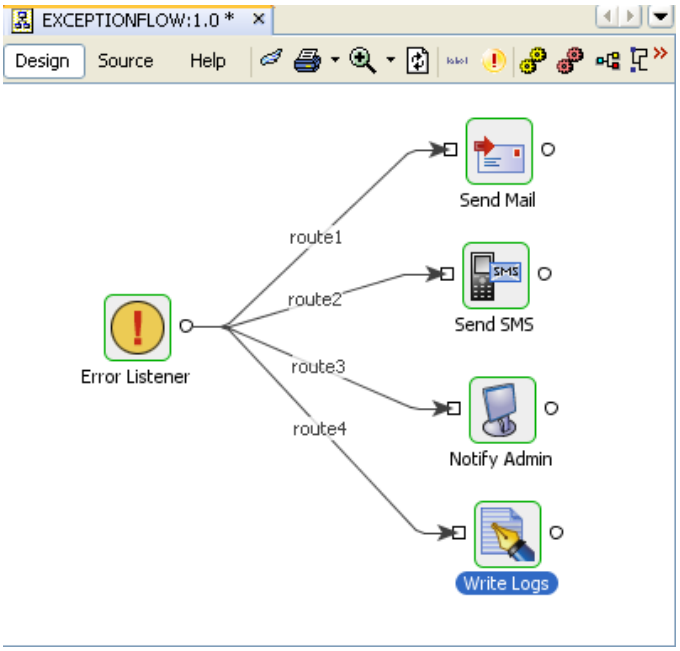


Figure 5.3.5: Designing a Global Exception Handler Event Process

5.3.7 Using the Error Ports View

The error ports view in any event process allows you to see the error ports associated with the service components. Event-process specific or component-instance specific error handling can be configured by connecting the error ports of components to the appropriate error handling component or event process.

Figure 5.3.6 illustrates the use of error ports to send email notifications on error.

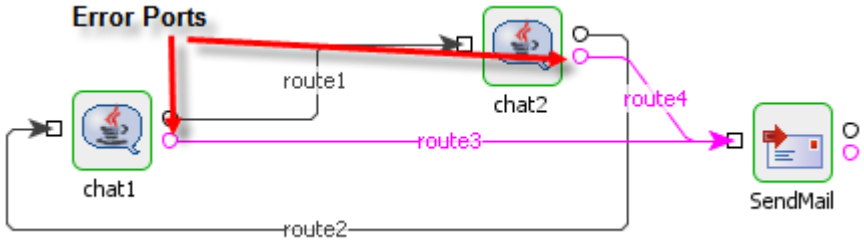


Figure 5.3.6: Using the Error Ports

The SendMail component instance of Figure 5.3.6 may be replaced by a general error handler, as illustrated in Figure 5.3.7.

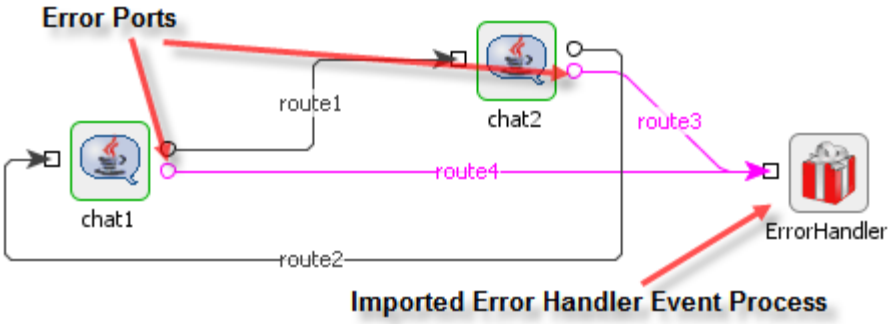


Figure 5.3.7: Sending Error Message

5.3.8 Document Tracking

Tracked document flows are defined by assigning multiple workflow items and one workflow end at the corresponding component ports. All the documents that pass through the workflow items till the workflow end are logically grouped together as a single workflow and stored into the DB. Therefore each message send across a defined workflow is treated as a single workflow entry. A workflow entry can have multiple documents (one for each workflow item/end). The document contains the actual message that reaches the port (or a part of it, depending on the configuration).

5.3.8.1 Configuring Document Tracking

Figure 5.3.8 illustrates how to configure Document tracking in the component ports.

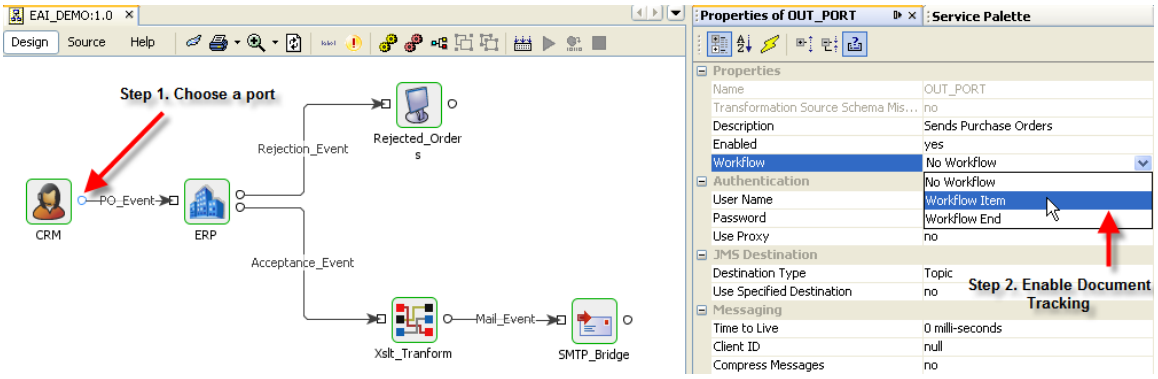


Figure 5.3.8: Component Port Document Tracking

Document tracking works as follows: whenever data traverses a port marked Workflow Item (up to and including the Workflow End), it is captured and stored in a database on the Fiorano Enterprise Server. Document tracking thus enables users to keep a log of all messages flowing on the network. The tracked documents can be viewed on a per-application basis using the Fiorano Event Manager tool. For more details, see section 5.7 Monitoring Event Processes.

5.3.8.2 Configuring Specific Database

The document tracking feature is configured as part of FES to track SBW events into H2 database running within Enterprise Server. This can be changed from Profile Management in Fiorano eStudio. For more details, see section 17.5.1 in Fiorano eStudio User Guide.

5.3.8.3 Database Table Structure

The exact schema of the tables varies from database to database according to the configurations provided in *<dbtype>_jdbc.cfg file*. An explanation of the tables and the various fields for the older and newer schema are given below:

Table name: WF_INST_EVENT_HISTORY

Column Name	Type (Before SOA 9.0.0)	Type (From SOA 9.0.0)	Description
EVENT_ID	INTEGER	INTEGER	Auto Generated
WORKFLOW_INSTANCE_ID	VARCHAR(255)	VARCHAR(255)	Auto Generated
WORKFLOW_ID	VARCHAR(255)	VARCHAR(255)	GUID of the corresponding Application
USER_DEFINED_DOC_ID	VARCHAR(255)	VARCHAR(255)	Can be set by the user
SERVICE_INST_ID	VARCHAR(255)	VARCHAR(255)	Service instance

Column Name	Type (Before SOA 9.0.0)	Type (From SOA 9.0.0)	Description
			name
STATE_ID	VARCHAR(255)	VARCHAR(255)	Port name
STATE_COMMENT	VARCHAR(255)	VARCHAR(255)	Description of the workflow item
STATE_EVENT_DATE	VARCHAR(255)	VARCHAR(255)	Date at which message reached the port
DOCUMENT_ID	VARCHAR(255)	VARCHAR(255)	Auto Generated
WORKFLOW_STATUS	VARCHAR(255)	VARCHAR(255)	EXECUTED or EXECUTING
IN_TIME	TIMESTAMP	TIMESTAMP	Set if its inport
OUT_TIME	TIMESTAMP	TIMESTAMP	Set if its outport
TOTAL_TIME	VARCHAR(255)	VARCHAR(255)	Time spend between inport and outport
DOCUMENT	**NA**	IMAGE	BLOB of actual message data

Table: DOCUMENT_ARCHIVE (This table is not present in current version of the product)

Column Name	Type	Description
WORKFLOW_ID	VARCHAR(255)	GUID of the corresponding Application
WORKFLOW_INSTANCE_ID	VARCHAR(255)	Refers to the corresponding parent entry
USER_DEFINED_DOC_ID	VARCHAR(255)	Can be set by the user
DOCUMENT_ID	VARCHAR(255)	Refers to the corresponding parent entry
DOCUMENT	IMAGE	BLOB of actual message data

In case, an earlier version of the product (version before SOA 9.0.0) was using a database that is being used by the latest version, sbw database tables will be automatically modified to accommodate these changes while the Enterprise server starts up. It is important to note that once a database is converted to newer version; it cannot be used by older version of the product.

The views presented in EVST and dashboard uses the above schema in logical groupings.

EVST View

Application view: Lists all workflows for a given application.

Workflow: All entries in event history table for a given workflow ID are grouped together as a workflow. The status of the workflow is the status of the entry with the latest time stamp. The cycle time is the sum of all total times.

Document view: Lists all the documents for a given workflow. This contains one item per entry in the event history table. The details have one-to-one correspondence.

Document: Shows the actual message content. This provides a view of the BLOB data present in archive table.

5.3.8.4 Structure of IMAGE/BLOB field

The IMAGE/BLOB field is a serialized form of an object of class fiorano.jms.services.msg.def.FioranoMessage (packaged in $FIORANO_HOME/fmq/lib/common/fmq-common-msg-impl.jar). To de-serialize the field, a user may use an API available in Enterprise Server's SBW module. Please refer to the sample named 'SBWDataReader.java' located under $FIORANO_HOME/esb/samples/DocTracking which provides sample usage of this API.

Following information is available in a BLOB field.

- Document Information: All the other fields of the document tracking table as explained above in section 5.3.8.3. For example, Source Peer Name, Event Process Name, Service Instance Name, Port Name, In Time, Out Time, Document ID, Workflow Instance ID etc.
- Header Information: Other message header properties represented by a java.util.HashMap object of property name vs. property value.
- Carry Forward Context: This information is present as an object of type fiorano.esb.util.CarryForwardContext. This object contains the following information:
  - Application Context (if defined)
  - Carry Forward Properties (The message properties carried forwarded from the message received at last doc tracked port)
  - An Enumeration of fiorano.esb.util.SourceContext containing information about the output port of the components from where the message has traveled so far.
- Attachment(s): This information is present as an object of type java.util.Hashtable<String, byte[]>. The String part represents the attachment name and byte[] represents the contents of the file in byte[] form.
- Message Text: Message Text can be retrieved using the API named MesageUtil.getTextData(message) which returns a String containing the message text.

5.3.9 Message Selector on Route

Message selectors can be defined on the event routes to allow/disallow the messages based on the specified condition. User can define following types of message selectors on the route:

- Sender Selector

- JMS Selector
- Message Body XPath Selector
- Application Context XPath Selector

Sender Selector: Filters the messages based on the source/traversed components. Allows messages sent or forwarded by the components specified in the condition. Condition can be specified by selecting single or multiple components.

JMS Selector: Filters the messages based on the JMS headers. Allows messages that contain the headers matching the specified the condition. Conditions should be specified using SQL-92 syntax.

Message Body XPath Selector: Filters the messages based on the message body. Allows messages whose message body content matches the specified XPath condition. Conditions should be specified using valid XPath expression. If the source port of the route is associated with the schema then user can use the XPath Editor to specify the condition else need to specify the condition manually.

Application Context XPath Selector: Filters the messages based on the application context. Allows messages whose application context content matches the specified XPath condition. Conditions should be specified using valid XPath expression. If the source port of the route is associated with the schema then user can use the XPath Editor to specify the condition else need to specify the condition manually.

Note: Incase there are multiple selectors defined on a routes then the message that matches all the condition will only be allowed.

5.3.9.1 Defining Message Selector on Route

Following are the steps to define various types of message selectors on route using Fiorano Studio:

1. Right-click on **route** and select **Selectors**, as shown in the Figure 5.3.9.

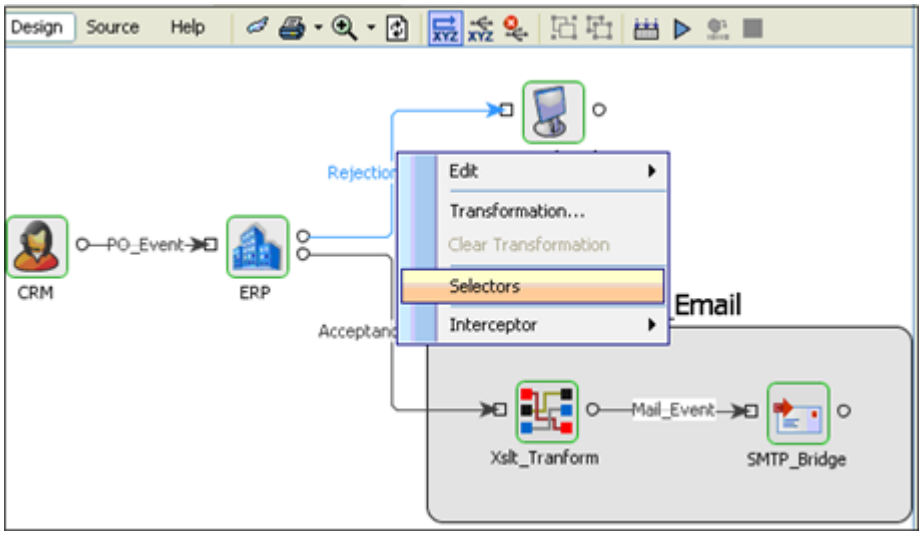


Figure 5.3.9: Message selector from route

2. Click **Add** and choose Selector type which needs to be added from the drop-down list as shown in Figure 5.3.10.
3. Specify the selector condition as described in the section 5.3.9 Message Selector on Route.

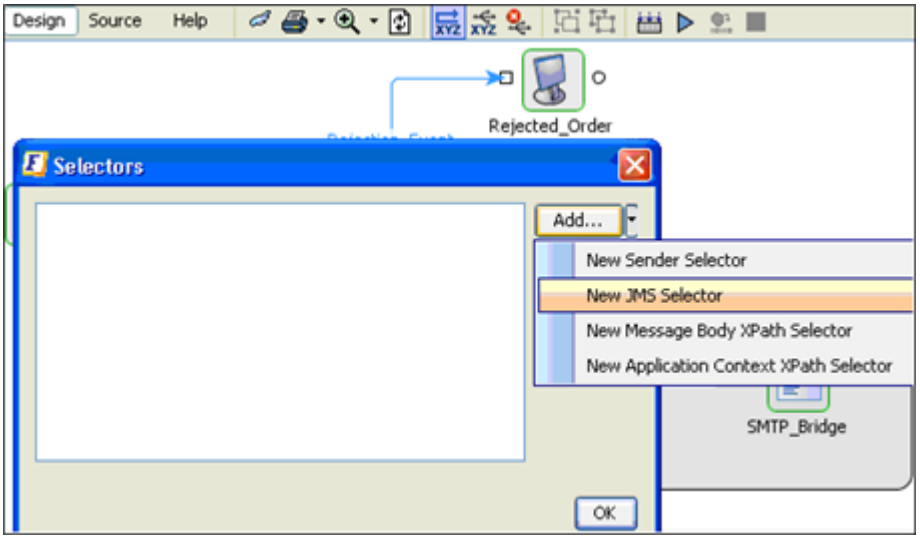


Figure 5.3.10: Adding New JMS Selector

5.3.10 Setting Alerts and Notification

Fiorano SOA provides you the facility to set mail alerts and notification, you can set SMTP Alert, JMS Alet, and SMS Alert. To set alerts, perform the following steps:

Note: Alerts can be sent to mail servers which do not require user authentication for sending mails.

9. Open **Studio** and open **FES** profile.
10. Now navigate to, FES>Fiorano>Esb>Controller>SystemEvent.
11. Expand the **SystemEvent** and select one of the component instances from the list, for example, right-click SECURITY_VIOLATION and select **Add**, now click on **SMTPAlert**, or **JMSAlert**, or **SMSAlert** from the pop-up menu. You can also add all the three alert types.

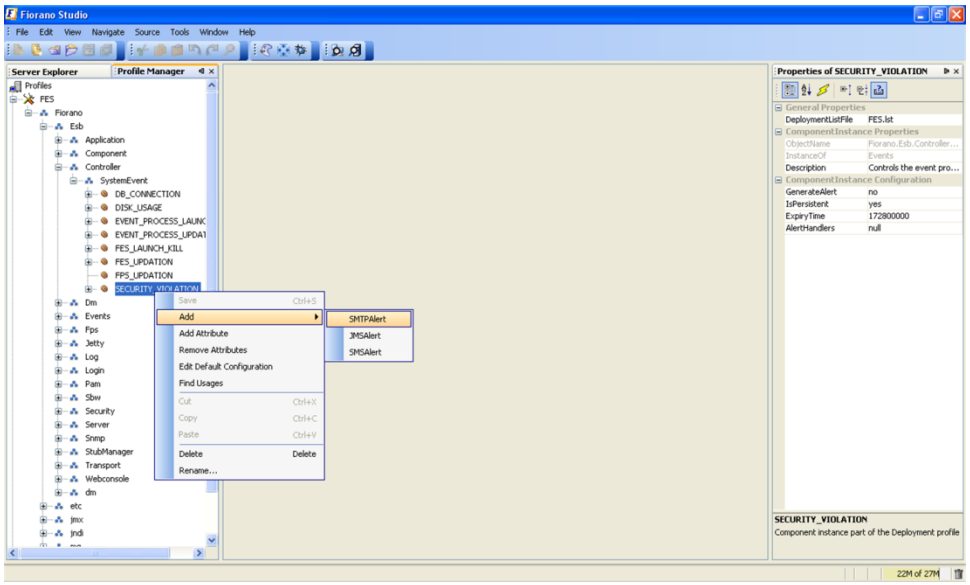


Figure 5.3.11: Adding new SMTPAlert

12. Now you can set the properties of the selected alert from the **Properties** Pane.

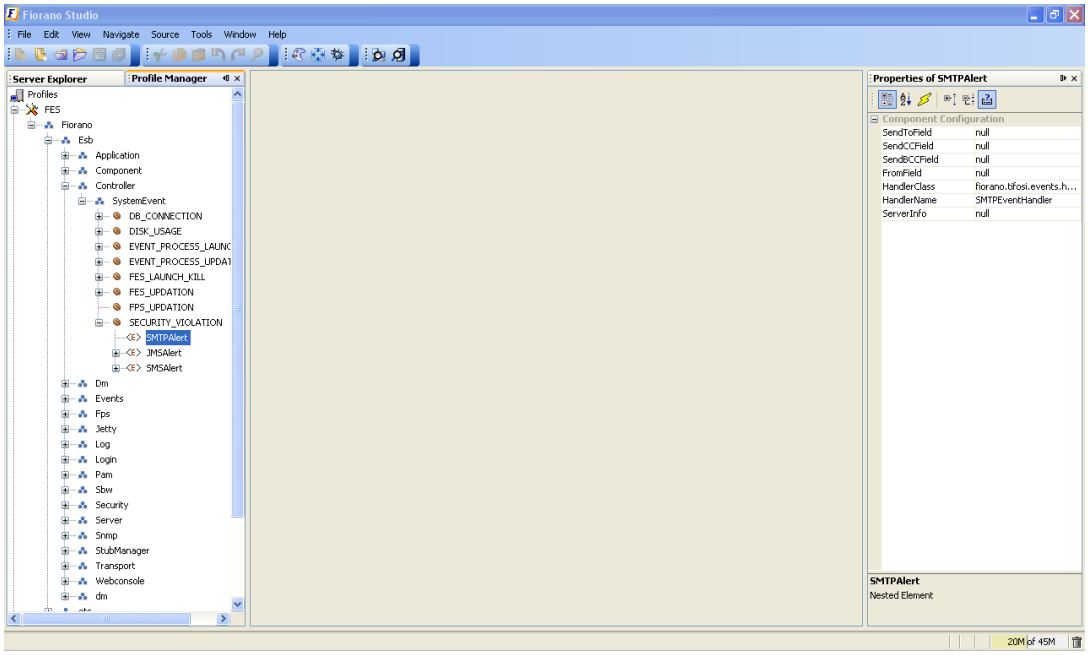


Figure 5.3.12: Setting the Properties

13. Save the profile to apply the changes.

5.3.11 Configuring the Application Context

While simple data transformation from source Event Port to target ports is a necessity, there are times when a target Event Port needs information which was produced by a Business component that occurred before in the workflow. Consider a Fiorano SOA Platform event process representing a ten-step business process. Each step is implemented using a Business component. By using application context you can enable a Business component, representing the tenth step to use information generated by the second Business component.

To configure the Application Context, perform the following steps:

1. Login to Enterprise Server and open an **Event Process** you want to configure.
2. Now in the **Structure** pane, right-click the open **Event Process** and click **Add Application Context** from the pop-up menu as shown in the Figure 5.3.13. The **Application Context** is now added under the **Event Process** tree in the Structure pane.

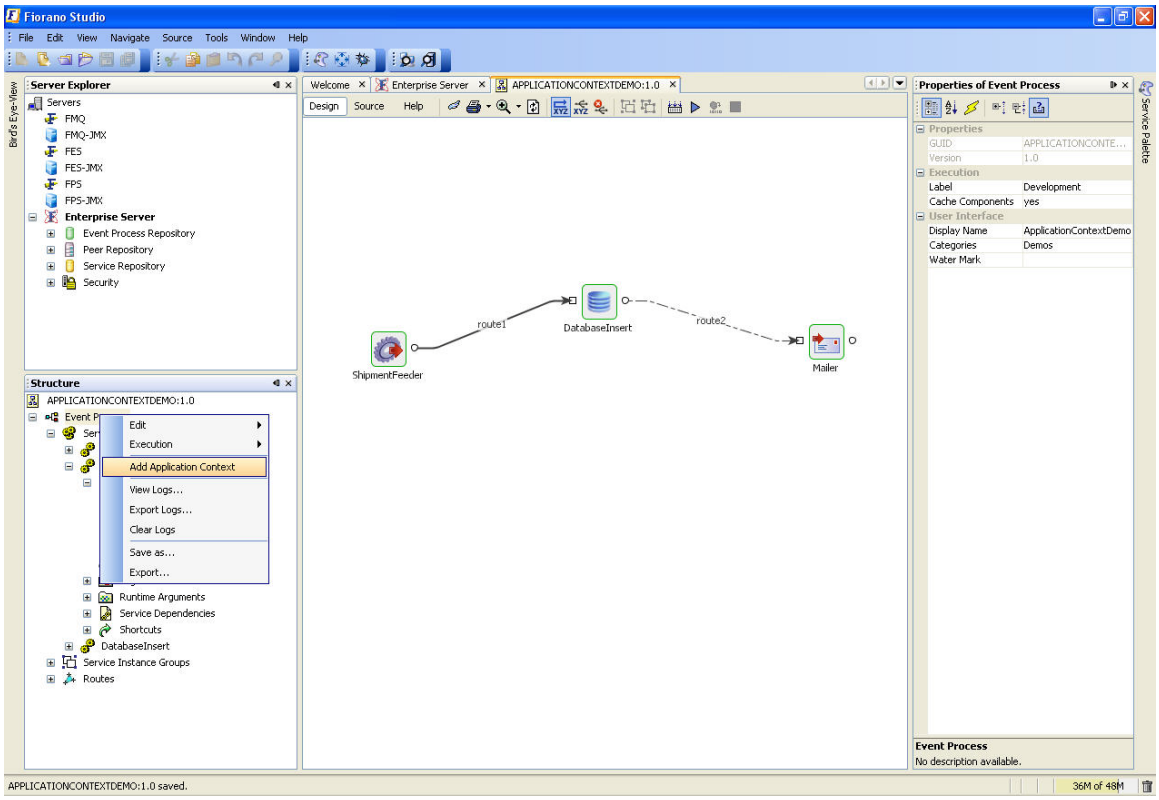


Figure 5.3.13: Adding Application Context

3. In the **Application Context** property, select the appropriate xml schema type (DTD or XSD) and give a valid schema in the content as shown in the Figure 5.3.14.

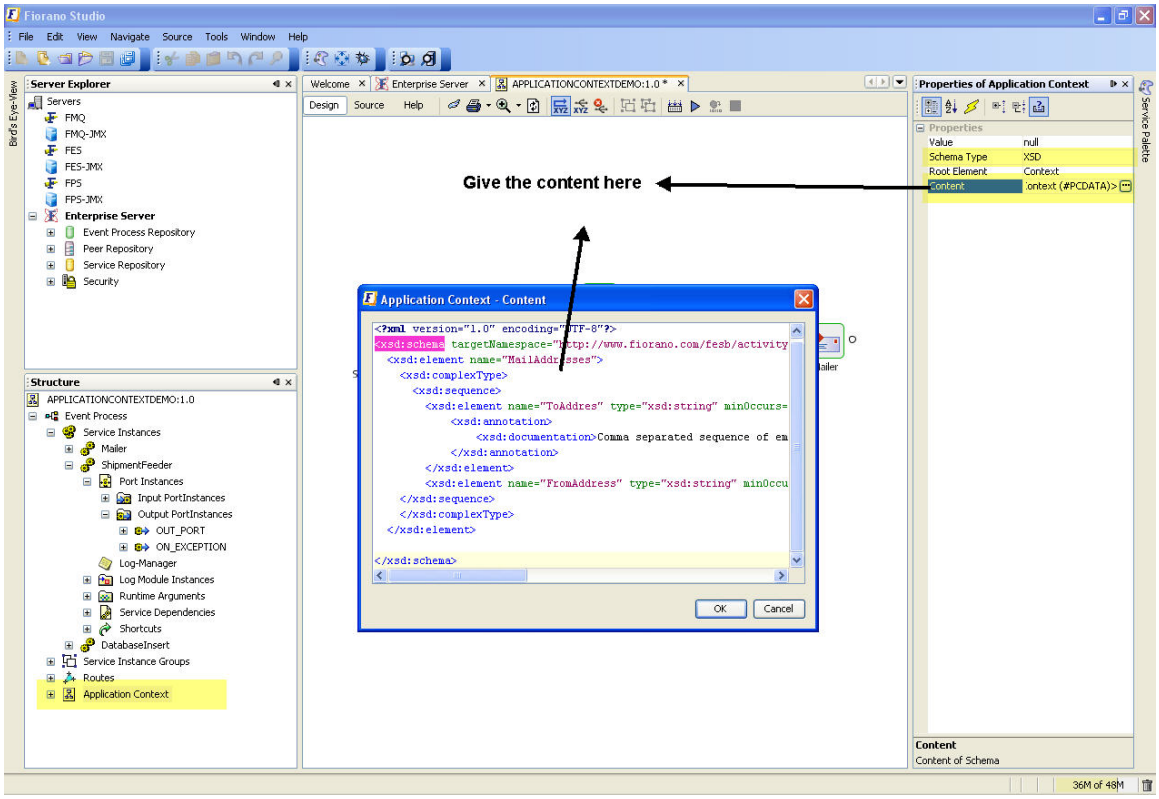


Figure 5.3.14: Application Context property panel

4. When this is done, the application context is available through out the event process. Default value can be provided by giving the xml in the 'value' of the 'application context' properties. If no default value is provided, empty xml is present all through the application, unless it is configured at any of the out ports. Once **Application Context** is configured at one of the out ports, the value is propagated in the message flow. To configure 'Application Context' at any of the outports, right-click on the port and select **Application Context** as shown in the Figure 5.3.15. The **Fiorano Mapper** tool window appears.

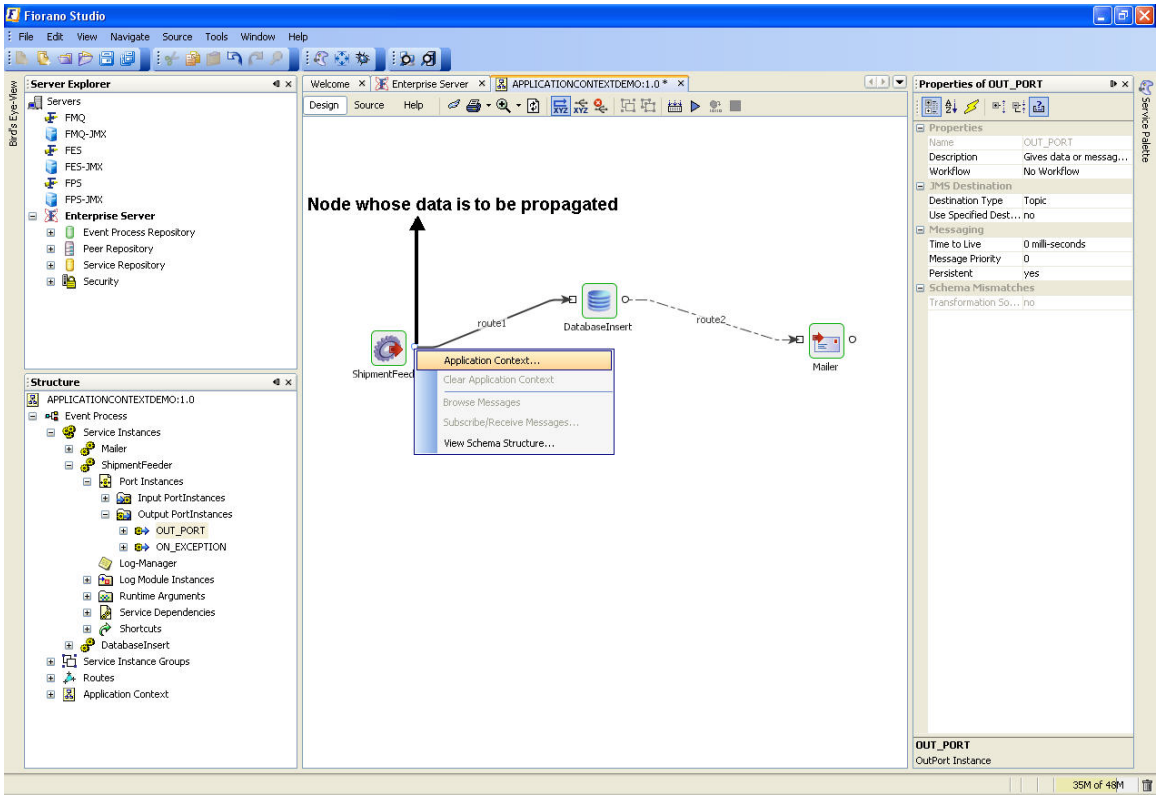


Figure 5.3.15: Fiorano Mapper tool dialog box

5. The Mapper window shows the output port structure and the application context structure. The data that is to be propagated from this port and is to be available all through the event process can now be configured as shown in the Figure 5.3.16.

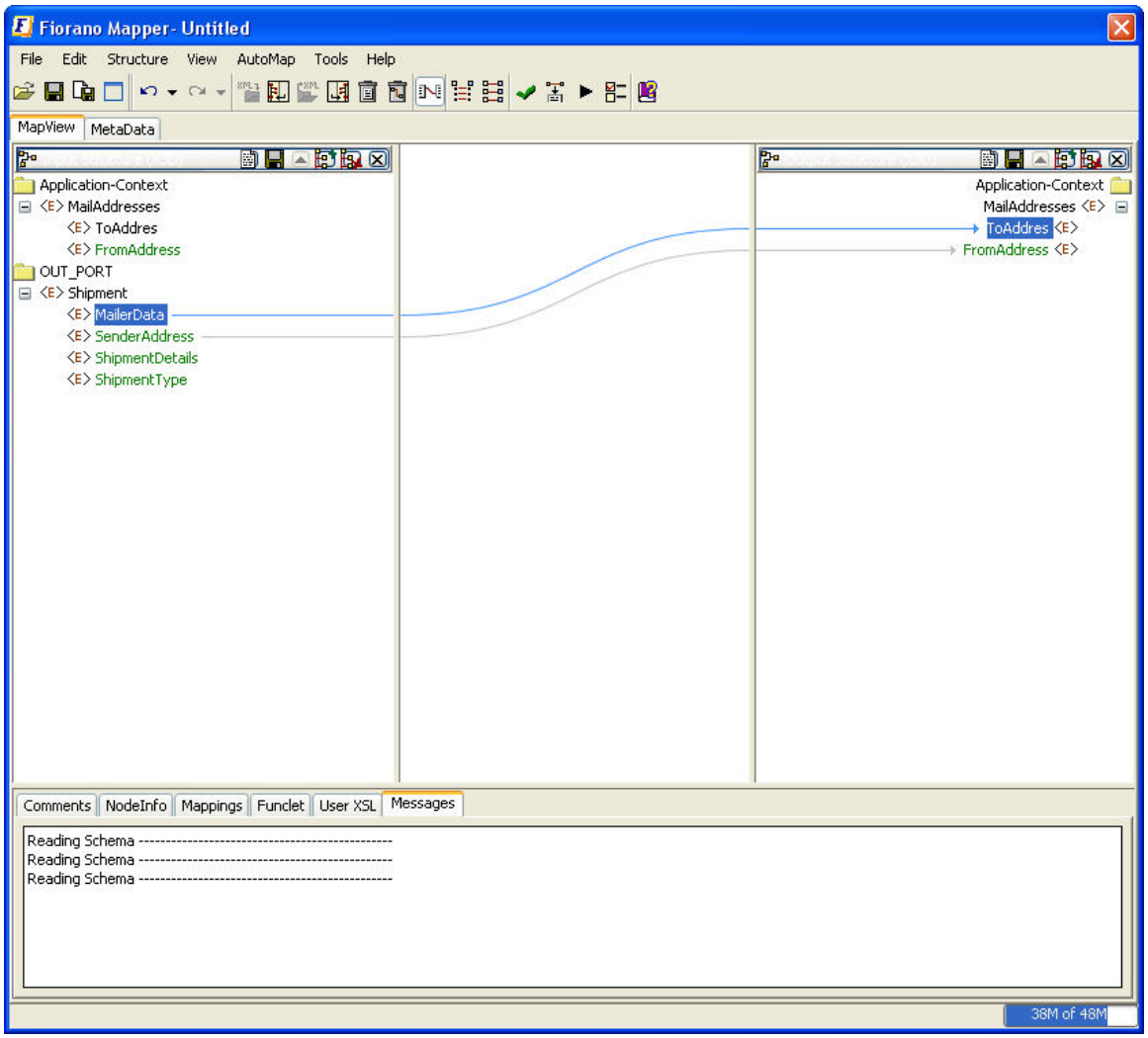


Figure 5.3.16: Event Process Configuration

6. The application context can be used any where in the event process via xslt component. The Figure 5.3.17 demonstrates the transformation that uses the application context values.

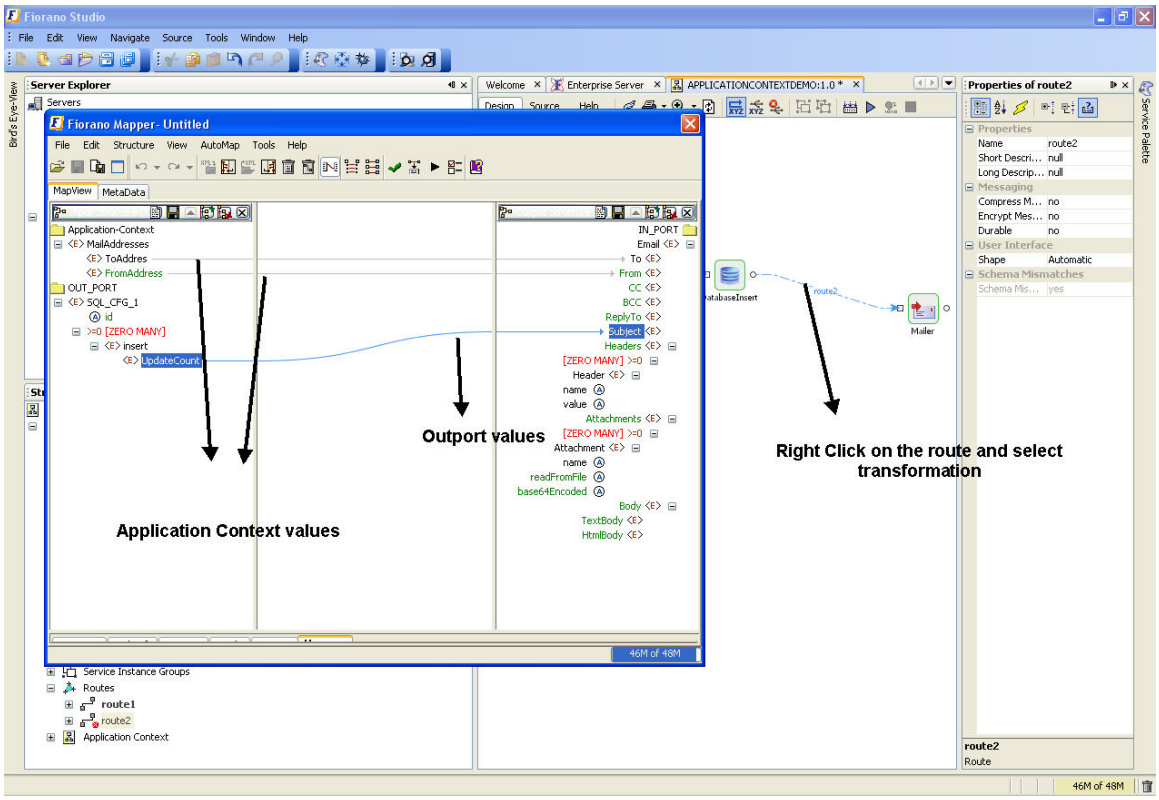


Figure 5.3.17: Demonstrates the Transformation

5.4 Using External Event Processes

5.4.1 Importing Remote Service Instance

User can import the service instances from other Application. The following image shows how to import a remote service instance.

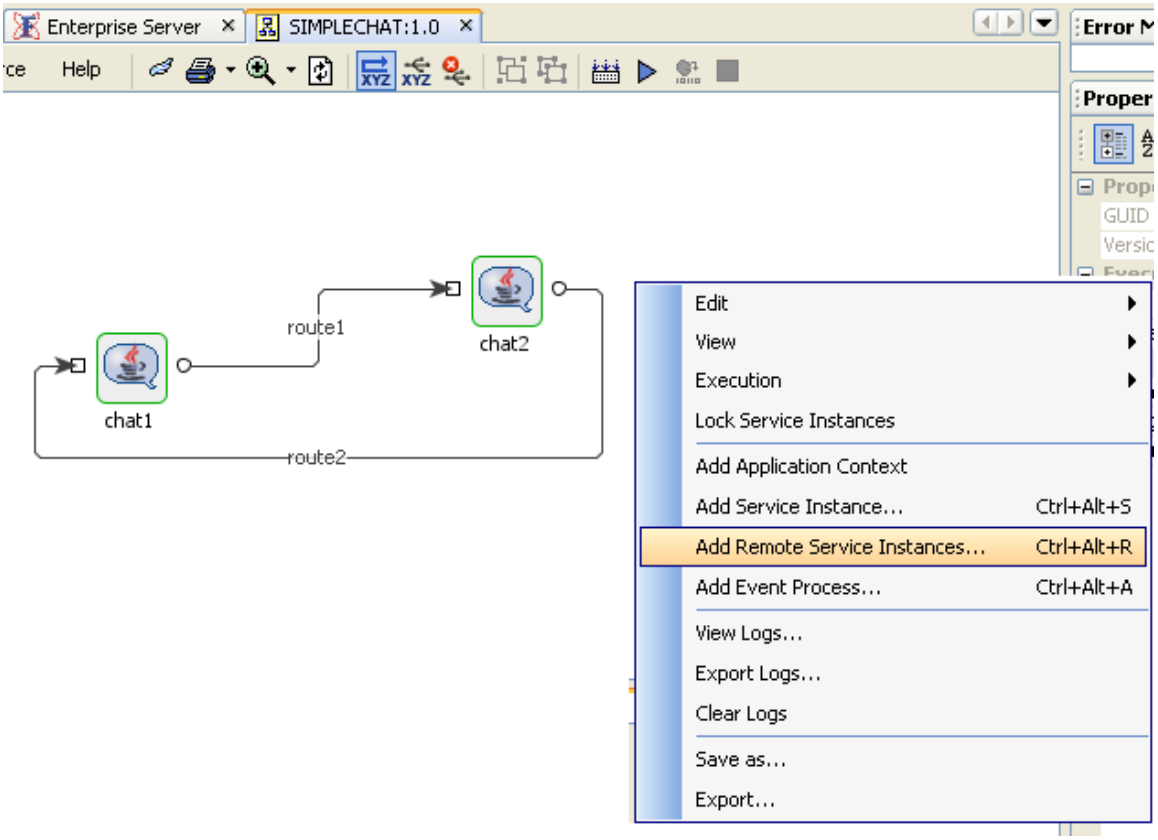


Figure 5.4.1: Adding Remote Service Instances

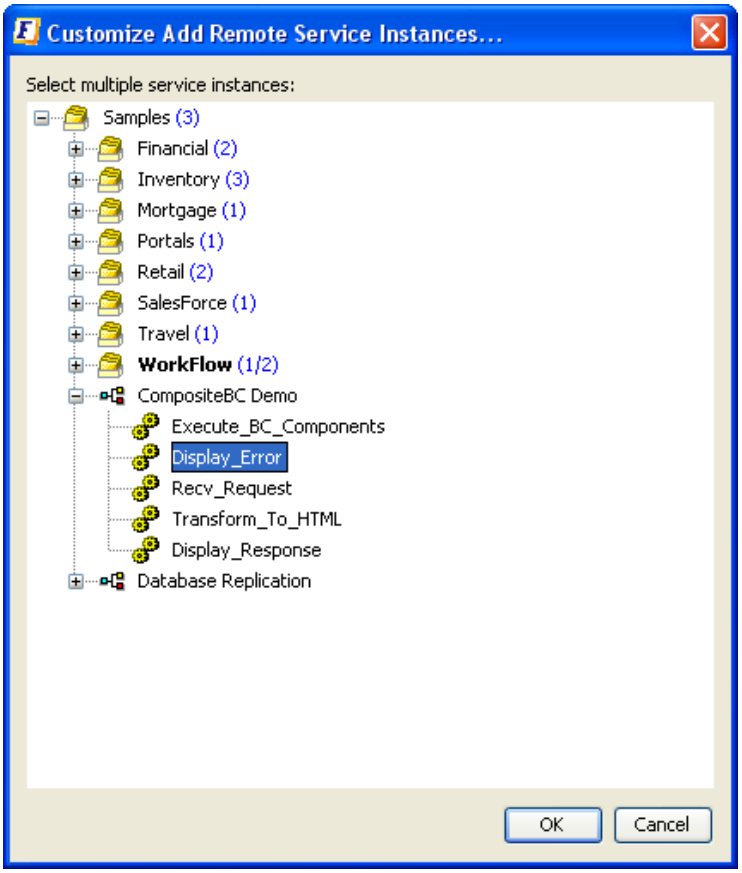


Figure 5.4.2: Selecting Service Instance

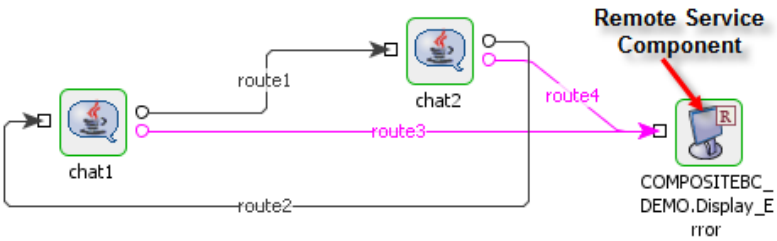


Figure 5.4.3: Using a Remote Service Component in an Event Process

The imported service instance is the reference to the service instance in parent application. Any changes made to the imported service instance in parent application are reflected in the current application. Current application is launchable when only application of remote service instance is running.

5.4.2 Using External Event Processes

The Fiorano Studio can import and orchestrate external event processes using its grouping capabilities.

Figure 5.4.5 and Figure 5.4.6 illustrates how you can import an external event process.

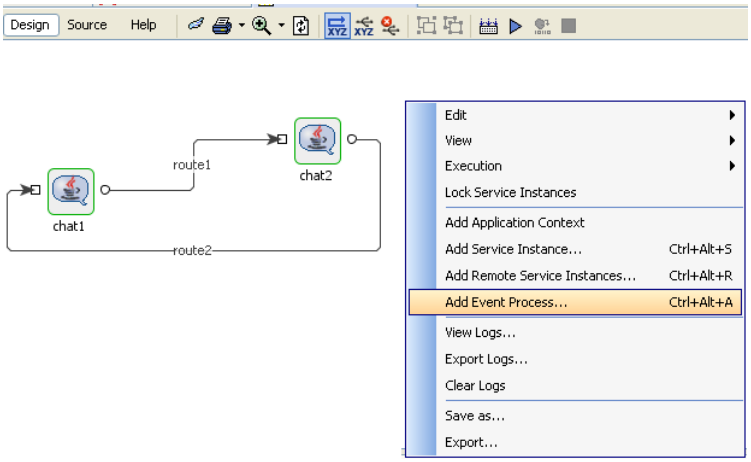


Figure 5.4.5: Adding Event Processes

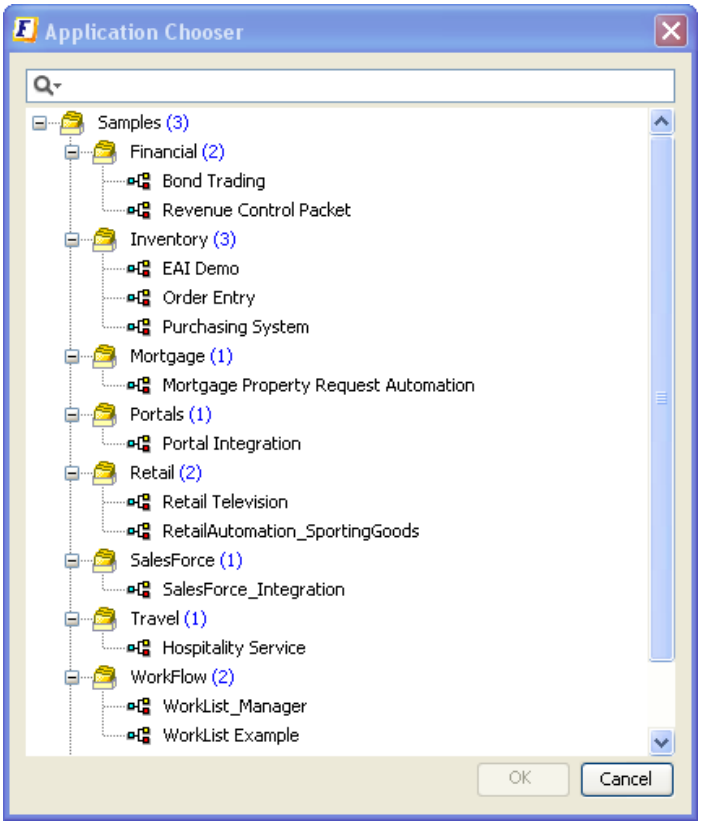


Figure 5.4.6: Importing an Event Process

Figure 5.4.7 illustrates how to use an imported event process in the Studio.

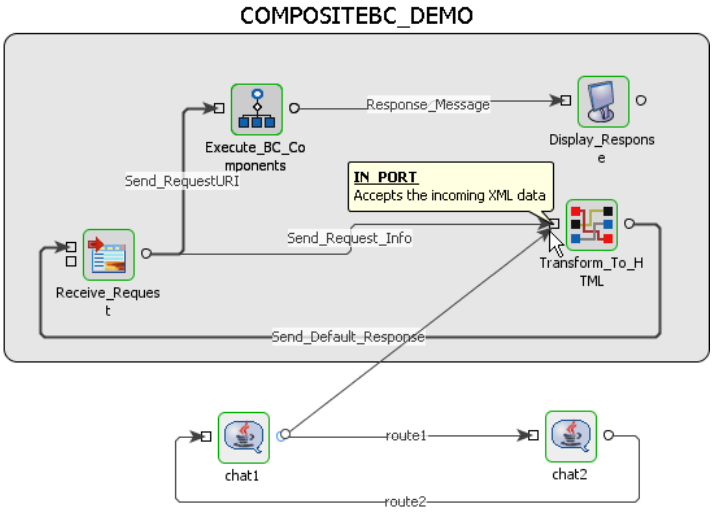


Figure 5.4.7: Using an Imported Event Process

5.5 Debugging Event Processes

Fiorano's unique Event Process orchestration model enables the debugging of live Event Processes in real time. The debugging model gives a view of the current state of executing business components within Event Processes and also provides a mechanism to setup *event interceptors* to capture, view, modify and discard messages flowing between business components on the same or different machines across the network.

5.5.1 Viewing Component Logs

Executing components typically write out debug logs on the peers to which they are connected. The current state of execution of a component can be captured from its logs. The component logs can be configured at different levels of detail by configuring the log module available in the properties window, as follows.

1. Select the component and view its properties panel. In the **Log Modules** property section, set the level for logging details, as illustrated in Figure 5.5.1.

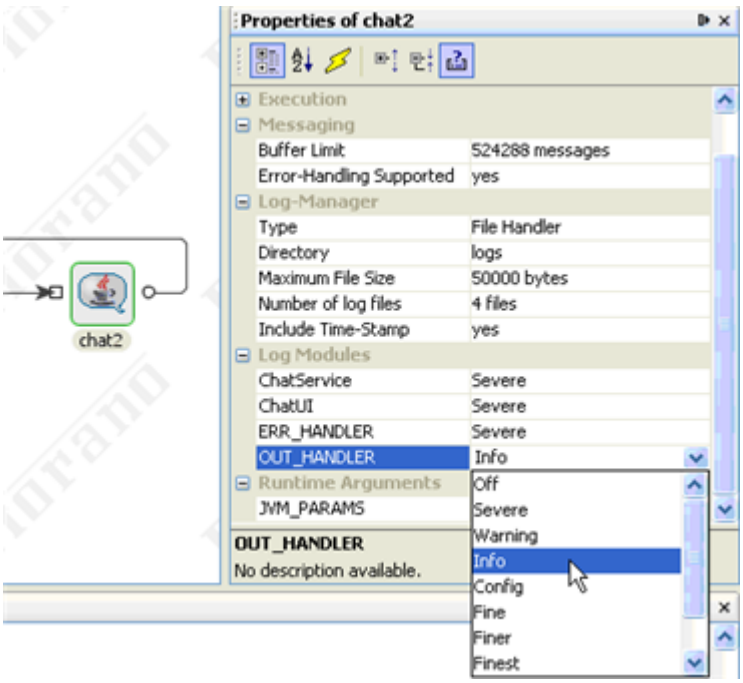


Figure 5.5.1: Configuring Log Module

2. To view component logs for a particular component at runtime either select the component and right-click to select **View Logs** as shown in Figure 5.5.2, or

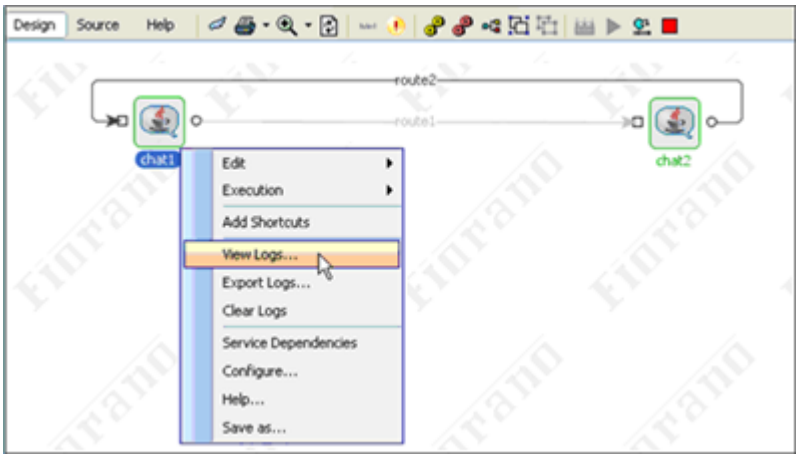


Figure 5.5.2: View Business Components Log

3. Click anywhere on the easel to select the Application and right-click to select **View Logs**, as shown in Figure 5.5.3.

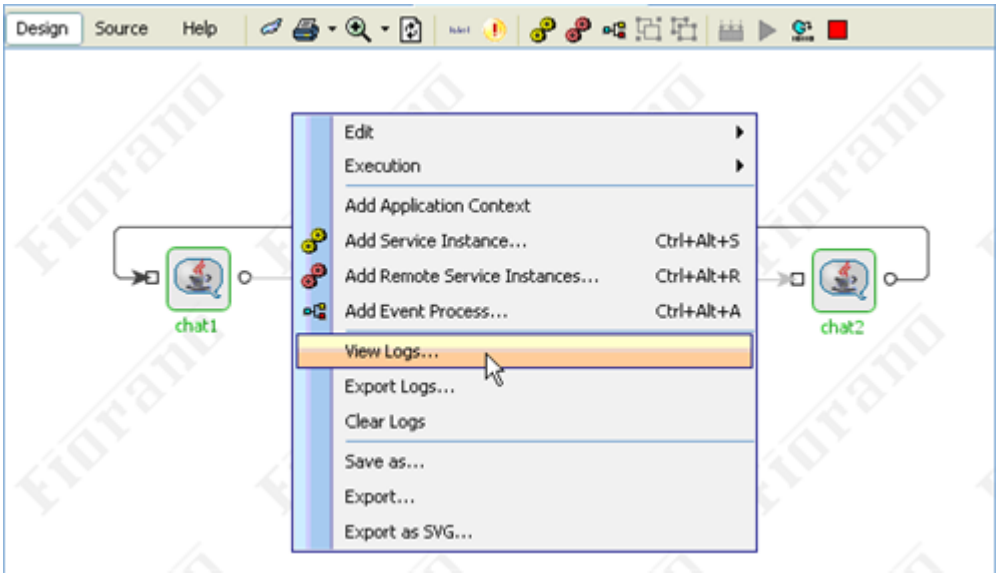


Figure 5.5.3: Viewing Business Component Logs from the Application Easel

In either case the log viewer pops up with the **Out logs** and **Error logs**, as shown in Figure 5.5.4. In the former case, the logs for the particular component under consideration are shown; while in the latter the view for the first component in the Application is shown. In both cases, the viewer can switch between different components within the application to view multiple component logs from a single dialog.

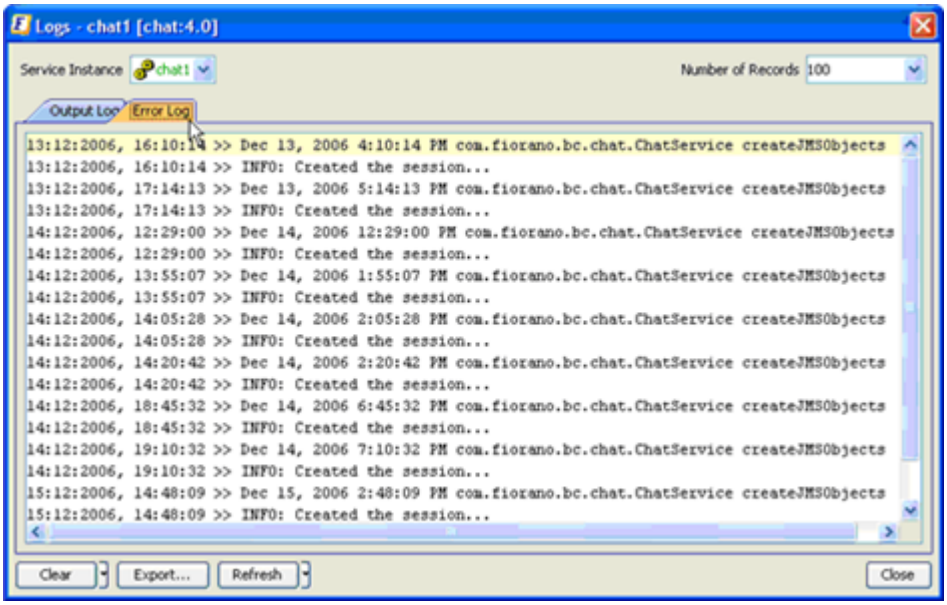


Figure 5.5.4: Error Log Screen

The out logs contain the execution steps for the component and the error logs contain the warning or errors thrown by the component while executing a request.

To choose between components within the application use the **Service Instance** drop box as shown in Figure 5.5.5.

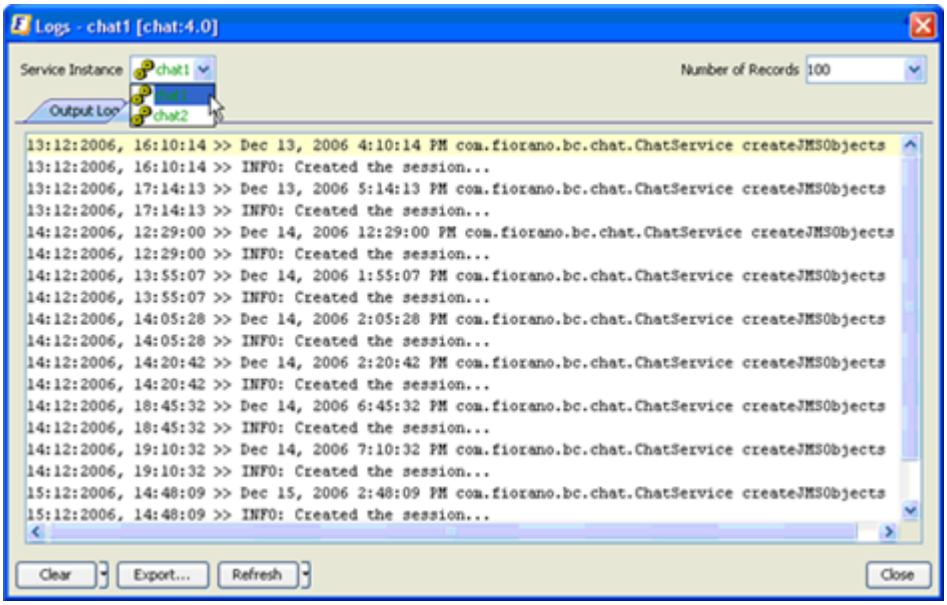


Figure 5.5.5: Select Service Instance

The **Number of Records** drop-down box on the top right hand side shows the logs for the last 'selected' records. A record in Fiorano parlance maps to a message executed by a component.

5.5.2 Setting Event Interceptors

Event Interceptors are break points that can be placed on routes in an Event Process. When an Event Interceptor is set on a route, it captures all messages flowing through the route. The messages thus intercepted can then be inspected, modified or discarded. Event interceptors are particularly powerful in that they allow data flowing between components on different machines to be inspected while completely shielding the user from the details of the underlying middleware.

An interceptor can be placed on a route at both at flow composition time and in a running Event Process.

5.5.2.1 Setting an Event Interceptor on a Route

1. Select the route and right-click on it. From the drop-down box, choose Interceptor → **Add Breakpoint**, as illustrated in Figure 5.5.6.

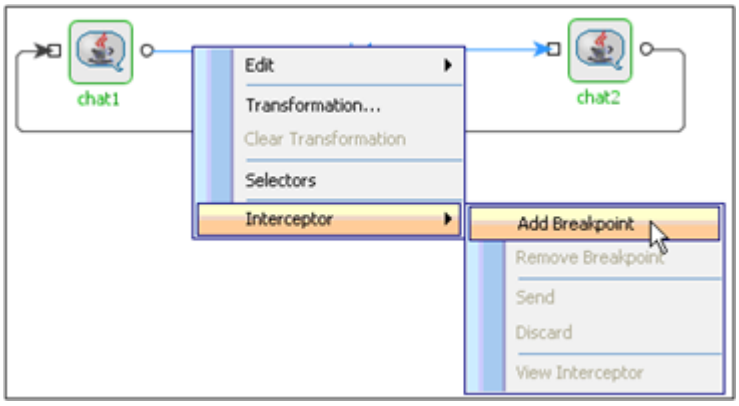


Figure 5.5.6: Add Breakpoint

2. Once the breakpoint is added the route changes its color to red indicating it is ready to intercept messages as shown in Figure 5.5.7.

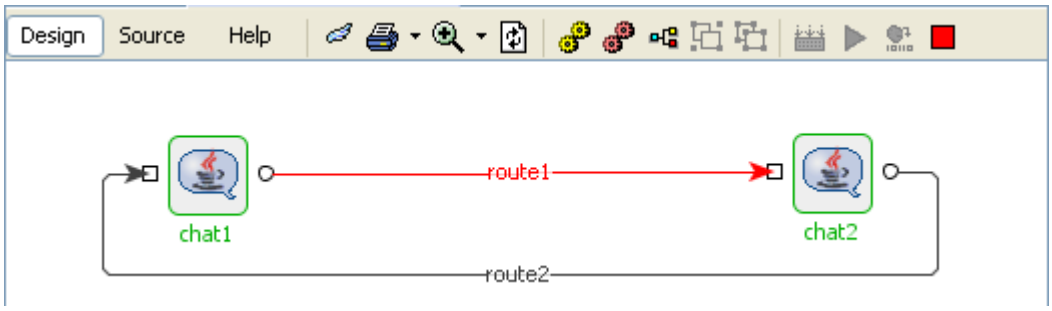


Figure 5.5.7: Route Color

3. When messages flow from the first component to the second, the route isgins to blink and a number appears on the route indicating the number of messages intercepted as shown in the Figure 5.5.8.

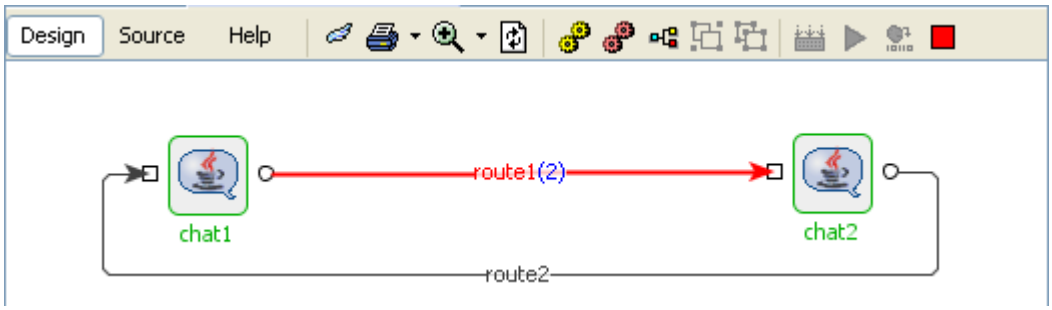


Figure 5.5.8: Route Color

5.5.2.2 Viewing Intercepted Messages

1. Double-click on the route to bring up the interceptor viewer, or
2. From the menu list, choose Interceptor → **View Interceptor**, as illustrated in Figure 5.5.9.

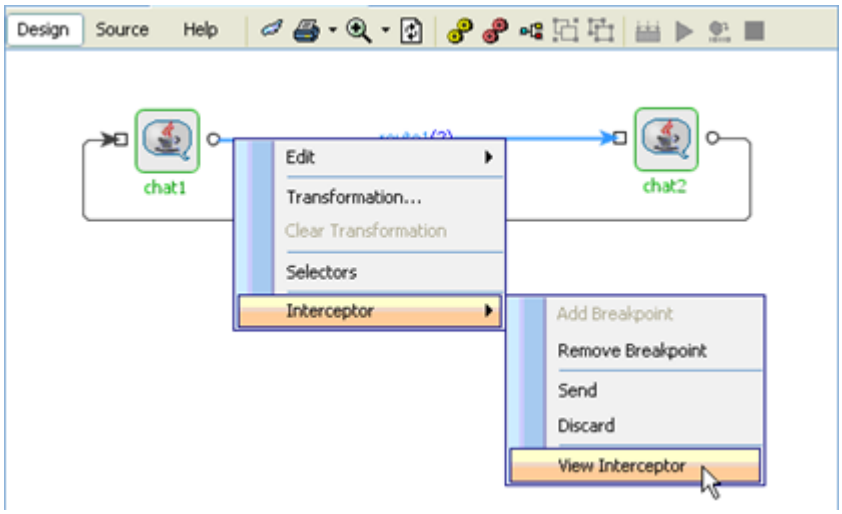


Figure 5.5.9: View Interceptor

This brings up the Message Interceptor View at the bottom of Studio as shown in Figure 5.5.9

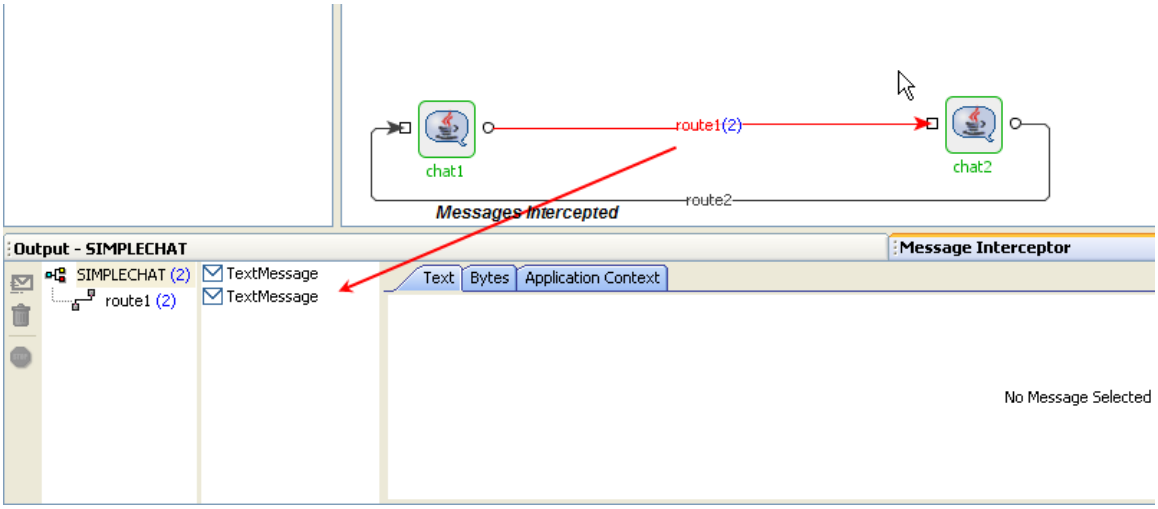


Figure 5.5.10: Message Interceptor View

The Event Process Name, the route on which the messages are intercepted and the messages can be viewed in the Message Interceptor View.

5.5.2.3 Viewing Content of an Intercepted Message

1. Select the appropriate route, and
2. Select the message and view its content as shown in Figure 5.5.11.

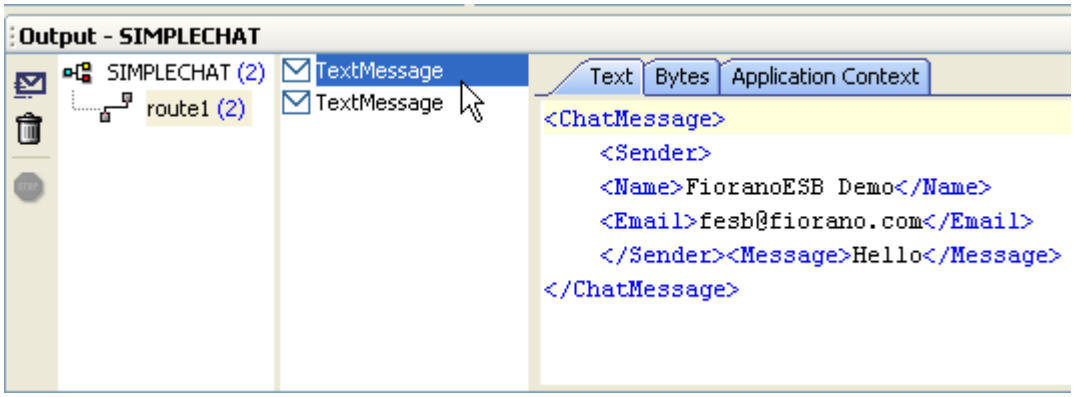


Figure 5.5.11: Message Interceptor View Screen

The intercepted message is fully editable and can have properties added/ removed and its contents modified; the message can be forwarded or discarded by right-clicking over the message icon and selecting the appropriate command from the drop-down list, as shown in Figure 5.5.12.

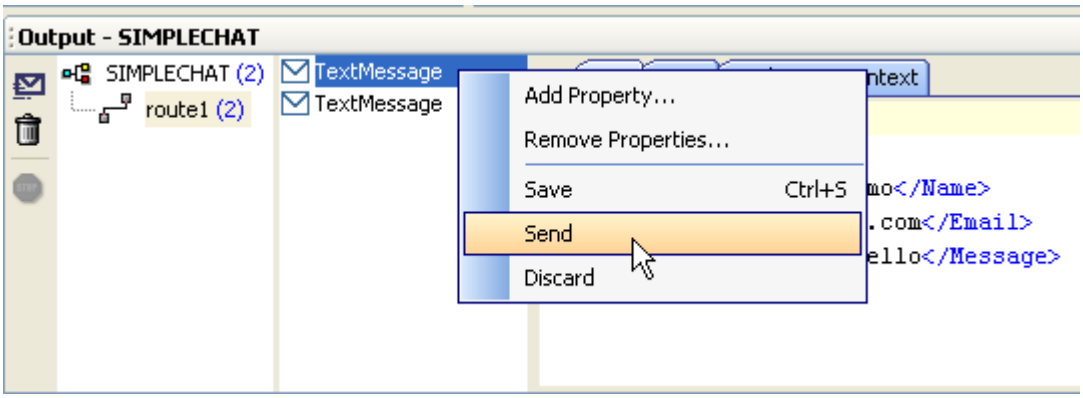


Figure 5.5.12: Sending Message from Message Interceptor View

Message Interception is a very powerful feature for debugging distributed event processes at run-time. It helps in enabling breakpoints across machines and geographical boundaries, giving the user a live representation for data flow inspection and modification.

5.5.2.4 Viewing Component Launch and Kill Time

The Fiorano Event Manager provides a view to monitor events occurring within the Fiorano Network. For instance, to view current status, version, and deployment node and launch time of a component in a running event process, log into the Fiorano Event Manager Tool, choose the appropriate event process from the list of event processes and select **Business Components**, as shown in Figure 5.5.13. Component information, including the Version, Status, Node Name, Launch Time and Kill time appears in the right-hand pane.

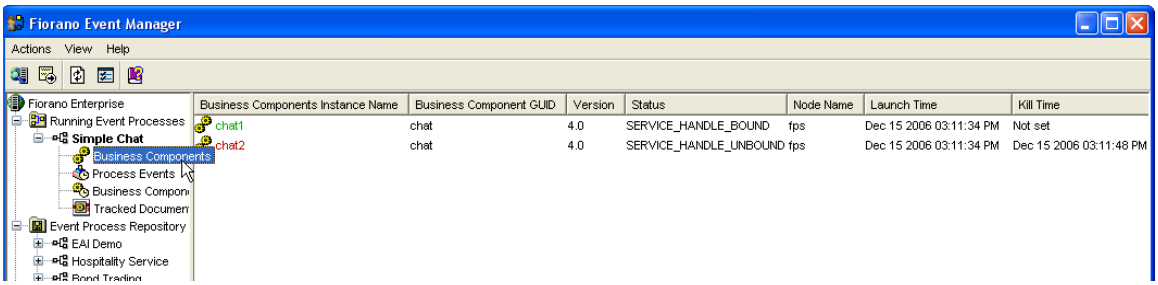


Figure 5.5.13: Fiorano Event Manager Business Component

5.5.2.5 Viewing Component Pending (Queued) Messages

To view pending messages to a component in studio, right-click on the input port of a component and select browse messages, which shows a list of messages pending on that queue.

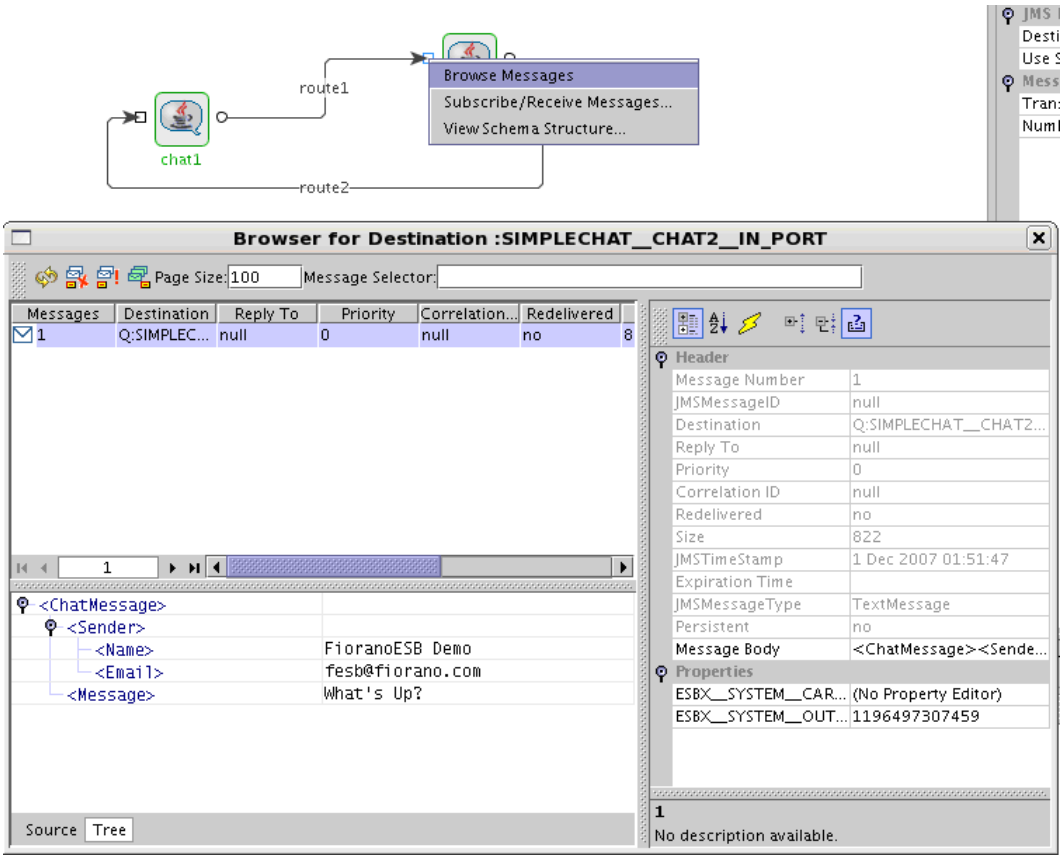


Figure 5.5.14: List of Messages Pending

5.6 Modifying Event Processes

Event processes can be modified at runtime, without stopping the event process, by dropping new components into the application easel and connecting routes to them.

5.6.1 Replacing a Component at Runtime

To dynamically replace an instance of a component within a flow with another component

1. Drag the second component from the palette and drop it over the component to be replaced, as illustrated in Figure 5.6.1.

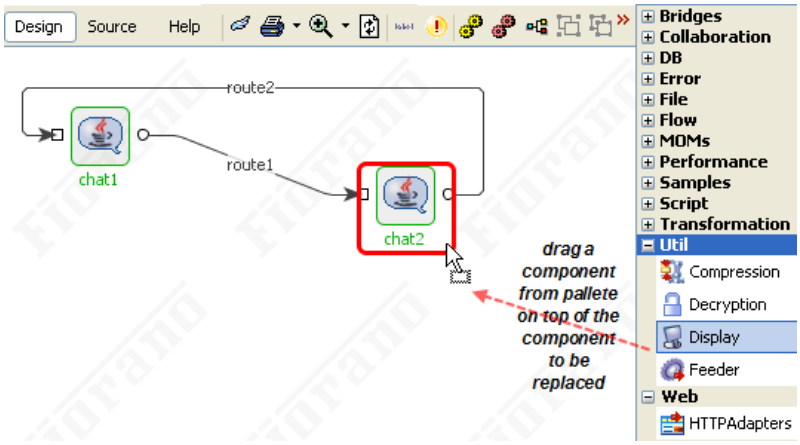


Figure 5.6.1: Modifying Event process

2. Right-click on the component, a pop-up menu appears with the choices **Add** and **Replace**, as illustrated in Figure 5.6.2.

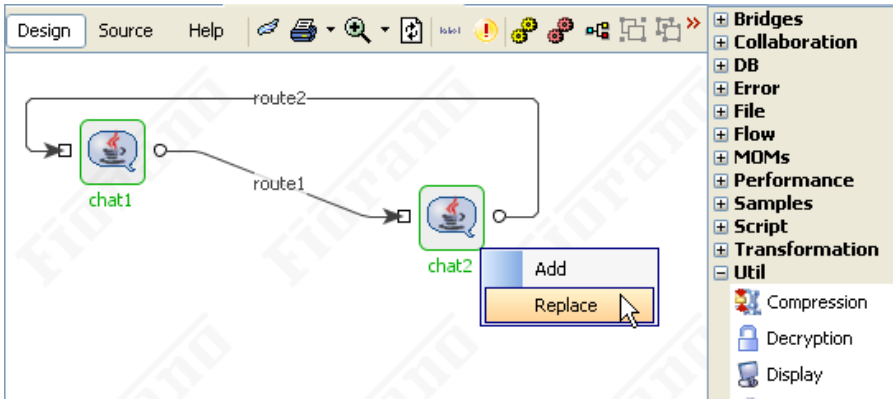


Figure 5.6.2: Add and Replace Menu

3. If you choose **Replace**, the component on the easel is replaced by the new component. Else an instance of the new component is added to the flow without replacing the first component.

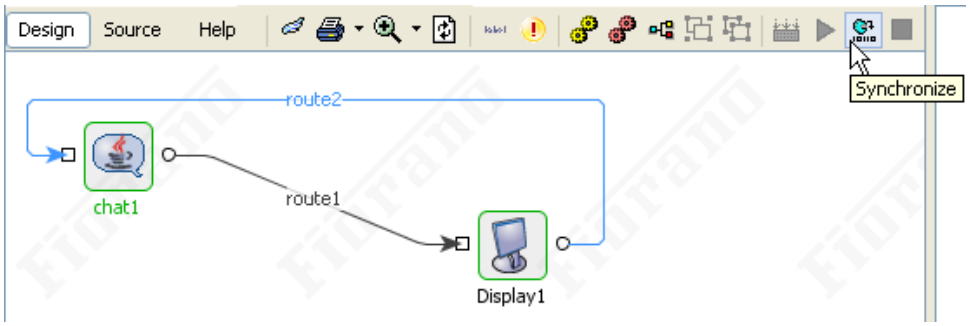


Figure 5.6.3: Synchronize Routes

Note: If the change is made in a running event process the Event Process will have to be synchronized by clicking the **Synchronize** button, as shown in Figure 5.6.3.

5.6.2 Adding a New Component Instance at Runtime

New components and routes can be added to a running event process in the manner described in the previous section. For instance, Figure 5.6.4 illustrates a new component instance, chat2, being added between the chat1 and display1 component instances. After the new instance is dropped on the easel, it is first configured and then routes are typically set to make the new instance part of the flow. The final step is to synchronize the application.

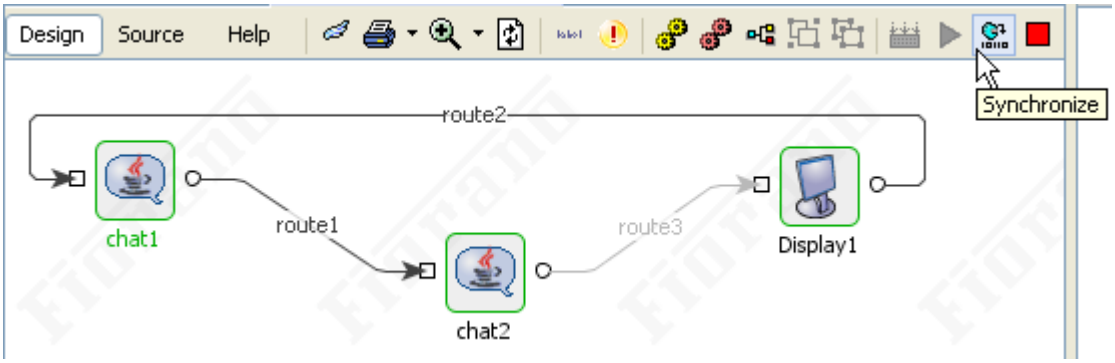


Figure 5.6.4: Synchronize Routes

5.7 Monitoring Event Processes

In a deployed Fiorano Network any event essentially becomes part of an event-flow, more generally referred to as a workflow. A workflow can span multiple Event Processes. Fiorano provides a one-click solution to define workflows and track events occurring within each flow.

5.7.1 Tracking Events within Processes

The Fiorano Event Manager provides a single view for events (also referred to as documents in the context of workflows) occurring in user-defined workflows in a deployed Fiorano Network and allows users to view component workflow logs.

The Document Tracking feature tracks events in a workflow by capturing workflow items and storing them in a database. The database used by default is the Derby Flat file database bundled with the platform, but any JDBC compliant database can be configured to capture Events as discussed below. The Document/Event tracking feature works on the ports of the business components in a given business process. Documents can be tracked both before and after they have been processed by a business component.

5.7.2 Defining a Workflow

A workflow in Fiorano terminology consists of an entry point, intermediary points and an exit point. The entry and intermediary points are defined as Workflow Items and the exit point is defined as a Workflow End.

5.7.2.1 Starting a Workflow

1. Select the entry point (that is, a port of a business component) from which documents need to be tracked. Mark this as a Workflow Item by setting the Workflow Property as illustrated in Figure 5.7.1.
2. Select intermediary points, ports of the other business components and mark them as Workflow Items as needed.

   All Workflow Items are colored green as shown in Figure 5.7.1.

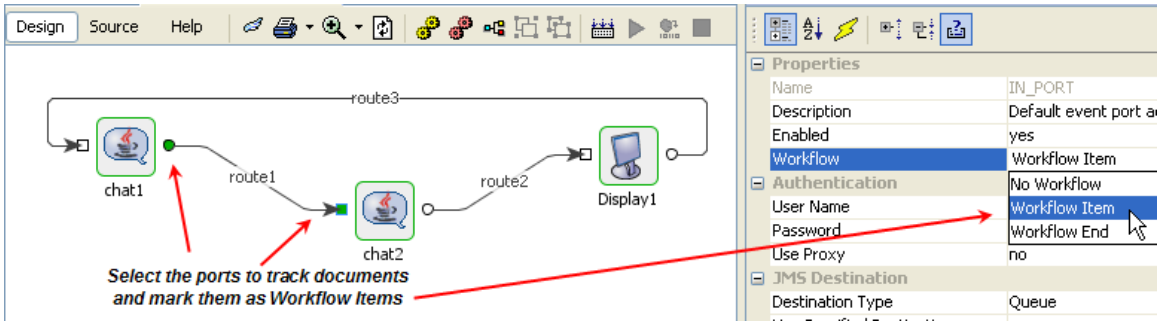


Figure 5.7.1: Workflow Items

3. To complete a workflow select the port of the last business component on which the documents are to be tracked and mark it as a Workflow End. The Workflow End is marked as red as shown in Figure 5.7.2.

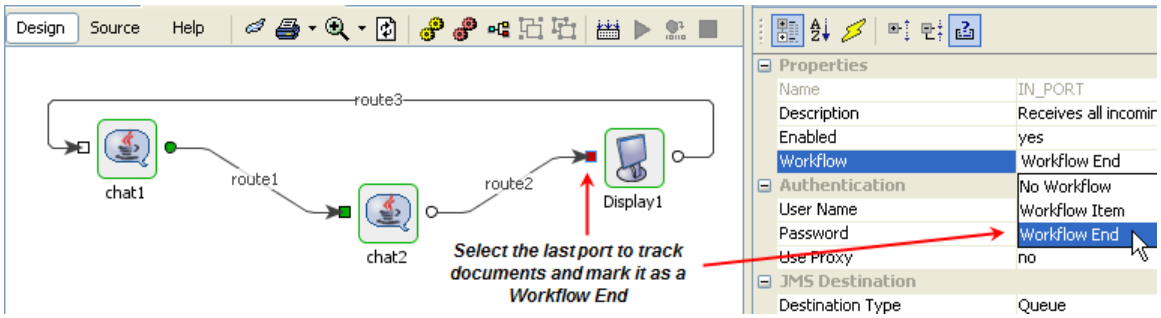


Figure 5.7.2: Workflow End

Note: The Workflow can be extended across multiple Event Processes. However, it is normally best to define process-specific workflows i.e. a message in the system will ideally pass through only one workflow.

5.7.2.2 Viewing Tracked Documents of a Workflow

To view the tracked documents, log into the Fiorano Event manager and select the running event process on which the tracked documents are to be viewed, as illustrated in Figure 5.7.3. All tracked documents for the Event Process are displayed on the right-hand pane of the Event Manager window.

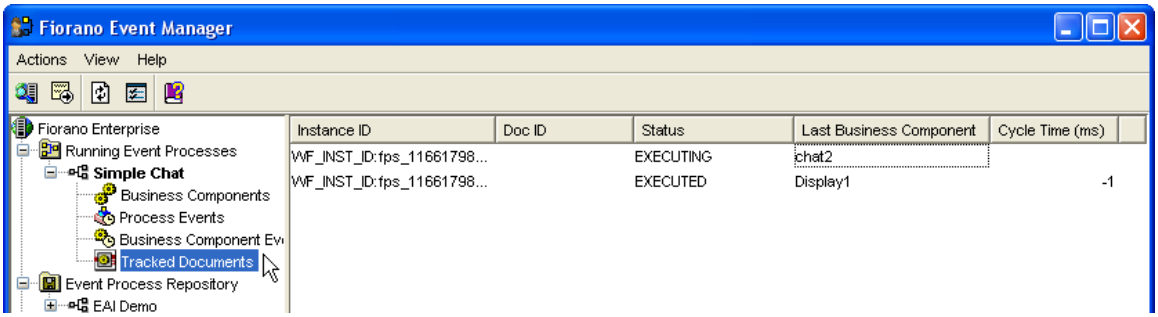


Figure 5.7.3: Fiorano Event Manager Tracked Document

5.7.2.3 Tracking Documents across Workflows

Sometimes it may be possible that Workflows intersect. To indentify the messages for a particular event-process/workflow, a unique property "ESBX__SYSTEM__USER_DEFINED_DOC_ID" can be set as a JMS property at the start point of the workflow to mark a message as belonging to that particular workflow. This JMS property is persisted for the message as it traverses through the workflow.

To define parameter for ESBX__SYSTEM__USER_DEFINED_DOC_ID, mention the parameter "value" which expects string values. In case you are using a string value more than 800 characters, the appropriate jdbc config file needs to be modified. For example, in case of MSSQL database, the file "FioranoSOA_Home\esb\server\profiles\<profile-name>\FES\conf\mssql_jdbc.cfg needs to be modified (DB_COLUMN_SIZE property).

5.7.2.3.1 Adding the JMS Property to a message

JMS property can be added to an existing message using the Mapper tool. In the funclets choose JMS Message functions, using the `setString` property function, set the name and value for the property and simply map it onto any element of the message structure on the right hand side. This property will then be set.

5.7.3 Setting up Database to store Tracked Documents

Fiorano uses an inbuilt H2 Flat File Database to track documents. However, the Document Tracking feature can be plugged into any external JDBC-compliant database.

The configuration files for Document Tracking is available under the conf directory of the profile of the Enterprise server for which this setting is to be enabled i.e.

%FIORANO_HOME%/esb/server/profiles/<Profile_Name>/FES/conf. For more information on how to use external database for document tracking, refer to section 5.3.8.2.

5.7.4 Re-Injecting Tracked Documents

A document tracked at a particular port can be re-injected into that port using ESB dashboard. At all pages (Application Documents Page, Document Search Page, Latest Events Page) where dashboard shows a tracked document, it provides options to re-inject that document to the port on which it was originally tracked. Users are provided the facility to edit the tracked document before re-injection.

5.7.4.1 Re-Injection Document Structure

A document to be re-injected contains the following information. Each of the below sections of the information can be edited by the user before re-injection.

1. Message Properties

This section is filled with carry forward properties of the tracked message by default. Users can choose to add/remove specific properties from this property set. At the time of re-injection, 2 additional properties are added to the list of carry forward properties of the tracked document. These properties are as explained below:

A. ESBX__SYSTEM__REINJECTED

This value of this property is set to 'true'. On a tracked document, this property can be used to identify whether the document under observation is a re-injected document or a normal document.

B. ESBX__SYSTEM__REINJECTION_POINT

The value of this property specifies the application name, service instance name and port name where the re-injection is being done. On a tracked document, if the above property is set to 'true', then this property can be used to identify the point at which re-injection was done.

2 Message Body

This section is filled with text contents of the tracked message by default. Users can edit the text contents before re-injection. However, while editing the message, user has to ensure that the new text message conforms to the schema specified on the port to which document is being re-injected.

3 Application Context

This section is filled with application context of the tracked message by default. Users can edit the application context before re-injection. However, while editing the application context, user has to ensure that the new value conforms to the schema specified for the application context.

4 Source Contexts

Source Contexts are the list of output ports (along with application and service instance name) through which the message has traversed till now. This section of the re-injection message is filled with source contexts of the tracked message by default. Users are not allowed to add more source contexts to the message being re-injected. However, users can delete all the source contexts from the message before re-injection so that the re-injected message looks like a fresh message.

5.7.4.2 Re-Injection of Tracked Documents

Each search result showing a tracked document in dashboard has a column named 'Re-Inject' on pressing which a menu pops up. The menu shows 2 options.

1 Re-Inject This Document

When user chooses this option, the document corresponding to the chosen search result will be made ready for re-injection.

2 Re-Inject Other Document From Same Workflow

When user chooses this option, he will be shown all the documents that belong to the workflow to which the chosen document belongs. User can then choose one of these documents and the selected document is made ready for re-injection. Note that re-injection is always done to the port on which the document was originally tracked.

After choosing the option above, the re-injection consists of 2 steps.

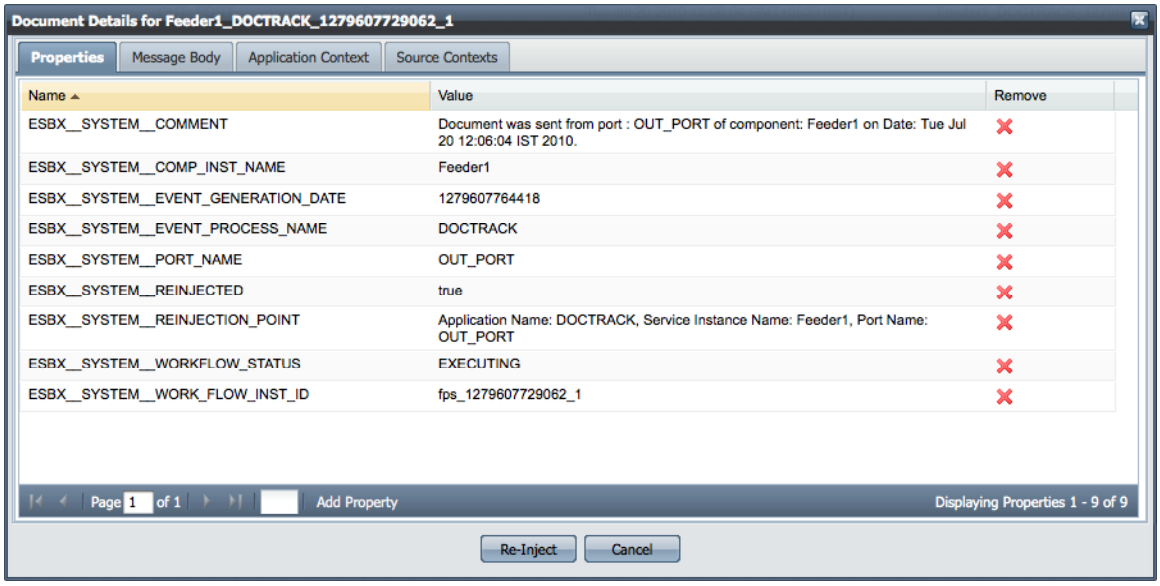
1. Message Composition
2. Message Re-Injection (Publishing the composed message)

An explanation for the actions performed in these 2 steps is provided in the following sections.

5.7.4.2.1 Message Composition

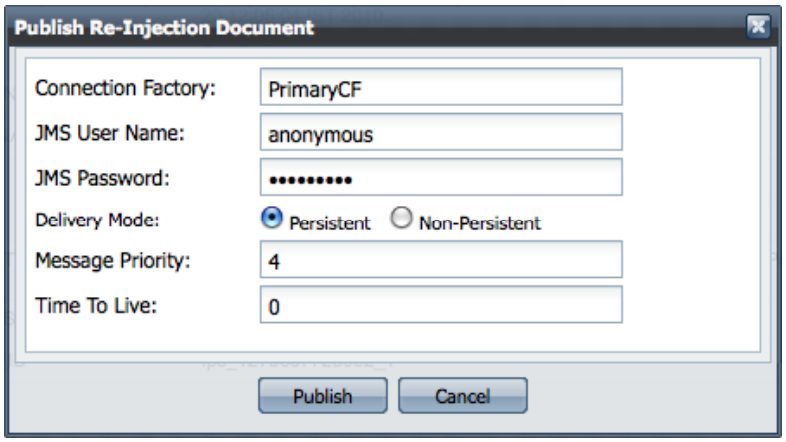
In this step, user can compose the message to be re-injected as per the Re-Injection document structure specified in section 5.7.4.1. After this step, user can press the Re-Inject button at the bottom of the Message Composition Window, which takes the user to next step.

Refer to screenshots below (taken from dashboard) showing a document being re-injected. The screenshot shows the Message Composition Window with 4 sections of Re-Injection Document Structure in different tabs. The Message Properties tab is in active state.



5.7.4.2.2 Message Re-Injection

In this step, user has to specify the following properties for the message being re-injected (See the screenshot below). After specifying these properties, the message can be re-injected using the Publish button as shown in the image. An explanation for the parameters shown is provided below.



1. JMS User Name
2. JMS Password

These 2 parameters will be used to create JMS connection to the Peer Server to which the message is being re-injected.

3. Connection Factory Name

   This parameter specifies the connection factory to be used to create JMS connection to the Peer Server.

4. Delivery Mode

   This parameter specifies whether the message being published is a Persistent or a Non-Persistent message.

5. Message Priority

   This parameter specifies the priority of the message being published. A higher priority message is processed before other lower priority messages residing in the destination's queue.

6. Time To Live

   This property specifies the maximum time interval for which the message will be stored in the destination before it gets picked up by the subscriber/receiver.

After filling these details, the message can be finally re-injected.

5.7.4.3 Re-Injected Workflow

When a document is re-injected, it may give rise to another workflow (set of documents) being tracked, as per the document tracking points set in the Event Process. By default, the newly generated workflow is merged with the old workflow. Thus, the new documents tracked because of re-injection become part of the old workflow. If users wish, they may choose to create a new workflow from the re-injected message. For this purpose, the user should perform the following steps while re-injection.

1. Remove the property named 'ESBX__SYSTEM__WORK_FLOW_INST_ID' from the properties panel while composing the Re-Injection message.
2. Remove source contexts from the Re-Injection Message by navigating to Source Contexts tab in the Message Composition Window. Although this step is not mandatory, it is recommended to do so to maintain consistency in tracked document data.

5.8 Import and Export Event Processes

Fiorano event processes are saved in the repository as XML files. An Event process can be exported out of the Fiorano System (typically to the desktop) and/or imported. The Fiorano Studio enables single-step import and export as discussed below.

5.8.1 Importing Event Processes

1. Select one or more event process in the **Server Explorer** window and right-click to select the Import Event Process tab from the pop-up menu, as shown in Figure 5.8.1.

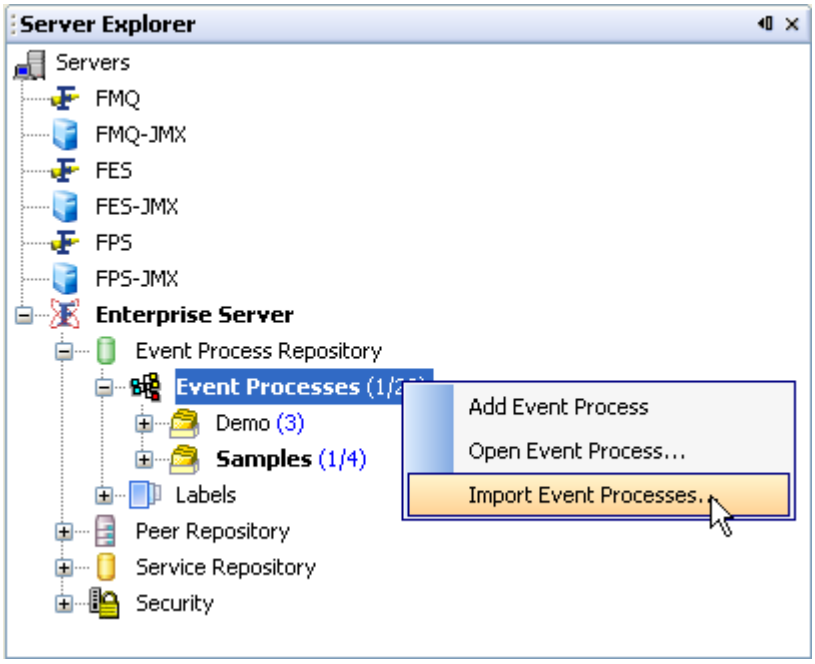


Figure 5.8.1: Import Event Processes

2. The **Import Event Processes** dialog box appears to select one or more Event Processes. Select the appropriate Event Process(s) and click **Open**, as shown in Figure 5.8.2.

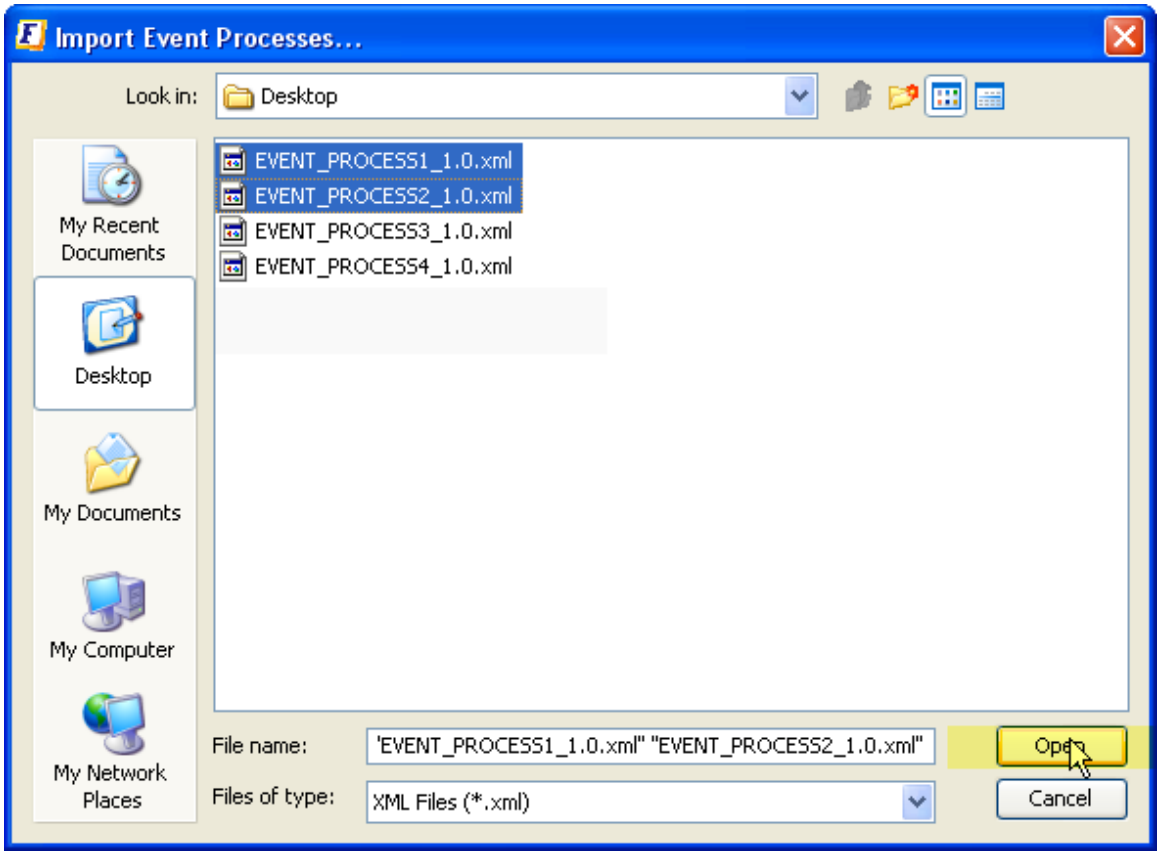


Figure 5.8.2: Browse Event Process to Import

3. The names and categories of the event process(s) can now be customized as shown in figure 5.8.3 and the processes saved into the repository, thus completing the Import process.

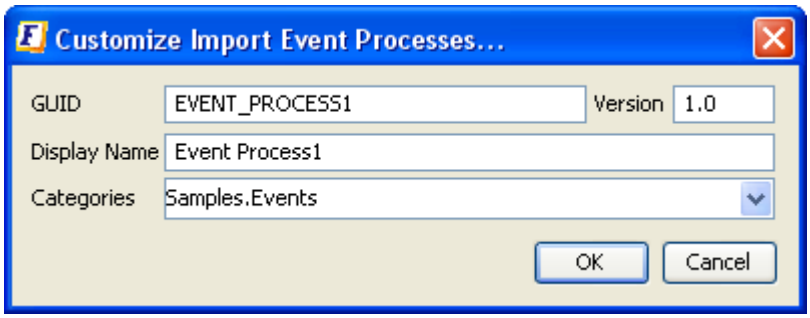


Figure 5.8.3: Customize Event Process

5.8.2 Exporting an Event Process

If the Event Process is already opened for editing, right-click on the Event Process and select **Export** from the pop-up menu, as shown in Figure 5.8.4. Specify the target location (typically a directory on the network) to export the Event Process. The Event Process is exported.

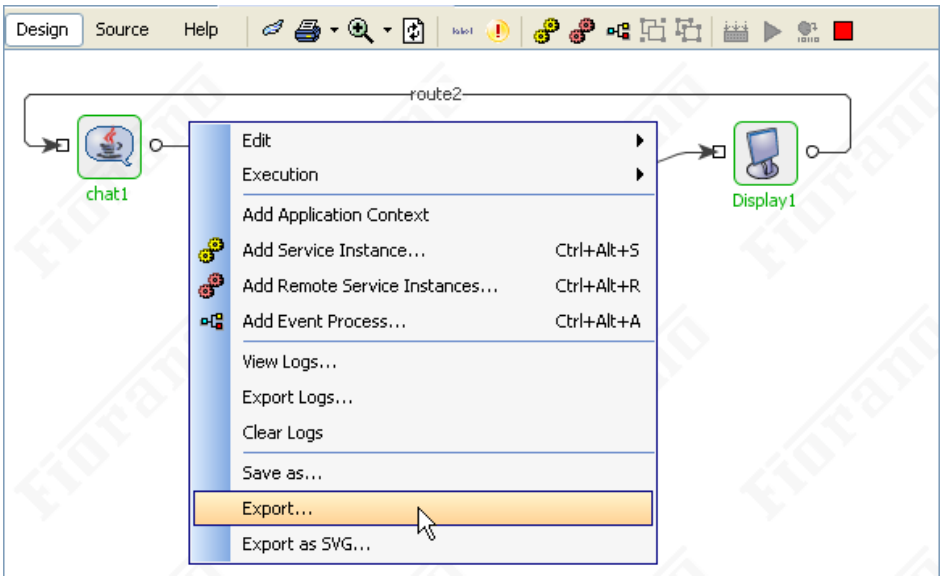


Figure 5.8.4: Export Event Process

The application.xml for the event process can be saved.

5.8.3 Exporting Multiple Applications

Select one or more Event Process from the **Server Explorer** window and right-click to select the Export from the pop-up menu. Specify the target location to export the Event process(s). The Event Process(s) is exported, as illustrated in Figure 5.8.5.

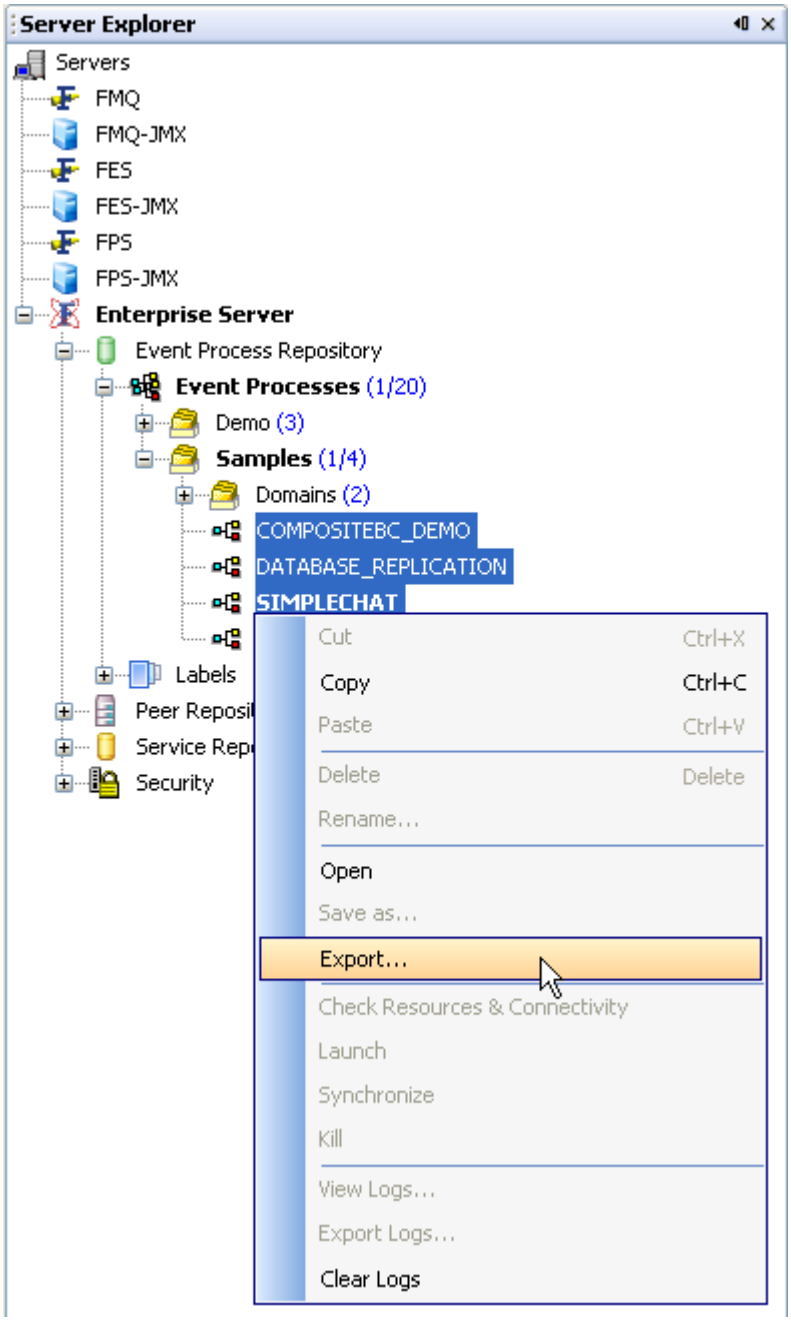


Figure 5.8.5: Export Multiple Applications

This process of import and export can also be done through the Event Process Command Line Interface as described in section 5.11 The Event Process Command Line Interface.

5.9 Deploying Event Processes

The typical life cycle of an event process in includes the creation of flows using components, component configuration, linking components with routes, the creationof transformations for mapping inputs from one linked component to another if required, checking the resources of the application and, finally, launching the event process.

Since an Event Process is simply a collection of Service Components linked via routes, it can be deployed at any point during the composition process. A set of components configured and connected via routes in the Studio easel is considered ready for deployment.

The toolbar at the top of the Studio orchestration easel provides tools to operate on developing event processes.

Figure 5.9.1: Toolbar menu

The following table explains the buttons on the toolbar:

Button	Description
Design	Open event process for editing
Source	View application.xml for an event process
Help	Enter notes for an event process
	Bird's eye view for an event process
	Print an event process
	Zoom in/out
	Refresh event process
	View the port names of the component
	View the error ports of the component
	Select a service instance in the current event process
	Select a service instance from another event process to current event process
	Select another event process from the current list of event processes
	Select multiple components to group them
	Ungroup a selected a group of components

Button	Description
	Perform the connectivity and resource check for an event process
	Launch the event process
	Synchronize the event process
	Stop a running event process or component

5.9.1 Connectivity and Resource Check

Fiorano enables the deployment of an event processes over a distributed peer to peer grid of infrastructure servers (known as "peer servers") at the click of a button. A developed event process contains a set of configured components connected via routes. The configuration for these components also includes the names and/or IP addresses of the grid-nodes (Fiorano Peers) on which the components are to be deployed.

Figure 5.9.2 illustrates the initiation of the Connectivity and Resource Check on an Event Process:

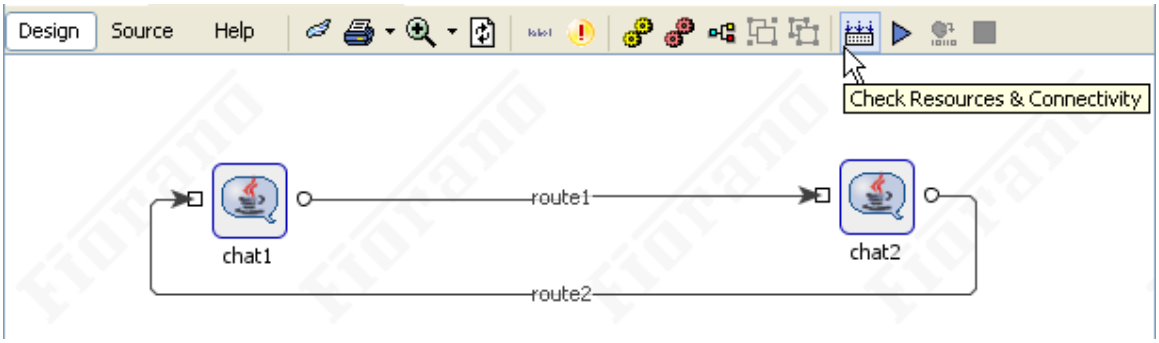


Figure 5.9.2: Check Resources and Connectivity

The Connectivity and Resource Check process involves the following:

- For each Component instance in the flow, checking if at least one of the peers in the deployment-node-list of the component instance is is live and available within the Fiorano Network.
- Dynamically deploying the components and the external dependencies for the components used in the flow into the local repository of the peer on which the components runs.
- Checking to ensure that any two components with different schemas have an appropriate transformation defined if they are connected via a route.
- After the first three steps are successfully completed, Registration of the Event Process as "launchable" from the Fiorano Enterprise Server.

Figure 5.9.3 illustrates availability of peers on the network: If the peer on which a component instance is to run is available, the component instance has a green border.

 Peer configured but unavailable on network	 Peer configured and available on the network (green border)

Figure 5.9.3: Peer availability

Thus at the click of a button and from a single point of control, services can be deployed on different peers across the enterprise network.

5.9.2 Enabling/Disabling the Component Cache

During the process of development, some components might have external resources added. Also, for custom built components the source files might be updated from time to time.

To reflect the changes for such components across the peers at runtime, the Fiorano Studio has in its properties panel a "Component-Cache Disabled" deployment configuration at both the Application (Event Process) as well Component levels, that optionally forces the resources of the component to be re-fetched each time the component is launched, as illustrated in Figure 5.9.4.

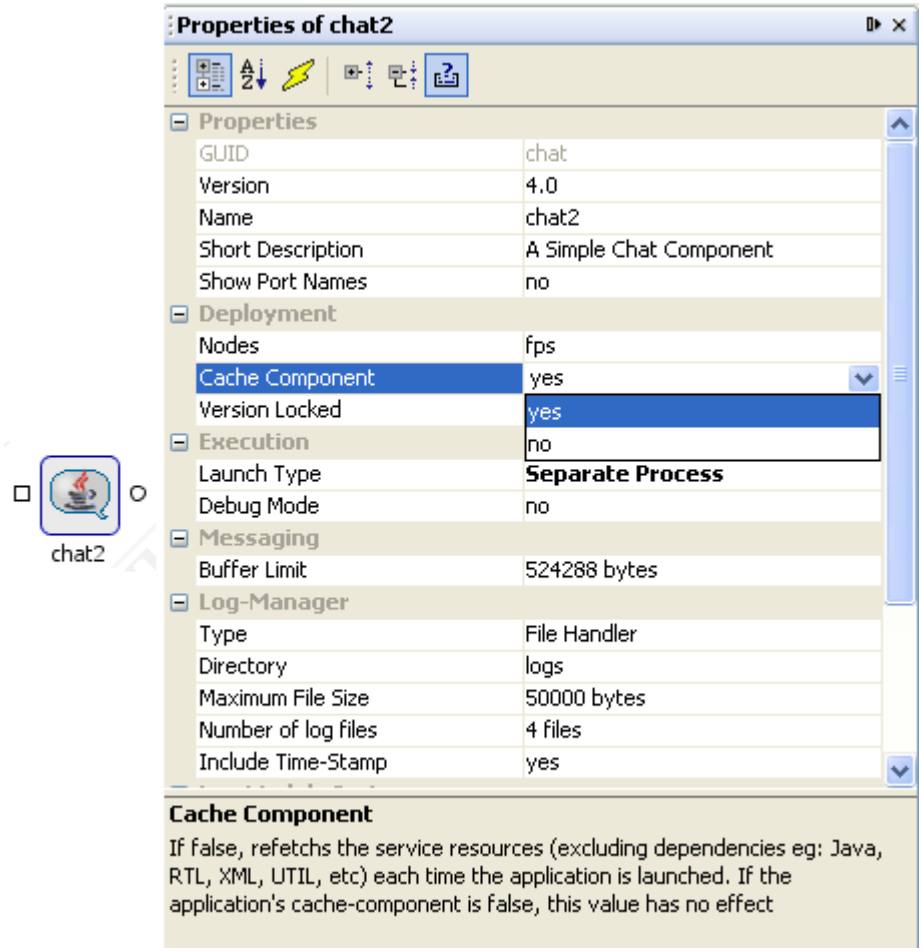


Figure 5.9.4: Fiorano Studio Properties Panel

The properties panel is activated when a component or an Event Process is selected. An Event Process can be selected by clicking anywhere on the orchestration easel.

The available settings for the Component-Cache Disabled parameter are:

No — This setting forces the resources of the components to be fetched from the enterprise server each time the application is launched even if they are present in the local repository. This is the default setting.

Yes — Picks up a component's resources from the peer's local repository without refetching the resources from the enterprise-server repository, even if the resources might have changed in the latter.

Note: If the Application level setting for "Component-Cache Disabled" is set to **Yes** then this setting applies to all components in the application. If the application-level setting is **No** then the Component level settings is considered separately for each component.

5.10 Launching Components and Event Processes from Studio

Under normal circumstances, a component is only be launched as part of an application. A component within an Event Process can be configured to be automatically launched in two ways:

- In an independent JVM on any machine across the network (including the machine running the peer server to which the Component is connected); this is the default option, or
- Within the same JVM as the peer server to which it is connected. Such a launch is called an "In-Memory" launch, since the component runs "in the memory" of the Peer-server JVM.

In addition to the two automatic-launch options above, it is possible to configure components for manual launch; that is, the launch is initiated from an external source, either manual or programmatic. Figure 5.10.1 illustrates how to configure the Launch Type of a component in the properties panel.

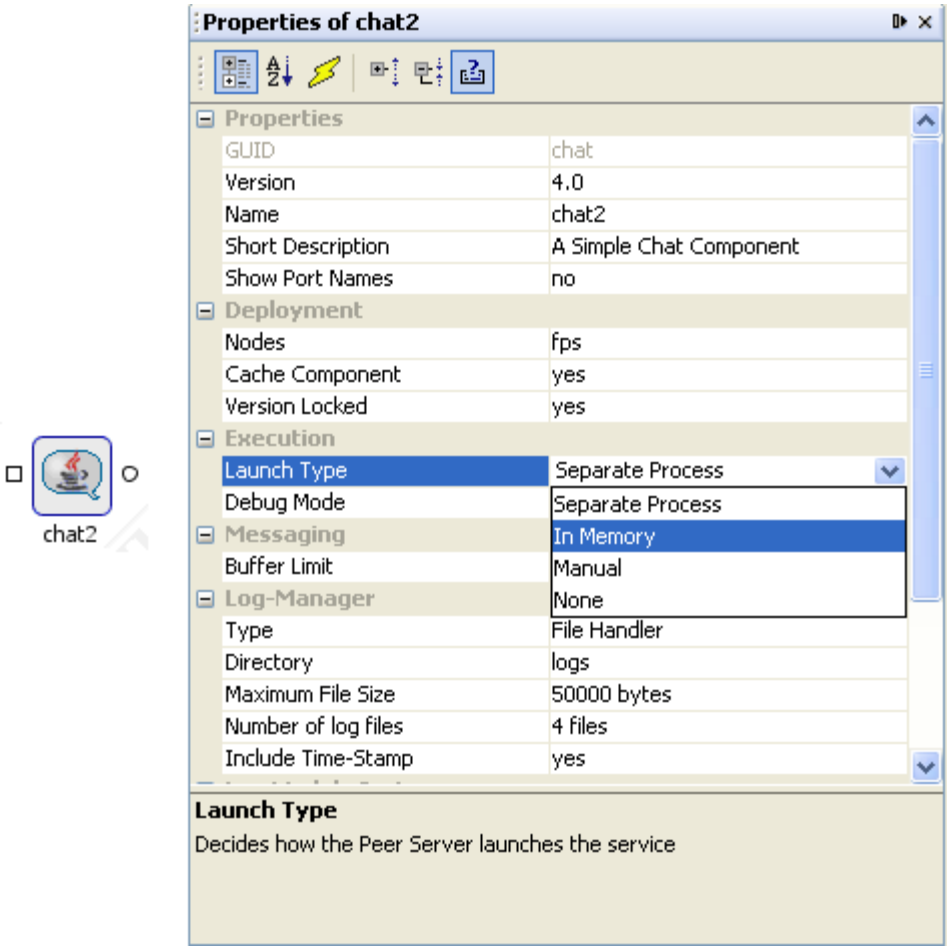


Figure 5.10.1: Fiorano Studio Properties Panel

Figure 5.10.2 illustrates how components launched differently are visually distinguished from one another:

Separate Process	In Memory	Manual Launch

Figure 5.10.2: Launching Component in Various Method

5.10.1 Launching an Event Process

1. Click anywhere on the easel to select the **Event Process**.
2. Click the **Run** button on the tool bar, as shown in Figure 5.10.3.

   Note: If a single component is selected in place of the entire Event Process, that single Component is started when the Launch icon is clicked.

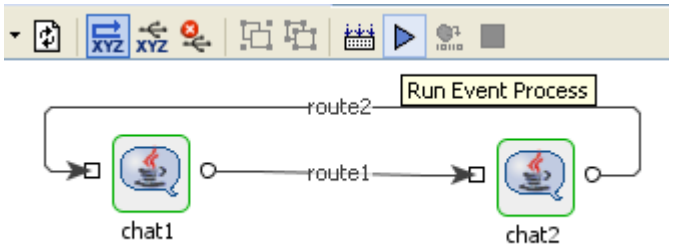


Figure 5.10.3: Launching an Event Process

When an Event Process is launched, the names of the components present in the orchestration easel turn green to indicate that they are now executing.

Stopping an Event Process

When an Event Process is up and running it can be stopped as a whole or individual component within it can be stopped and re-started as needed.

5.10.2 Stopping an Event Process

1. Click anywhere on the easel to select the Event Process.
2. Click the **Stop** button on the toolbar; all running component instances in the Event Process are stopped and the Event Process is stopped.

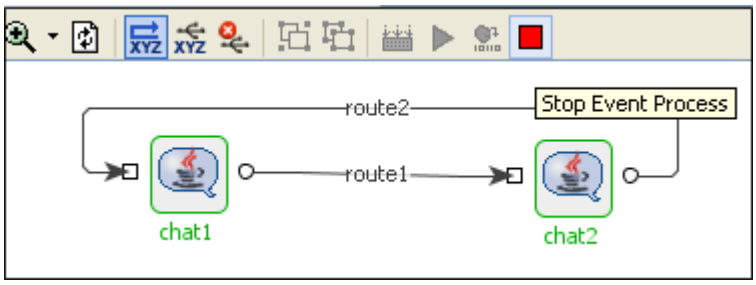


Figure 5.10.4: Stopping an Event Process

Note: When a component is killed its name turns red in the orchestration easel.

5.10.3 Synchronizing Event Processes

Fiorano has the capability to modify existing running applications on the fly. To do this, add another component-instance to the flow from the component palette, configure the component, and place and configure the appropriate routes. The application then needs to be *synchronized* to reflect the changes.

Figure 5.10.5 illustrates the toolbar button synchronizes changes made to a running event process.

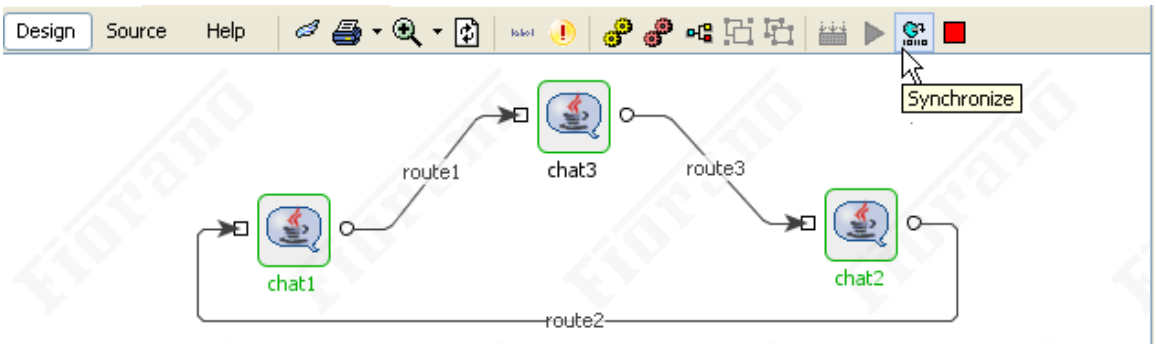


Figure 5.10.5: Event Process Synchronization

Synchronization occurs across the whole application. The effect of synchronization is to save the new application as updated with new components, new routes and any new configurations.

5.10.4 Launching and Stopping Individual Components

It is also possible to stop and then launch individual components within a running Application.

To stop an individual component, right-click the Component and select **Stop** from the drop-down menu. Alternatively, you can select the component and click the **Stop** button:

To restart a stopped component service in a running application, select the component and then click the **Run** button as shown in Figure 5.10.6.

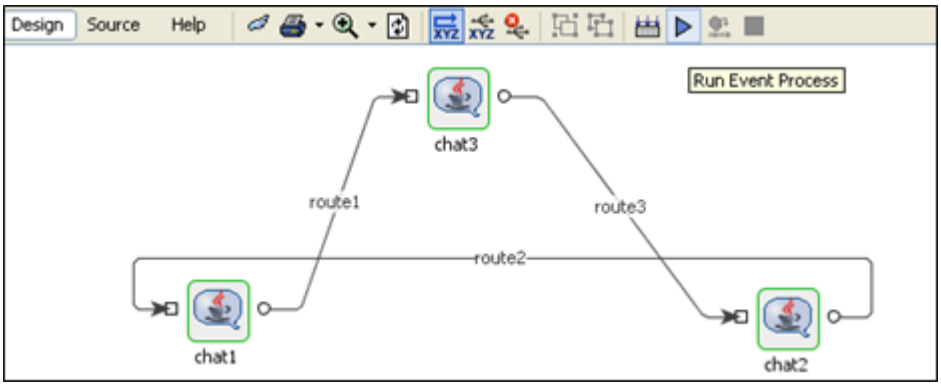


Figure 5.10.6: Restart and Stop Running Application

Alternatively, you can right-click the component and select the **Run** from the menu list as illustrated in Figure 5.10.7.

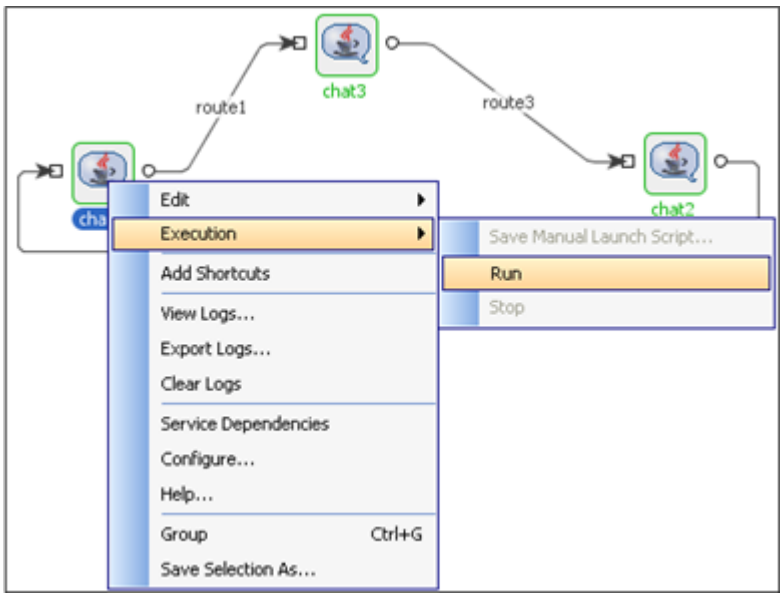


Figure 5.10.7: Launch Application

5.11 The Event Process Command Line Interface

In addition to the visual interface of the orchestrator, Fiorano provides a command line interface to launch and perform other operations on an entire Event Process and/or particular components within an Event Process (those configured with a manual launch type).

The command line interface for launching Event Processes is based on Ant tasks and is available in the installation directory at

Windows

1. Naviate to %FIORANO_HOME% /esb/tools/cli
2. Run **cli.bat** file
3. Type command `ant <targetname>`

Linux

1. Naviate to $FIORANO_HOME/esb/tools/cli
2. Run **cli.sh** file (this wil run the default target in the build file, that is, launchApps)
3. The user can also specify the target by using `./cli.sh <targetname>`

Figure 5.11.1 displays the tree listing in typical windows installation of the platform

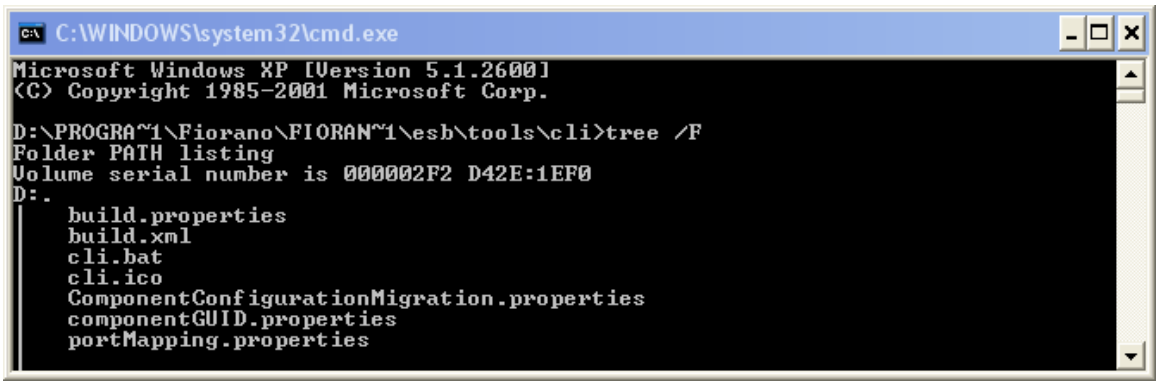


Figure 5.11.1: Tree Type Listing

- Depending on the operating system you are using, the cli.bat *or* cli.sh file sets up the environment to interact with the exposed Fiorano API. Ant commands can be executed from the cli.bat (or cli.sh, as appropriate) files to issue requests to the Fiorano Enterprise Server.
- The build file for ant tasks is build.xml. This contains the list of ant tasks provided by Fiorano by default to interact with the exposed API.
- The build.properties file is the properties file for the build.xml file containing the ant tasks.

5.11.1 List of Ant Tasks provided by command Line Interface

Following are the list of ant tasks available in Command Line Interface tool

1. **importApps** - imports the list of applications that are specified for property IMPORT_APPLICATION_LIST in build.properties file. The directory from which the applications to be imported can be specified with APPLICATION_IMPORT_DIR. if the property OVERWRITE is true, it will overwrite the existing application.
2. **importwithlibs** - by using this target one can import the application with services that are required by application. The application Zip file name can be specified with APPLICATION_ZIP in build.properties file.
3. **exportApps** - exports the list of applications that are apecified for property EXPORT_APPLICATION_LIST in build.properties, provided the application exists in Fiorano Application repository. The directory to which the applications to be exported can be specified with APPLICATION_EXPORT_DIR.
4. **exportAppsWithLibs** - by using this target one can export the list of applications specified for property EXPORT_APPLICATION_LIST in properties file along with services that are used in application. The property APPLICATION_EXPORT_DIR_LIB can be used to give the path of Export directory.
5. **importServices** - Import Business Components from IMPORT_SERVICES_LIST

   If the property OVERWRITE_IMPORT_SERVICE is true while importing the service, it will overwrite the existing service.
6. **exportServices** - Export Business Components from EXPORT_SERVICES_LIST.
7. **launchApps** - Launch all the applications specified in LAUNCH_APPLICATION_LIST
8. **stopApps** - Stop all the applications specified in STOP_APPLICATION_LIST
9. **replace** - Replaces properties in Event process, the properties file to be specified as value of PROPERTIES_FILE in build.properties file.
10. **stopServices** - Stops the list of services that are specified for property STOP_SERVICES_LIST in build.properties file . The services to be stopped belong to the Application specified by the property STOP_SERVICES_OF_APPLICATION. Only one application name can be specified with the property STOP_SERVICES_OF_APPLICATION.
11. **stopAllServices** - Stops all Services of the list of applications that are specified by the property STOP_ALLSERVICES_OFAPPLICATION_LIST in build.properties file.
12. **startServices** - Starts the list of services that are specified for property START_SERVICES_LIST. The services to be started belong to the Application specified by the property START_SERVICES_OF_APPLICATION. Only one application name can be specified with the property START_SERVICES_OF_APPLICATION. The application whose services are to be started should be running.
13. **startAllServices** - Start all Services of the list of applications that are specified by the property START_ALLSERVICES_OFAPPLICATION_LIST in build.properties file. All the applications whose services are to be started should be running.

5.11.2 Launching an Event Process from Command Line

To launch an event process, navigate the build.properties file to edit the value of properties required by the target launchApps.

LAUNCH_APPLICATION_LIST=<Comma Separated Event Processes>

For example: LAUNCH_APPLICATION_LIST=SimpleChat, SimpleDemo

Figure 5.11.2 illustrates a snippet from a build.xml file.

```
<target name="launchApps" description = "Launch all the applications specified in LAUNCH_APPLICATION_LIST">
<fiorano:foreach
                list="${LAUNCH_APPLICATION_LIST}"
                delimiter=","
                param="APPLICATION_GUID"
                target="launchSingleApp_internal"
                inheritall="true"/>

</target>
```

Figure 5.11.2: Build.xml Snippet

Now from the command prompt, the command: **ant launchApps** launchs all the event processes specified in LAUNCH_APPLICATION_LIST.

All the properties accessed in this command is available in the build.properties file.

- In build.properties, change the value of LAUNCH_APPLICATION_LIST to the name of the event process to be started. All other values are provided for a default connection to an enterprise server on the local machine and can be changed as required. Then simply call the ant task on the command line as shown below.

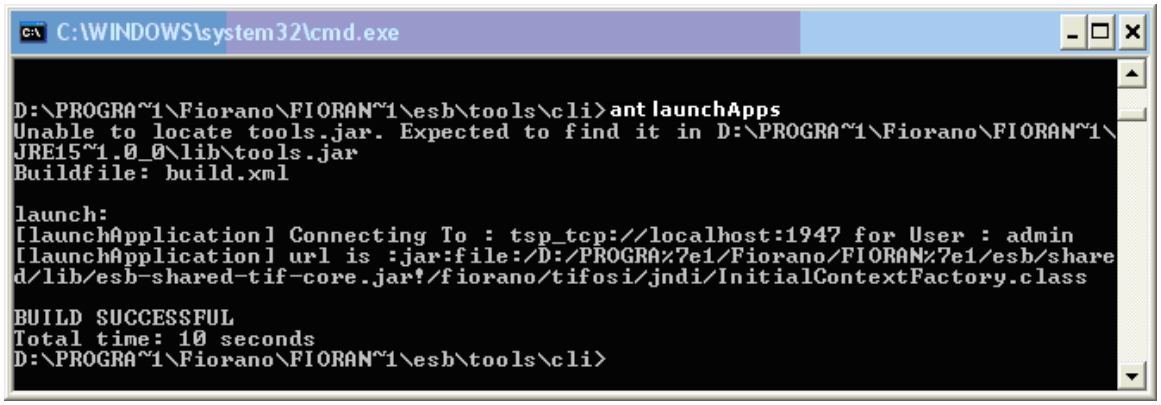


Figure 5.11.3: Build Properties

- If the build is successful that means the command has been issued to the Enterprise Server successfully and the Application is launched.
- It is also possible to loop over these basic ant tasks provided by Fiorano.

5.11.3 Launching Components from Command Line

Components whose **Launch type** is set to Manual may be launched from the command line. The interface for launching such components is available in the installation directory at

%FIORANO_HOME% /esb/tools/scriptgen

The following screen displays the tree listing in a typical windows installation.

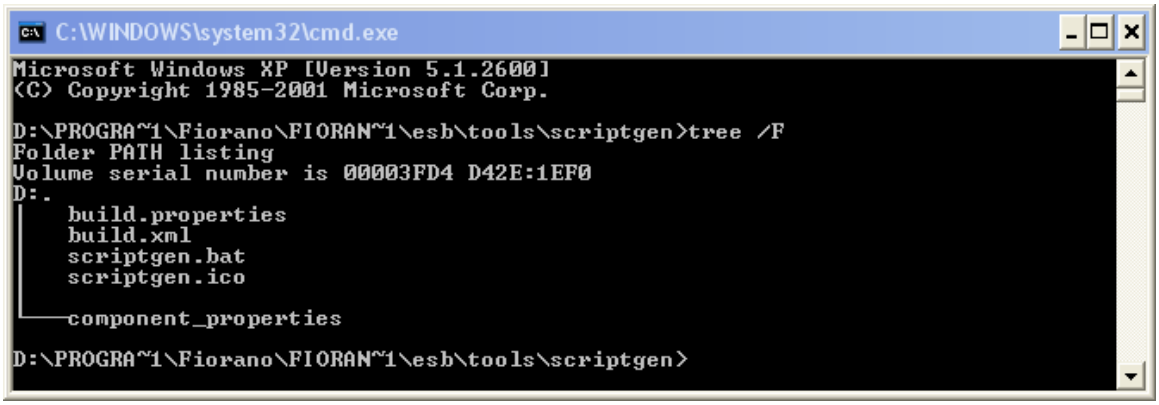


Figure 5.11.3: Launching Components from Command Line

1. Depending on the Operating system you are using, the cli.bat or cli.sh file sets up the environment to interact with the exposed Fiorano API. Ant commands can be executed from the cli.bat (or cli.sh, as appropriate) files to issue requests to the Fiorano Enterprise Server.
2. The build file for ant tasks is build.xml. It contains the list of ant tasks provided by Fiorano by default to initiate the manual launch of particular components.
3. The properties file for the build.xml is build.properties.
4. The component_properties folder includes the manual launch scripts, although they can be stored anywhere.

5.11.4 Executing Components Manually

To launch a component from an external source, a script defining the properties for that component is required. To get a manual launch script for a component configured to be executed in manual mode

1. Select the component and right-click on it. A menu list appears.
2. Select **Execution** and choose the **Save Manual Launch Script** option.

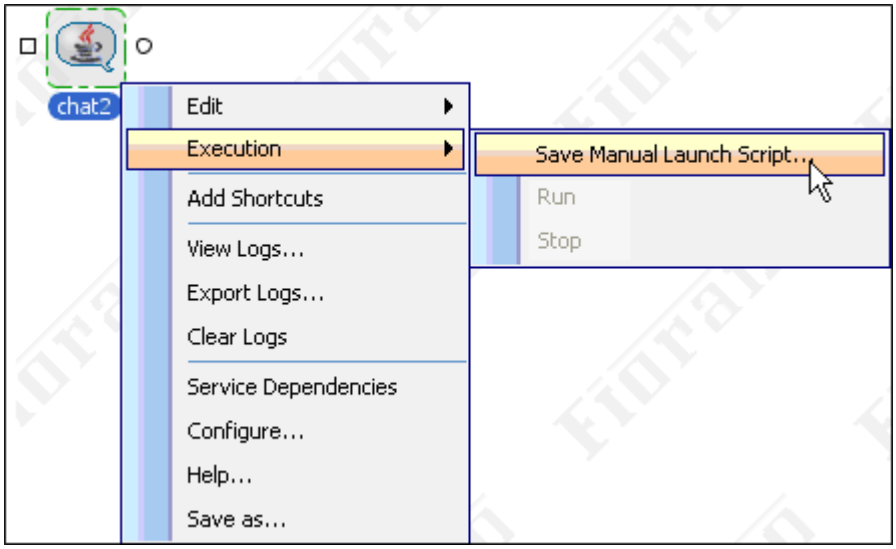


Figure 5.11.4: Save Manual Launch Script

Note: A prompt for saving the script in the default component_properties directory is shown and you can choose to save the script in a directory of your choice.

3. Invoke the command line interface using scriptgen.bat and build the scriptgen by calling ant which has the default ant target launch

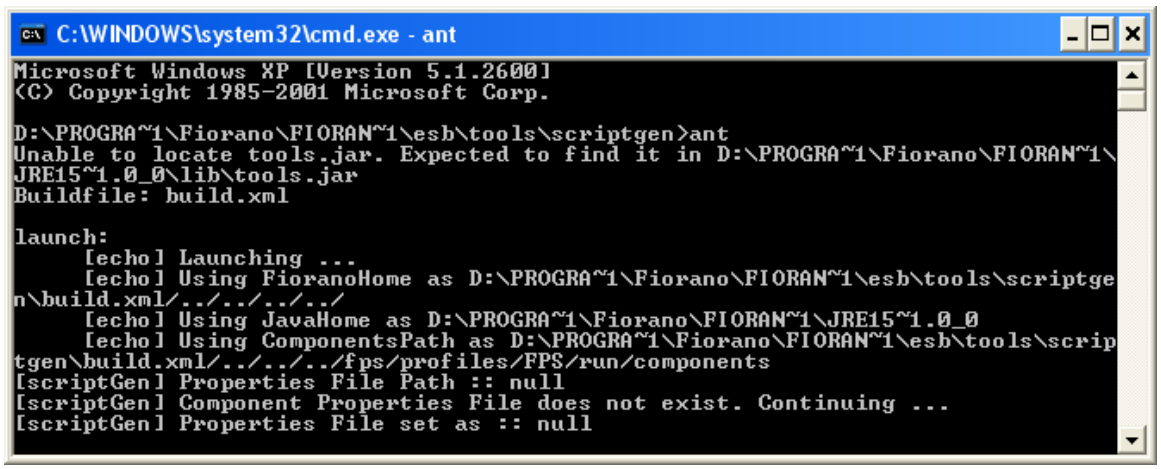


Figure 5.11.5: The Command Line Interface

4. This launches the GUI shown in Figure 5.11.6. Click on the ellipsis to point to the manual launch script which was saved and then click the **Load** button.

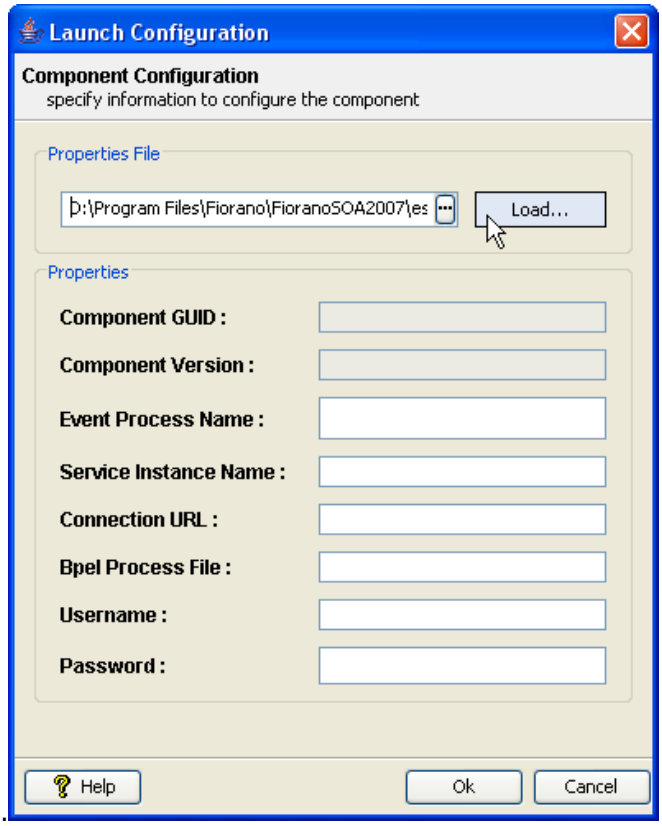


Figure 5.11.6: Component Configuration Screen

5. The properties from the launch script are loaded. Enter the username and password in the **Username** and **Password** fields and click the **Ok** button.

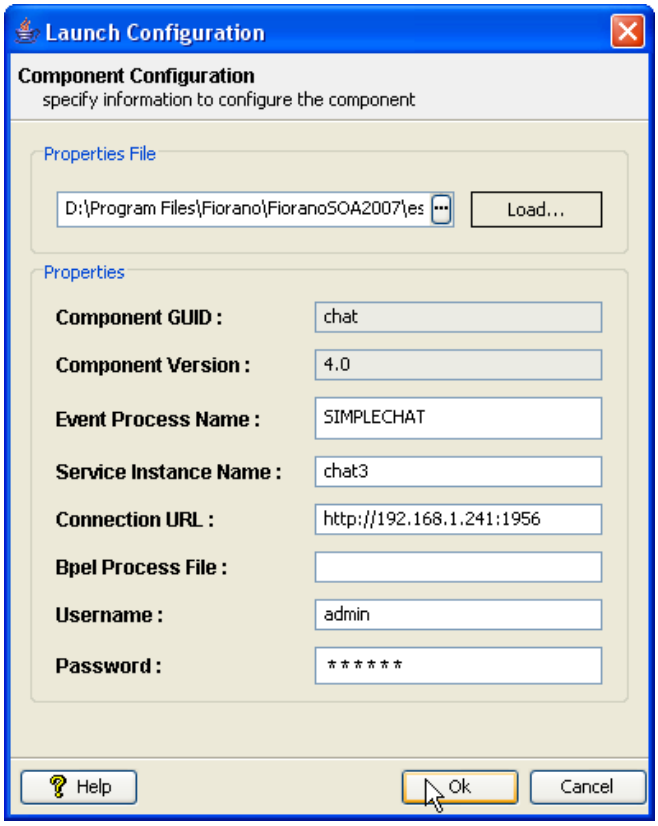


Figure 5.11.7: Launch Configuration Dialog box

This issues the command to the Enterprise Server to launch the component configured manually and the component is launched.

5.12 Best Practices in Deployment

A typical business problem is solved using a set of business processes, each of which is implemented of a group of flows. Each flow should represent a unit of business operation. A series of flows communicating with each other and performing units of business operations enable complex business problems to be solved in a modular fashion.

Fiorano Event Process Orchestrations capture the essence of Service-Oriented development where each Service/Component performs its own function. In this case a group of self-contained Fiorano components connected via routes for information exchange perform units of business operations, achieving modularity by segregation of tasks while at the same time providing an entirely visual composition.

Best practices for Event process development include:

- Model each Event Process to perform one or a small number of business operation(s).
- To speed development, use separate teams of developers to develop different Event Processes.
- Event Processes communicate with each other using port bindings, as explained in the next section
- Ensure that all the components have appropriate names mapping their business or technical functions.

All external dependencies for components must be noted and added to the resources for the relevant components using the Fiorano Services and Security Manager tool.

- To avoid cluttering in the easel, visual shortcuts for a configured component should be placed to ensure clarity.
- It is recommended to keep the event-process files created and or modified in version and/or source control systems. The files typically include the following:
    - Enterprise Server and Peer Server profiles
    - Event Processes
    - Custom service components
    - Custom mapping functions
- Choosing proper Destination names (Queues/Topics) in cases where components communicate via explicit JMS destinations.

Process to Process Communication - Using Port Bindings

The ports of a component are simple JMS destinations. To send a message from one business component to another within the same Event Process routes are explicitly created; each route creates on-the-fly destinations without programmer intervention to get messages from senders to receivers. To send messages across Event Processes the ports of components must be bound to a particular JMS destination.

For an outport of a component, this implies that messages processed by the component are sent to a particular bound destination.

For an inport of a component this implies that the component has a subscriber on a destination and receives its messages from that particular bound destination only.

5.12.1 Creating Port Bindings Between Components in Different Event Processes

1. Select the end flow business component's OUT_PORT to view its properties. Under **JMS Destination** select **Use Specified Destination** to **yes** and choose a **Destination Name**, as shown in Figure 5.12.1.

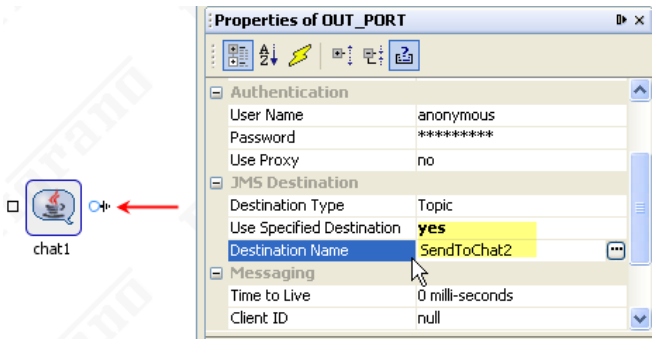


Figure 5.12.1: Properties of OUT_PORT

2. At the entry point of the other flow, select the IN_PORT for first business component.

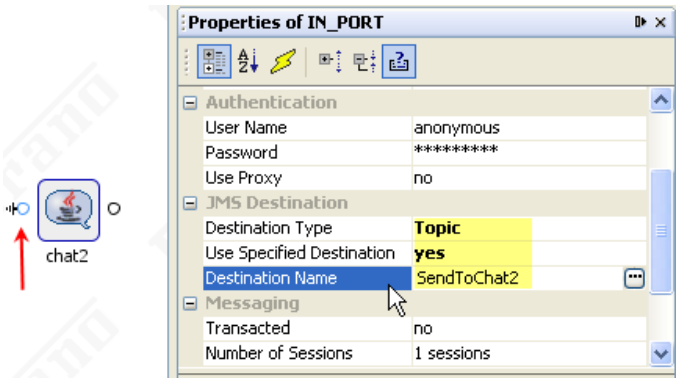


Figure 5.12.2: Properties of IN_PORT

Note: The **Destination Type** is changed to type Topic (the exit point for Flow one), "Use Specified Destination" is set to 'yes' and the "Destination Name" is set to the destination from which the message is to be received.

Figure 5.12.3 illustrates how a port bound queue or topic appears visually

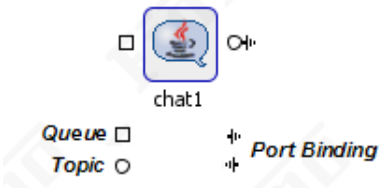


Figure 5.12.3: Port Bound Queue

5.13 Testing Event Processes

If event processes are modeled as Services performing a unit of business operation then the entry point for a flow can be defined. A feeder (comprising the appropriate XSD and a sample xml message) can be placed at the entry point to sanity test the execution task for the required flow.

This feeder component can then later be removed.

To test scenarios on which the execution of a component depends upon the content of the input received, a route interceptor can be placed and values changed on the fly to check the validity of each execution.

5.14 Sample Event Processes

All the files required to run these sample event processes are available in ***applications*** directory as per the following path:
%FIORANO_HOME%\esb\fes\repository\applications.

Fiorano SOA Platform includes a rich set of Sample Event Processes that demonstrate its use in real environments. Based on real use-case scenarios these easy-to-run event processes illustrate how the various components of Fiorano SOA Platform integrate and enable you to solve problems at various verticals.

Before running these sample event processes, please ensure that the following components are running on your Fiorano SOA Platform network.

- Fiorano ESB Server
- Fiorano ESB Peer Server
- Studio

5.14.1 Bond Trading

Scenario

This demo demonstrates a typical Bond Trading scenario wherein messages are exchanged at a very fast rate between various parties.

The prices of the bonds change at a fast rate. Buyers keep track of the price and based on the current price can request for a quote from a trading centre. Based on the availability of the Bond at the requested price, the centre can send an appropriate response to the buyer.

In this demo, there are three financial centers viz. London, Singapore, and Boston. They publish their Bonds identified by unique ISINs.The Bond details appear on GUIs for users to keep track of the changing prices. The GUI allows user to buy a bond, by specifying the quantity and asking price. Based on the ISIN, the request is routed to the appropriate centre.

Event Process Diagram

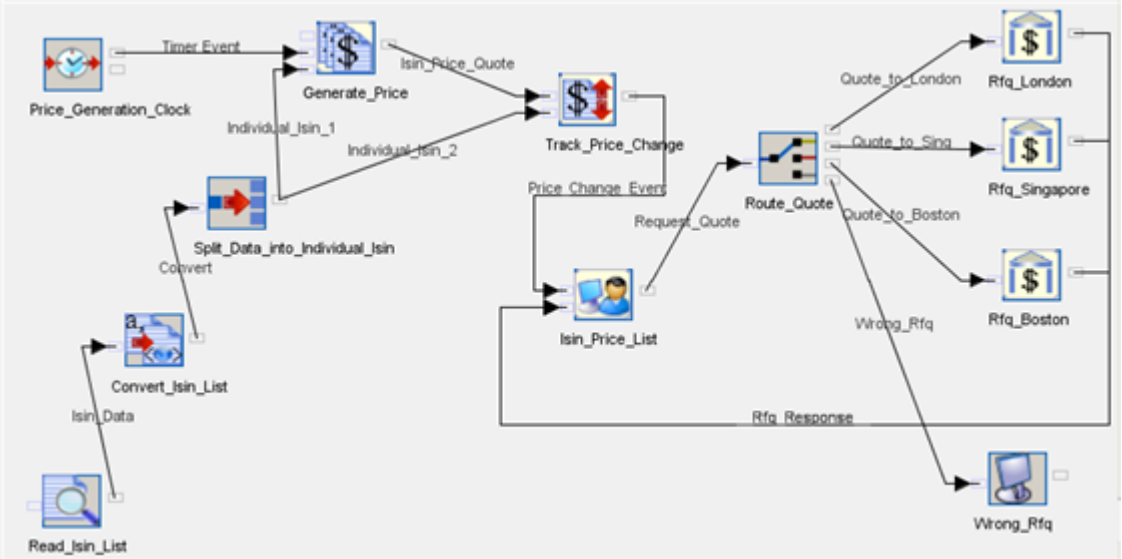


Figure 5.14.1: Bond Trading Event Process Diagram

Components Used

- FileReader
- Text2XML
- XMLSplitter
- CBR (Content Based Routing)
- Prices - Custom Component
- TradeBus - Custom Component
- MarketPricesGui - Custom Component
- RfqManager - Custom Component

Running the Event Process

1. Please ensure that all the Fiorano Servers are running.
2. Invoke the Studio by selecting Start > Programs > Fiorano > Fiorano SOA > Fiorano Tools > Studio.
3. Double-click the event process in the **Event Process Explorer** pane.
4. Perform connectivity and resource check. Launch the event process.

   Two screens appear displaying the Bond details. One is the BOND PRICES-MAINTENANCE SCREEN, the other one is the Bond Pricing screen.
5. Double-click on any ISIN in the Bond Pricing screen. It opens up a pop-up asking for the number of bonds you want to request for. Specify the size and click on **Request**.

   The response is sent back from the appropriate centre asking the buyer to fill the price.
6. After filling the price, click on **Send Quote**.

5.14.2 Database Replication

Scenario

The Database Replication Demo demonstrates the usage of DBAdapter for database replication.

The event process consists of a CRM service instance, using which PO is generated. This PO is fed into the OracleIn instance of the DBAdapter. The OracleOut writes the PO into the PurchaseOrders table in the Oracle Database.

The OracleOut instance of the DBAdapter, which is monitoring table "PurchaseOrders" for insert operations, gets these records, converts it into XML and passes it on to the SqlServerIn instance of the DBAdapter. The SqlServerIn inserts into the "PurchaseOrders" table in the connected SQL Server database.

Event Process Diagram

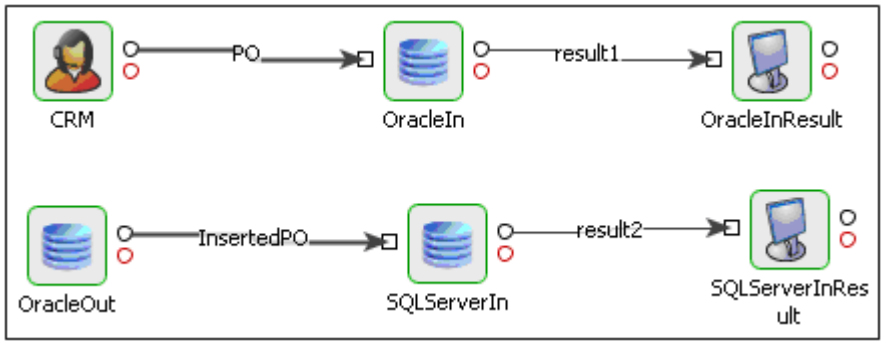


Figure 5.14.2: Database Replication Event Process Diagram

Component Used

- CRM
- DB
- Display

Preliminary Steps

a) If you have not run the batch file **orcl.bat** Located in %FIORANO_HOME%\ esb\ samples\ EventProcesses\ DatabaseReplication\ resources. This batch file creates a schema named tifositest and a table named purchaseorders in the Oracle Database that you can use to run this event process. In case you have already run this batch file and created the schema and table on the database, you can skip this step.

b) If you have not run the batch file **mssql.bat** Located in %FIORANO_HOME%\ esb\ samples\ EventProcesses\ DatabaseReplication\ resources. This batch file creates a schema named TifosiTest and a table named purchaseorders in the MSSQL Database that you will use to run this event process. In case you have already run this batch file and created the schema and table on the database, you can skip this step.

c) Double click on OracleIn Service. This opens its CPS. Edit the JDBC URL to jdbc:oracle:thin:@**<your database location>**:**<your database port>**:**<your database name>**. For example, a typical JDBC URL would look like: jdbc:oracle:thin:@164.164.128.115:1521:orcl. Leave rest of the settings as is. Press the Finish button to save your settings.

d) Do the same for OracleOut Service.

e) Double click on SQLServerIn Service. This opens its CPS. Edit the JDBC URL to jdbc:odbc:Driver={SQL Server};Server=**<your database location>**;Database=**<your database name>**;Uid=TifosiTest;Pwd=TifosiTest;. For example a typical JDBC URL would look like: jdbc:odbc:Driver={SQL Server};Server=164.164.128.115;Database=master;Uid=sa;Pwd=;. Leave rest of the settings as is. Press the Finish button to save your settings.

Running the Event Process

1. Please ensure that all the Fiorano Servers are running.
2. Invoke the Studio by selecting Start > Programs > Fiorano > Fiorano SOA > Fiorano Tools > Studio.
3. Double-click the event process in the **Event Process Explorer** pane.
4. Perform connectivity and resource check. Launch the event process.
5. Send a PO from the CRM Service.

   This adds a record in the database configured in OracleIn service. In addition, a similar record is entered in the database configured in SQLServerIn Service.

5.14.3 EAI Demo

Scenario

The EAI Demo event process is used to show a typical integration scenario, where a Purchase Order (PO) comes from a CRM Service to an ERP Service. The ERP Service sends out an Email to the concerned person if the Order is accepted, or sends out a rejection notice in case the PO is rejected.

Event Process Diagram

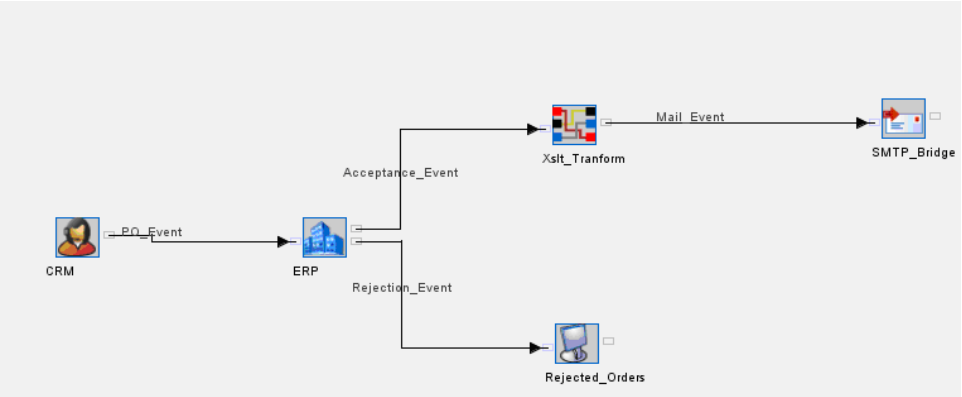


Figure 5.14.3: EAI Demo Event Process Diagram

Component Used

- CRM
- ERP
- XSLT
- SMTPBridge
- Display

Running the Event Process

1. Please ensure that all the Fiorano Servers are running.
2. Invoke the Studio by selecting Start > Programs > Fiorano > Fiorano SOA > Fiorano Tools > Studio.
3. Double-click the event process in the **Event Process Explorer** pane.
4. Perform connectivity and resource check. Launch the event process.
5. Send a purchase order from the CRM Service.
6. The purchase order is received by the ERP Service. When the PO arrives, a pop-up window appears showing that a PO has been received.
7. Close the pop-up window. On closing the pop-up window, another window pops up displaying details of the PO. In that pop-up dialog, either accept or reject the PO.
8. On accepting the order, an acceptance mail is sent to the person who sent the PO via the CRM Service. The default user for this is Ayrton (ayrton@fiorano.com).
9. On rejection, the details of the PO are shown on the Rejected Orders Display Service.

5.14.4 Order Entry

Scenario

This demo is a scaled down version of an implementation of a real life project done by Fiorano at a client site. The application shows how Fiorano SOA Platform can be used to integrate front end applications with varied backend databases and legacy systems utilizing different kinds of communication protocols. The application also shows how a business scenario can be built using the Fiorano SOA Platform Event Process Orchestrator, by simply dragging and dropping various coarse grained components (or Fiorano Services) and orchestrate a business process out of them based on the current business needs.

In this application, a user is provided a web based interface to send a purchase order to a company. The purchase order is received by HTTPReceive component, which has an embedded web server inside it. The purchase order request is inserted in a database table and forwarded to a legacy system. Here we are using the ERP Service as a manual intervention legacy system, which provides facility for the business manager to view details of the order received, and accept or reject them. Based on whether the order is accepted or rejected, corresponding actions are taken. In case the order is accepted, an email is sent to the customer regarding the acceptance. At the same time an HTTP component is used to POST the order delivery request to a third party vendor. If the order is rejected, a rejection mail is sent to the customer.

Note: The message format sent out by the Manual Intervention Service is different from the message format that is expected by the SMTP Service. To provide a seemless and real time transformation of the business documents flowing through the Fiorano Network, Fiorano SOA Platform provides the XSLT component. This component converts the incoming business document format to an outgoing format as per the configurations done in it. It uses the Fiorano Mapper Tool to configure mapping details between the input and the output structures and utilizes the standard XSLT Engine to do the transformation.

Event Process Diagram

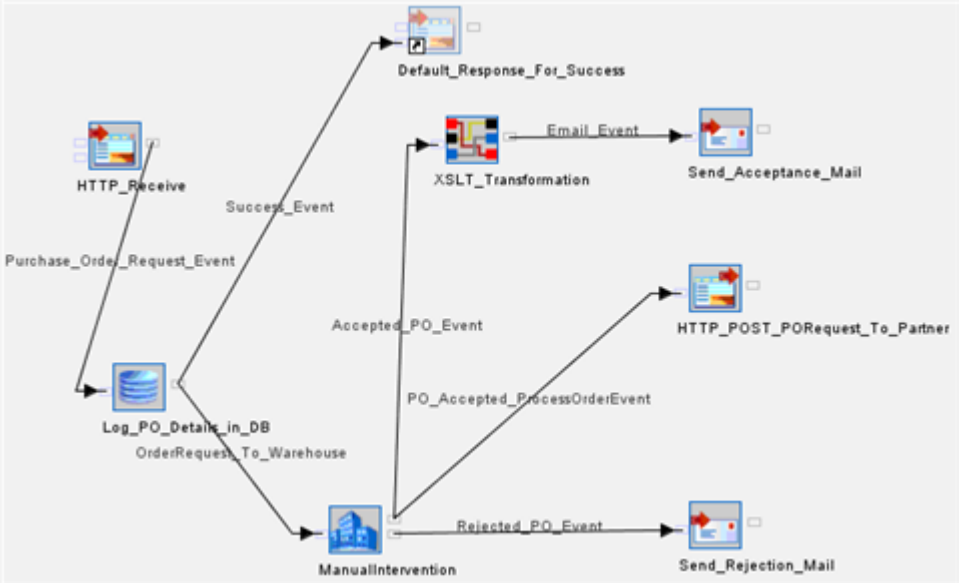


Figure 5.14.4: EAI Demo Event Process Diagram

Components Used

- HTTPReceive
- DB
- ERP
- XSLT
- SMTP
- HTTP (Post)

Running the Event Process

1. Please ensure that all the Fiorano Servers are running.
2. Invoke the Studio by selecting **Start > Programs > Fiorano > Firoano SOA > Fiorano Tools > Studio**.
3. Double-click the event process in the Event Process Explorer pane.
4. Start the HSQL database server by executing this batch file: **starthsql.bat** located in %FIORANO_HOME%\esb\samples\hsql.
5. Also run the batch file **init.bat** from %FIORANO_HOME%\esb\samples\EventProcesses\OrderEntry\resources.

6. Perform connectivity and resource check. Launch the event process.
7. Wait for some time so that all the services become green.
8. Use your web-browser to open the file: Send_PO_Request.html
9. Fill up the form and press the **submit** button.
10. Wait for some time and the order is placed. You will get a default response back. Reach to the UI window of the ManualIntervention Service. It would be showing a PO received dialog box. Close that dialog box.
11. You will see the PO details. Either accept or reject the order.
12. A corresponding email would be sent based on this to the email address which you filled in the form.
13. Once you are done with the demo, run the batch file **cleanup.bat** from %FIORANO_HOME%\esb\samples\EventProcesses\OrderEntry\resources.
14. Also, shut down the HSQL server by pressing Ctrl+C in the window in which it is running.

5.14.5 Portal Integration

Scenario

This event process shows how an HTTP Client can communicate with a back-end database using Fiorano SOA Platform. An HTTP request from a client like your default web-browser is sent to HTTPReceive service, which generates a "Get_PO_Details_Event" out of it. The event is read by the DB adapter and details of the PO provided in the event, are send as PO_Details_Event to XSLT. HttpReceive reads the Final_PO_Details_Event and sends the PO Details back to the client based on the configurations done in it.

Event Process Diagram

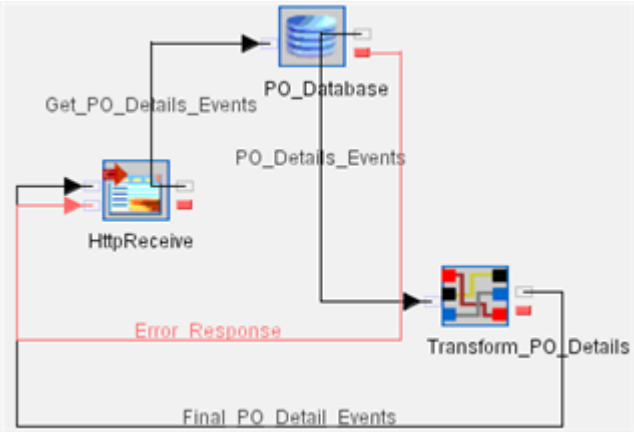


Figure 5.14.5: Portal Integration Event Process

Component Used

- HTTPReceive
- DB
- XSLT

Running the Event Processes

1. Start the HSQL database server by executing this batch file: **starthsql.bat** located in %FIORANO_HOME%\esb\samples\hsql.
2. Also run the batch file **init.bat** from %FIORANO_HOME%\esb\samples\EventProcesses\PortalIntegration\resources.
3. Perform connectivity and resource check. Launch the event process.
4. Use your web browser to open the *GetPODetails.html* page.
5. Click the **Submit** button.
6. Wait for some time while the details are fetched from the DB and displayed on your web browser
7. Once you are done with the demo, run the batch file **cleanup.bat** from %FIORANO_HOME%\esb\samples\EventProcesses\PortalIntegration\resources.
8. Also shut down the HSQL server by pressing Ctrl+C in the window in which it is running.

FAQ

Why does my html page not respond for long time on pressing submit?

When the HttpReceive Service starts, it takes some time to start the embedded HTTP Server. It might happen that the HTTP Server didn't start and you pressed the submit button. Try resending the request. Alternatively, restart the event process, and wait for a few moments after every service becomes green and then send the request.

5.14.6 Purchasing System

Scenario

This demo illustrates how Fiorano SOA Platform can be used to integrate front end applications with backend databases and web based systems with different kinds of communication protocols. The application also shows how a business scenario can be built using the Studio, by simply dragging and dropping various coarse grained components (or Fiorano Services) and orchestrates a business process out of them based on the current business needs.

This application involves the following steps.

1. A purchase request is sent by the user to a company through a web based interface.
2. The purchase order is received by HttpReceive Adapter, which has an embedded web server inside it. The purchase order consists of three inputs namely REQUEST, CREDENTIALS and SYNC_PO_002. REQUEST is an identifier string for the request. CREDENTIALS is an xml string containing the login and password details of the user. SYNC_PO_002 is an xml string containing the actual details about the requested purchase.
3. The purchase details xml is verified to be conforming to a predefined schema by the XMLVerification component.
4. The login and password details are verified by a custom LdapAuthenticator component which connects to an underlying LDAP server and the flow continues only if this authentication is successful.
5. The request identifier is verified by a CBR (Content Based Router) component to be the correct one for which this system is expected to service the request.
6. The purchase details are then entered into a database and the purchase order is sent for processing to a downstream web based application.
7. Once the processing is complete in the downstream web based application, a confirmation is displayed in the Display component.
8. To provide a seamless and real time transformation of the business documents flowing through the Fiorano Network, Fiorano SOA Platform provides the XSLT Service. This service converts the incoming business document format to an outgoing format as per the configurations done in it. It uses the Fiorano Mapper Tool to configure mapping details between the input and the output structures and utilizes the standard XSLT Engine to do the transformation. Also for checking the validity of an xml document, Fiorano SOA Platform provides the XMLVerification service. And for XPath based content based routing facility, it provides the CBR service.

Event Process Diagram

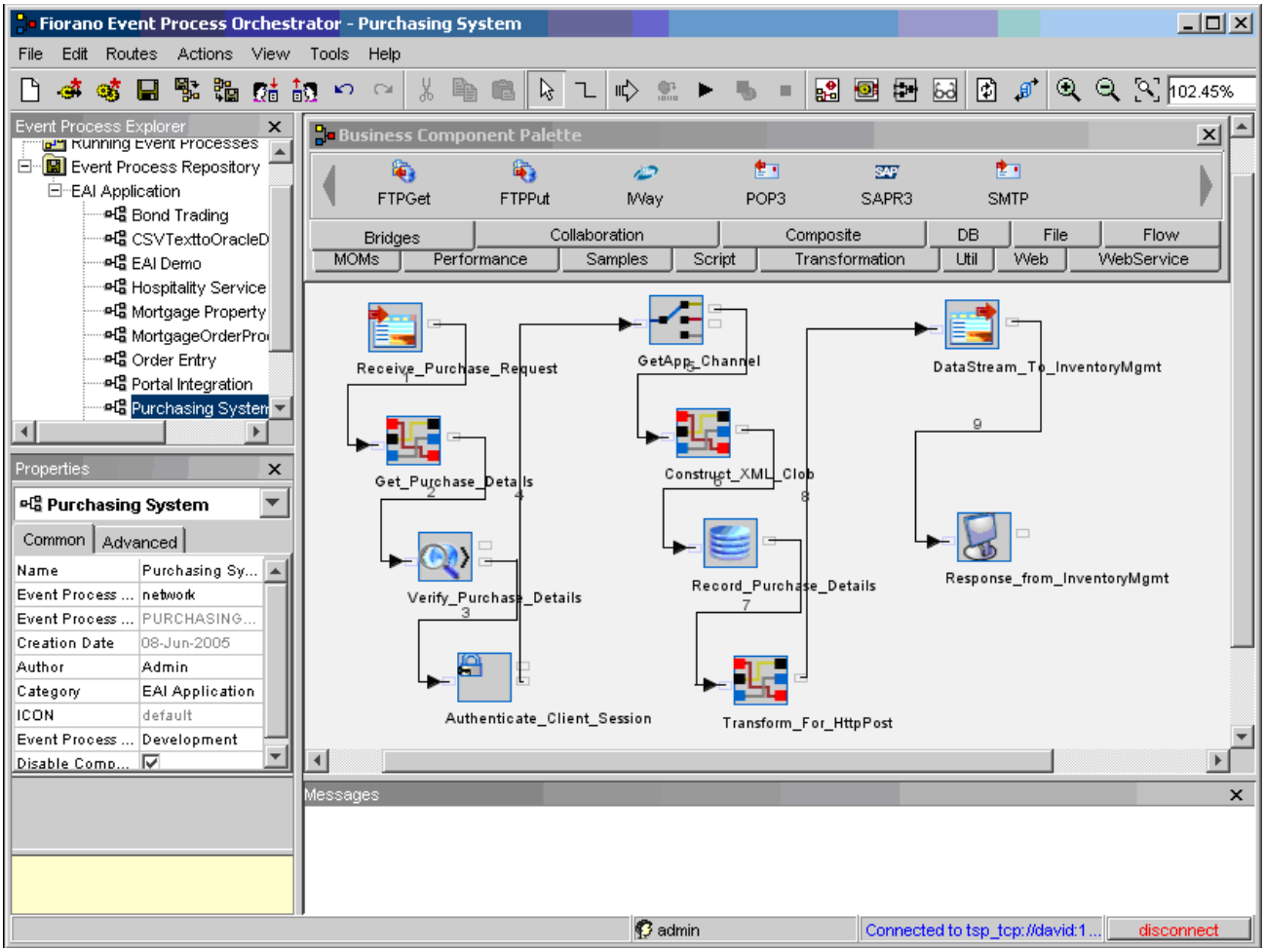


Figure 5.14.6: Purchasing System Event Process Diagram

Components Used

- HttpReceive
- DB
- XMLVerification
- XSLT
- CBR
- HTTPAdapters
- LdapAuthenticator
- Display

Running the Event Processes

1. Ensure that the Fiorano ESB Server and the Fiorano Peer Server are running.
2. Invoke the Studio by selecting Start > Programs > Fiorano > Fiorano SOA Platform > Fiorano Tools > Studio.
3. Double-click the event process in the Event Process Explorer pane.
4. Start the HSQL database server by executing this batch file **starthsql.bat** located in %FIORANO_HOME%\esb\samples\hsql.
5. Also run the batch file **init.bat** from %FIORANO_HOME%\esb\samples\EventProcesses\PurchasingSystem\resources.
6. Perform connectivity and resource check. Launch the event process.
7. Wait for some time so that all the services become green.

   Run **PurchasingSystem_Install.bat** (on Windows) or **PurchasingSystem_Install.sh** (on Linux) located in %FIORANO_HOME%\esb\samples\EventProcesses\PurchasingSystem\resources
8. Start the Fiorano ESB Web Container.
9. If you are using Windows, start the pre-configured Open LDAP server by running **slapd.exe -d 1** from %FIORANO_HOME%\esb\samples\openldap. If you are using Linux, please read the file **LdapSetup_Linux.txt** located in %FIORANO_HOME%\esb\samples\PurchasingSystem\docs for instructions on how to set up the Open LDAP server for Linux.
10. Use your web-browser to open the file *PurchasingSystem_Input.html* located in %FIORANO_HOME%\esb\samples\EventProcesses\PurchasingSystem\resources.
11. The sample data given in the file **PurchasingSystem_SampleInput.txt** located in %FIORANO_HOME%\esb\samples\EventProcesses\PurchasingSystem\resources can be used to fill the input form. For conveneince, the form has already been filled.
12. Fill up the form and press the **Submit** button.
13. Wait for some time and the request is sent.
14. Look at the Display window that opens automatically. Once the processing is complete, an appropriate message appears in it.
15. Once you are done with the demo, run the batch file **cleanup.bat** from %FIORANO_HOME%\esb\samples\EventProcesses\PurchasingSystem\resources.
16. Shut down the HSQL server by pressing Ctrl+C in the window in which it is running.
17. Also, shut down the ESB web container and the standalone LDAP server.

FAQ

Why does my html page not respond for long time on pressing submit?

When the HttpReceive Service starts, it takes some time to start the embedded HTTP Server. It might happen that the HTTP Server didn't start and you pressed the submit button. Try resending the request. Alternatively, restart the event process, and wait for a few moments after every service becomes green and then send the request.

5.14.7 Retail Television

Scenario

This demo is a scaled down version of a real life implementation for a leading television network in retail. The application also shows how a business scenario can be built using the Studio, by simply dragging and dropping various coarse grained components (or Fiorano Components) and orchestrating an event process based on the current business needs. In this application, a media production request arrives in a specific directory. The directory is polled using Fiorano's File Reader. The request is written to a database table using Fiorano's DB adapter. This table is monitored for insertion events by another instance of Fiorano's DB adapter. The request is transformed and further processed by a web service in order to create a preview for the request.

Event Process Diagram

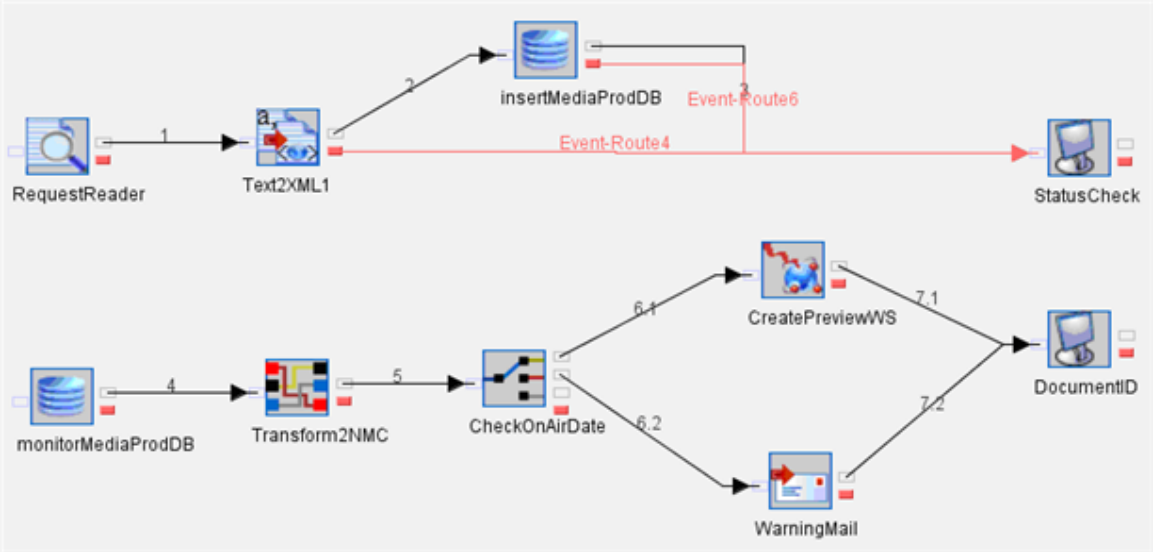


Figure 5.14.7: Retail Television Event Process Diagram

Component Used

- FileReader
- Text2XML
- XSLT
- Web Service Consumer
- CBR
- SMTP
- DB Adapter

Running the Event Processes

- Please ensure that all the Fiorano Servers are running.
- Invoke the Studio by selecting **Start > Programs > Fiorano > Fiorano SOA > Fiorano Tools > Studio**.
- Double-click the event process in the Event Process Explorer pane.
- Start the ESB Web Container from **Start** > **Programs** > **Fiorano** > **Fiorano SOA** > **Fiorano Servers** > **ESB Web Container**.
- Start a console window from **Start** > **Programs** > **Fiorano** > **Fiorano SOA** > **Fiorano ESB Console** > **EDBC Sample**. Change directory to EventProcesses\RetailTelevision\resources and deploy the web service by executing **ant** on the console.
- Start the HSQL database server by executing this batch file: `starthsql.bat` located in %FIORANO_HOME%\esb\samples\hsql.
- Also run the batch file `init.bat` from %FIORANO_HOME%\esb\samples\EventProcesses\RetailTelevision\resources.
- Perform connectivity and resource check. Launch the event process.
- Wait for some time so that all the services become green.
- Create a directory called `production_requests` under %FIORANO_HOME%\esb\samples\EventProcesses\ RetailTelevision.
- Go to the directory %FIORANO_HOME%\esb\samples\EventProcesses\ RetailTelevision\resources. Copy the file `media_production_request_1.txt` into the directory called `production_requests`.
- The last date field in the file determines whether a warning email is sent or the create preview service is called. You can change the day in the date to be greater than 10 so that an email is sent.
- Once you are done with the demo, run the batch file `cleanup.bat` from %FIORANO_HOME%\esb\samples\EventProcesses\RetailTelevision\resources.
- Also, shut down the HSQL server by pressing Ctrl+C in the window in which it is running.

5.14.8 Revenue Control Packet

Scenario

This demo is a scaled down version of a real life implementation at a customer site. It represents a very typical situation in a production support environment where the data in different systems needs to be synchronized. The data modification communication between the systems involved is in the form of proprietary flat files. The application also shows how a business scenario can be built using the Studio, by simply dragging and dropping various coarse grained components (or Fiorano Components) and orchestrating an event process based on the current business needs.

Orchestrating this application / event process involves the following steps.

1. A particular directory is being monitored for files of type *.LDF.
2. The contents of these files are in proprietary format. So a custom component is used to
    a. Extract the content of these files,
    b. Convert the extracted flat- file data into its corresponding XML,
    c. Then, use the same to update a back-end database.
3. Errors occurring in any of the intermediate steps are
    a. Displayed
    b. Written to a file
    c. Communicated via e-mail to a particular user

To provide a seamless real-time transformation of the business documents flowing through the composed Fiorano Network, Fiorano SOA Platform provides the XSLT component. This Fiorano component converts the incoming business documents to an outgoing format as per its configurations done at the time the event process was composed. It uses the Fiorano Mapper Tool to configure mapping details between the input and the output structures and utilizes the standard Xslt Engine to perform the required transformations.

Event Process Diagram

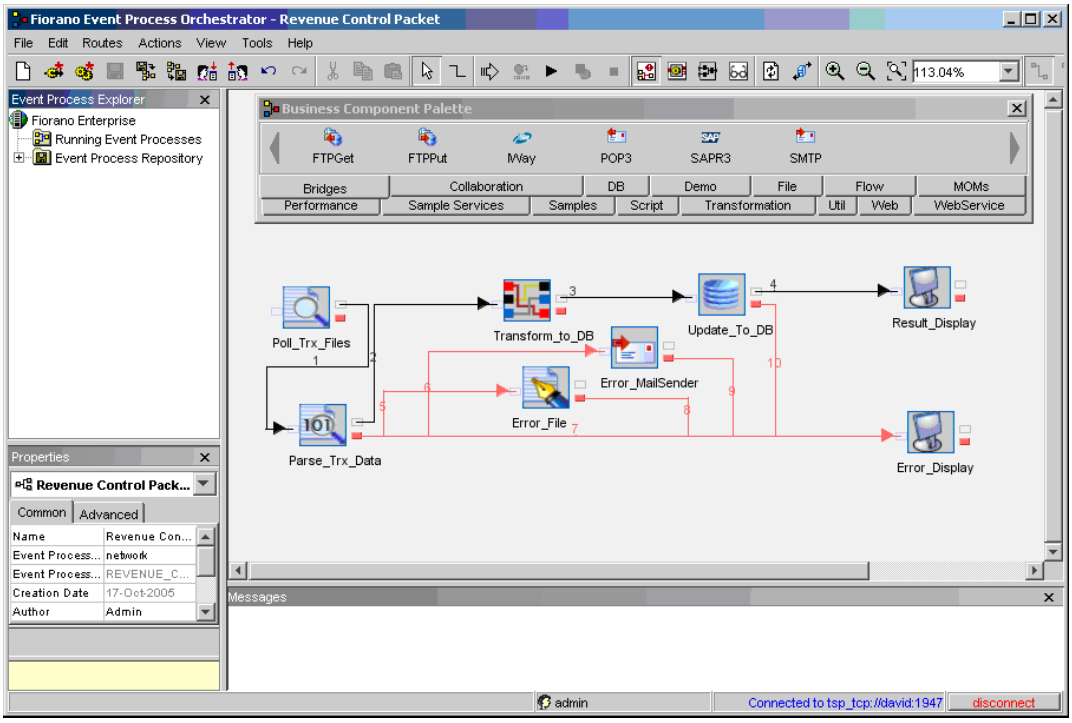


Figure 5.14.8: Revenue Control Packet Event Process Diagram

Component Used

- FileReader
- BinaryFileReader
- DB
- XSLT
- FileWriter
- SMTP
- Display

Running the Event Processes

1. Ensure that the Fiorano Enterprise Server and the Fiorano ESB Peer are running.
2. Invoke the Studio by selecting **Start > Programs > Fiorano > Fiorano SOA Platform > Fiorano Tools > Studio**.
3. Double-click the event process in the Event Process Explorer pane.
4. Start the HSQL database server by executing this batch file `starthsql.bat` located in %FIORANO_HOME%\esb\samples\hsql.
5. Also run the batch file `init.bat` from %FIORANO_HOME%\esb\samples\EventProcesses\RevenueControlPacket\resources.
6. Perform the Connectivity and Resource Check. Launch the event process.
7. Wait for some time so that all the services become green.
8. Run the batch file `RevenueControlPacket_Install.bat` located in %FIORANO_HOME%\esb\samples\EventProcesses\RevenueControlPacket\resources.
9. View the Display windows that opens automatically. Once the processing is complete, an appropriate message appears in at least one of the Display windows.
10. Once you are done with the demo, run the batch file **cleanup.bat** from %FIORANO_HOME%\esb\samples\EventProcesses\RevenueControlPacket\resources.
11. Also, shut down the HSQL server by pressing Ctrl+C in the window in which it is running.

5.14.9 Simple Chat

Scenario

The SimpleChat event process initiates a chat session on two or more nodes of a Fiorano SOA network. This event process also demonstrates how messages flow from one service to the other in a Fiorano SOA network

Event Process Diagram

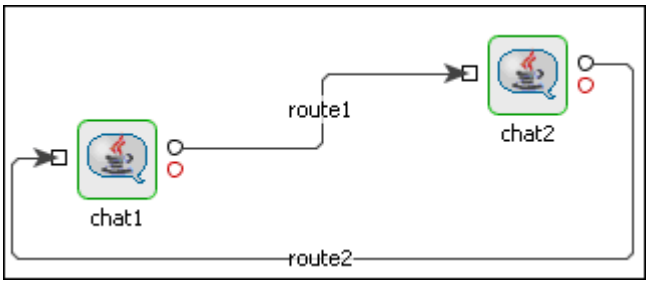


Figure 5.14.9: Simple Chat Event Process Diagram

Component Used

- Chat

Running the Event Processes

1. Please ensure that all the Fiorano Servers are running.
2. Invoke the Studio by selecting Start > Programs > Fiorano > Fiorano SOA > Fiorano Tools > Studio.
3. Double-click the event process in the **Event Process Explorer** pane.
4. Perform connectivity and resource check. Launch the event process.
5. Send some messages from Chat1. You will receive them on Chat2.
6. Alternatively you can send messages from Chat2 and they is received at Chat1.

5.14.10 WorkList Sample

Scenario

This demo represents a real life scenario which allows Enterprises to define 'Manual intervention' based business processes quickly as well as define rules to perform time based or role-based escalation of specific tasks. The WorkList business service is used for managing XML documents by introducing a pause in the path of a document in a workflow event process. It uses RDBMS to store the XML documents arriving from various event processes running on the Fiorano Network. The business service stores the content until it receives an external signal triggering a release of its content. The WorkList Service can be found in the 'Flow' category of the Studio.

Event Process Diagram

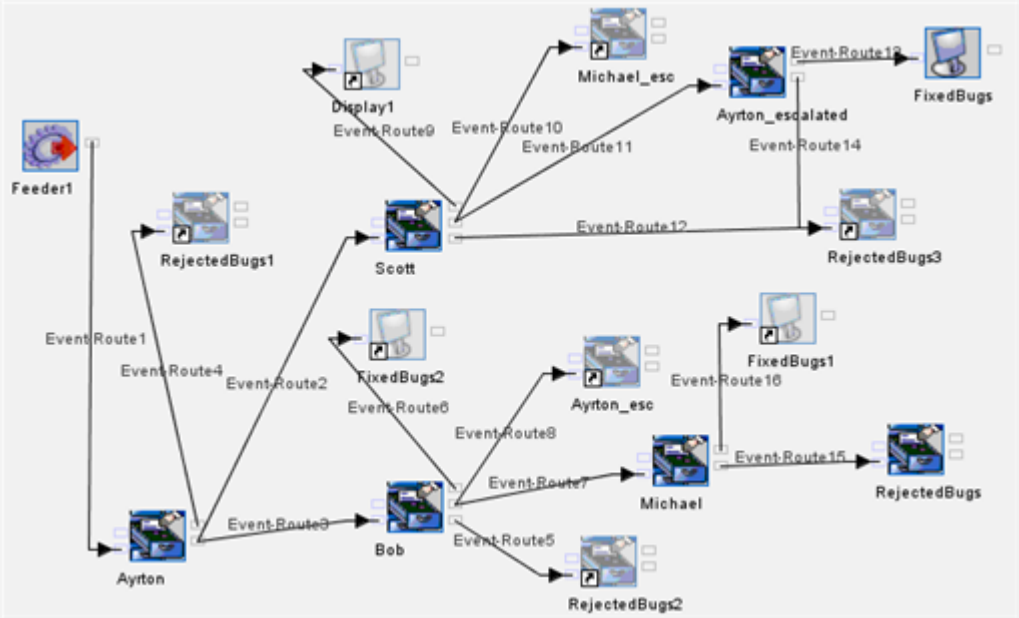


Figure 5.14.10: WorkList Sample Event Process Diagram

Component Used

- WorkList
- Feeder
- Display

Running the Event Processes

1. Please ensure that all the Fiorano Servers are running and you have followed the PRELIMINARY STEPS.
2. Invoke the Studio by selecting **Start > Programs > Fiorano > Fiorano SOA > Fiorano Tools > Studio**.
3. Double-click the event process named `WorkListExample` in the Event Process Explorer pane.
4. Perform connectivity and resource check. Launch the event process.

5. Wait for some time so that all the services become green.
6. In the feeder check the replace $index option.
7. Send some messages by clicking the **Send N times** button.
8. In the web browser login using login: ayrton, password: senna.
9. FES admin and password details should be provided in a properties file as shown below:

   FES_ADMIN_USER=admin

   FES_ADMIN_PASSWD=password
10. A system property FES_URL_PROPERTIES whose value is the path of file mentioned above should be added in `fes.conf`. Entry should be made as:

    FES_URL_PROPERTIES=C:/fioranodev/svn/head/installer/esb/fes/bin/fesurl.properties
11. Wait for some time and you would see the messages listed as work items in the worklist.

Chapter 6: High Availability

The importance of 24X7 service availability and the growing importance of conducting real-time business are driving the demand for High Availability Enterprise Systems. The goal is to maximize system availability and eliminate any single point of failure. This chapter discusses the High Availability (HA) features provided by the Fiorano SOA Platform.

The Fiorano SOA Platform provides multiple tiers of High Availability:

1. Message Level HA
2. Enterprise Server HA
3. Peer Server HA

These multiple levels of High Availability eliminate the requirement for expensive RAID, OS clustering software, or third-party HA frameworks in the messaging layer. No matter how complex in-process transactions continue to completion without any expensive rollback or recovery time.

Fiorano provides complete flexibility to administrators giving them an option to either use a shared database (between active and passive server) or use database replication (from active to passive server). So in scenarios where it is not possible to share the database, administrators can still use Fiorano's High Availability Enterprise Systems using the in-built replication support.

6.1 ESB Server High Availability

This section describes how Failover can be configured on the Fiorano Enterprise Server (FES). This feature allows you to specify a backup Enterprise Server as a contingency measure against the failure of the primary Enterprise Server in the Fiorano Network. If the default FES fails, the Event Processes running on the Fiorano network connect to the backup FES and continue to run. The tools do not automatically reconnect to the backup Server; the user has to manually connect them. The Fiorano Enterprise Server HA is implemented in both shared and replication mode. If there is a failure in the Active Server while it was launching an application, the Secondary Server assumes the Active role and starts launching the Event Processes again.

Figure below illustrates the failover connection on the FES.

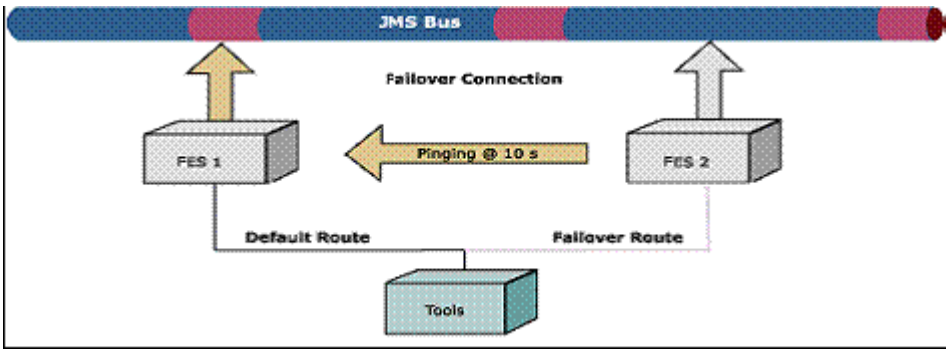


Figure 1: Failover connection on the FES

6.2 Peer Server High Availability

This section describes how Failover can be configured on the Fiorano Peer Server (FPS) in conjunction with FES High Availability. You can specify a Backup Peer Server as a contingency measure against the failure of the Primary Peer Server in the Fiorano network. If the default FPS fails, all components running on the peer connect to the backup FPS and continue to send and receive messages. This reconnection is completely transparent to the components. The state of the application is restored after each failover. The Fiorano Peer Server High Availability is implemented in both shared & replication mode. If there is a failure in the Active Server while it was processing the messages from the components, the Secondary Server assumes the Active role and continues to process the messages.

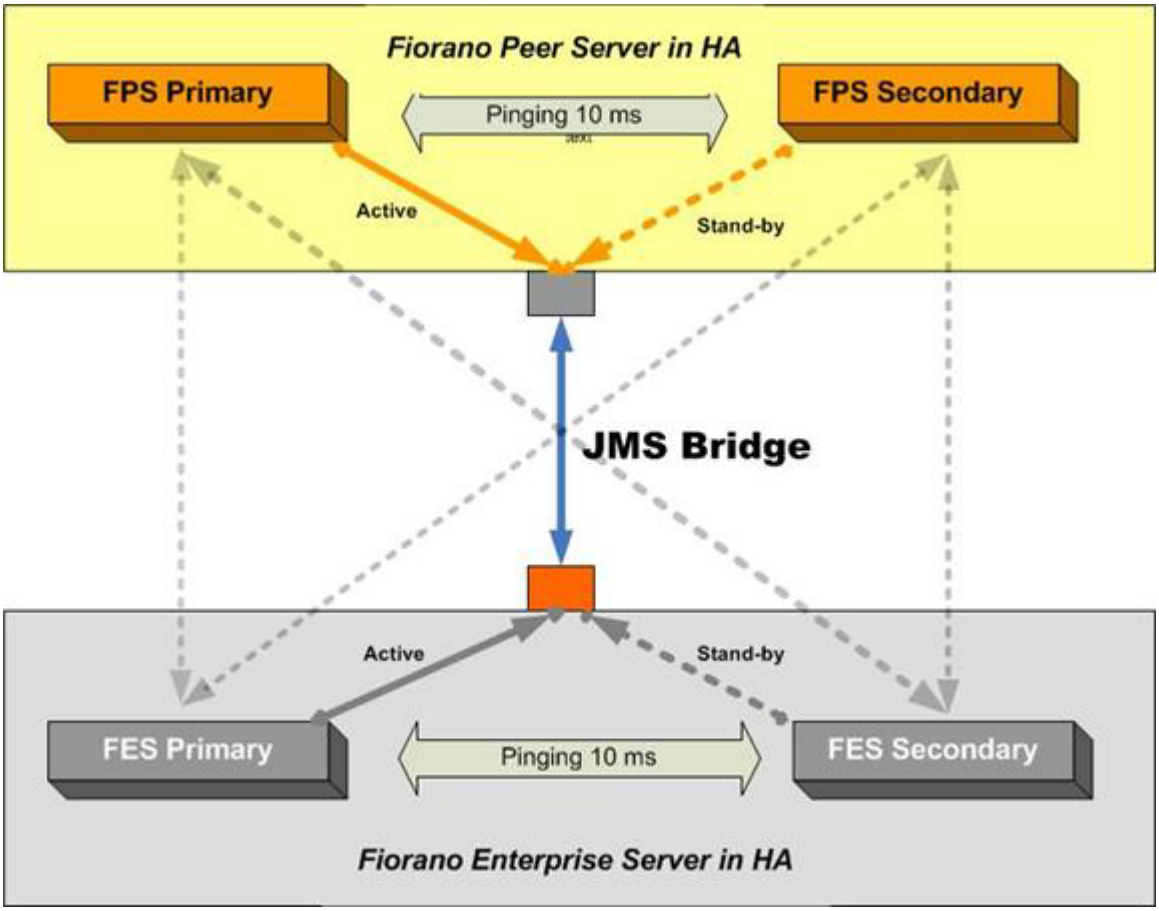


Figure 2: Configuration of Failover on the FPS

6.3 Fiorano Replicated High Availability Working

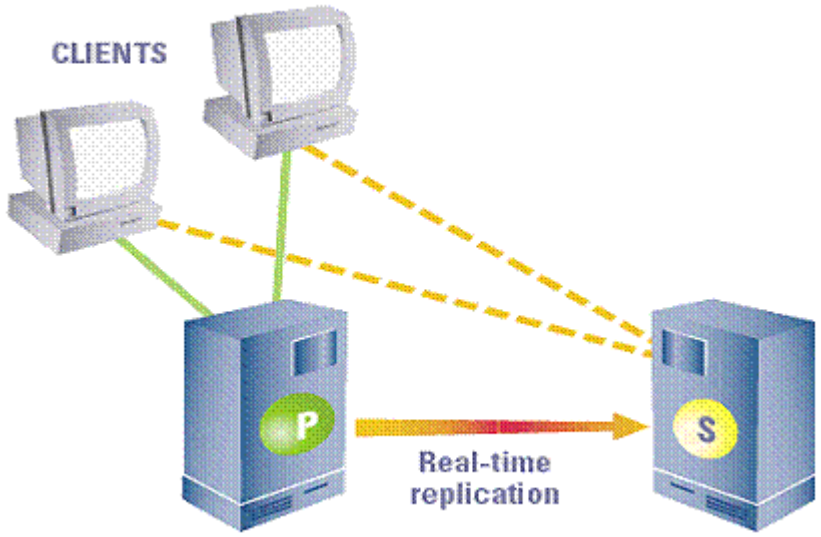


Figure 3: Replication of servers

The central concept of backchannel replication is that the Active Server (the server which is in the Active State) replicates its data store and state to the Passive Server, thus keeping both servers in sync. This replication channel is supported on a private network dedicated to the synchronization of the broker state and messaging data.

The secondary server accepts no client connection while in its hot-standby (passive) role, but is prepared to immediately transition to the Active role as soon as it detects that the Active Server is unavailable. If the primary fails, all Fiorano applications fail over from the primary and reconnect to the designated secondary backup broker.

The primary and secondary broker-pair use the replication channel to routinely seek the heartbeat of the other and watch for any interruption in the data flow or connection to switch states. A locking mechanism (explained below) is also employed to determine the state of the servers.

This Hot-failover process is immediate and is completely transparent to all client applications. The Secondary Server in the active role is sensitive to re-establishment of the replication channel. This reconnection may come from a recovery of the Primary Server or from a replacement Primary Server. Once the primary comes up again, it assumes the role of the Secondary Server (since the original Secondary Server is now the Primary Server).

Important: If servers are running in Replicated HA mode, it becomes essential to have a reliable network for database synchronization calls.

6.3.1 HA Locking Mechanism

HA locking mechanism is employed by the servers in replicated mode to determine the server state in case, a server of the pair is unavailable or if the network fails. A read and write permissions file is shared on a machine this file is referred as the LockFile. The machine hosting the LockFile is referred to as the gateway machine. A server can switch to Active only if it holds a lock over the LockFile.

In HA implementation prior to the locking mechanism, a network link failure between the servers could have led to both servers switching to Standalone state. Since the lock can be held by only one server at a time, it prevents both servers from switching to Active/Standalone state.

The locking mechanism makes the state switching of a HA server more deterministic.

6.3.2 Server States

A server at any point of time can be in the following states:

- Active
- Passive
- Standalone - Same as active. Indicates that the backup server is down/not present in network.
- Dead - Indicates that the server is down/not present in network.
- Active - Sync/Active Transition
- Passive - Sync/Passive Transition
- Waiting - Same as passive. Indicates that the databases of the Active and Passive Servers need to be synchronized.

The following diagram explains the transition to various states:

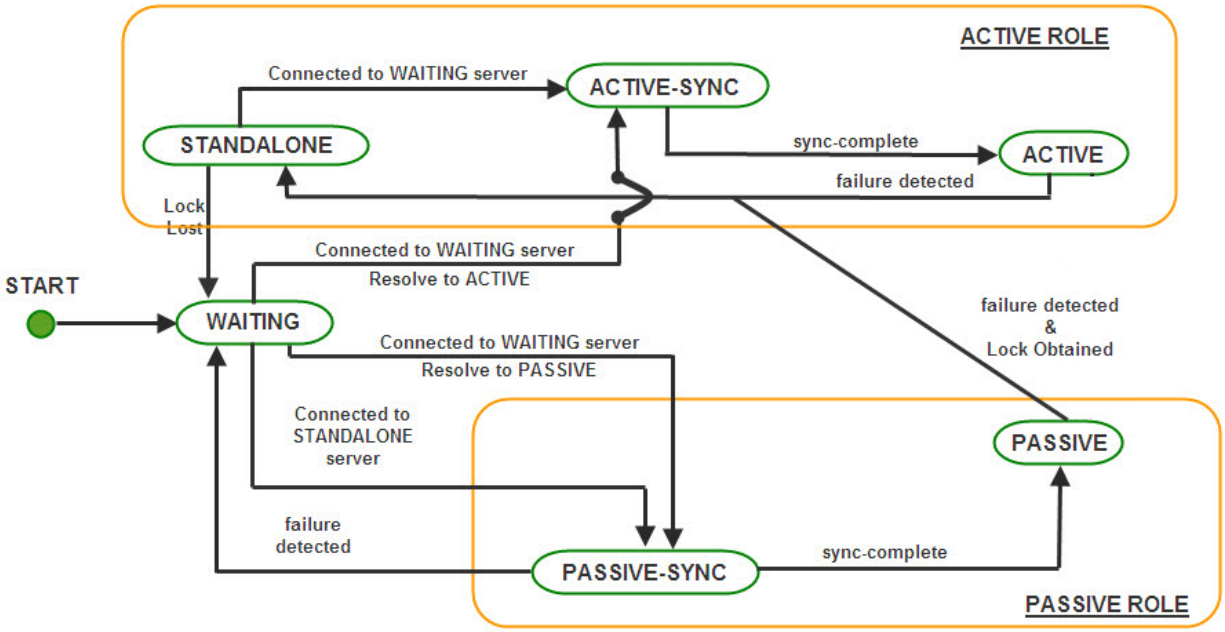


Figure 4: Transition to various states

Note:

- failure detected – refers to the link between the servers being broken
- sync-complete – database synchronization complete
- Lock Lost – lock over the LockFile is lost
- Lock Obtained – lock obtained over the LockFile
- Resolve to Active/Passive – based on which server obtains the lock

1. On startup, the Server enters into WAITING state. In this state, the server is waiting for its backup server to connect to it. This is the initial synchronization state, which is required to sync up the primary server with the secondary to avoid any message loss. This server will change state if one of the following occurs.
   - Switch to PASSIVE SYNC state: If the HA channel is established and the other server is in STANDALONE state.
   - Switch to PASSIVE(STANDBY) SYNC or ACTIVE SYNC state: If the HA channel is established and the other server is also in WAITING state, then the servers assumes themselves as being in Active or Passive roles depending on the Repository Timestamps (whichever server has the latest timestamp is assumed to be the primary).

     If the repository does not have Timestamps (the server-store is cleaned up), then the server which is configured as primary turns to ACTIVE SYNC.

2. When the Server is actively serving clients and its backup server is not running or if the HA transport channel is broken and it has the lock over the lockfile, the state of the server is STANDALONE. If the server in STANDALONE state establishes the HA channel and the other server is in WAITING state, then the STANDALONE server shifts to ACTIVE SYNC state. A passive (standby) server can switch to STANDALONE if the other server is not running or if the transport channel is broken and it acquires the lock over the LockFile.
3. When the Server is in ACTIVE SYNC state, the server starts synchronizing its data with the backup server which is in PASSIVE SYNC. The Server in ACTIVE SYNC continues to serve its clients. Completion of the Runtime Synchronization Protocol causes a transition of the backup server to the ACTIVE state and the server in PASSIVE SYNC state moves to PASSIVE state.
4. Once the Primary Server completes the synchronization, it enters into the ACTIVE state and begins actively transmitting state information and Call Replications onto the PASSIVE Server. At this point, if there is a failure of the ACTIVE server, the Hot Standby PASSIVE Server is ready to move into the STANDLONE state and starts accepting requests from the clients.
5. An active server can switch to WAITING if the transport channel is broken and it loses the lock over the LockFile. A STANDALONE server can switch to WAITING if it loses the lock over the lock file.
6. Whenever there is a change in the server state, it broadcasts the present and previous state to the Backup Server. The Servers transition is a function of its own state, the present and previous state of the Backup Server and whether or not it holds the lock over the LockFile.

6.3.3 Configuring Fiorano SOA High Availability Servers

The FioranoSOA installer comes with prebuilt profiles for replicated mode which are preconfigured and ready to run on a single machine.

Table 1: The default profiles to be used for Primary and Secondary Servers

Server	**Location**
Fiorano Enterprise Server HA Primary	$Fiorano_home/esb/server/profiles/haprofile1/primary/FES
Fiorano Enterprise Server HA Secondary	$Fiorano_home/esb/server/profiles/haprofile1/secondary/FES
Fiorano Peer Server HA Primary	$Fiorano_home/esb/server/profiles/haprofile1/primary/FPS
Fiorano Peer Server HA Secondary	$Fiorano_home/esb/server/profiles/haprofile1/secondary/FPS
Fiorano Peer Server1 HA Primary	$Fiorano_home/esb/server/profiles/haprofile2/ primary/FPS
Fiorano Peer Server1 HA Secondary	$Fiorano_home/esb/server/profiles/haprofile2/secondary/FPS

To launch the server on one of these profiles, the user can use the script server.bat/sh in $Fiorano_home/esb/server/bin.

Examples:

- server.bat -mode fes -profile haprofile1/primary - to launch the primary server of Fiorano Enterprise HA-profile1
- server.bat -mode fps -profile haprofile1/secondary - to launch the secondary server of Fiorano Peer HA -profile1
- server.bat -mode fps -profile haprofile2/primary - to launch the primary server of Fiorano Peer HA -profile2

6.3.4 Configuration Steps

- Setting up the lockfile
- Configuring the profile

6.3.4.1 Setting up the LockFile

A file is created and the directory containing it is shared with read/write permissions.

The lock file, if present on a machine having UNIX/Solaris operating system should be shared by using the NFS protocol - version 4. If the lock file is present on Windows it should be shared using the Samba Protocol. (The lock file can be shared on Windows using NFS – version 4, if it supports it.)

The table below gives some of the possible combinations of operating systems in the HA Setup:

OS hosting the Lock File & Protocol Used	**OS hosting the servers**
Windows - Samba	Windows / Linux
Linux - NFSv4	Linux / Solaris 8,9,10
Solaris - NFSv4	Linux / Solaris 8,9,10

Note:

The user has to make sure that, the operating system hosting the server supports the protocol used for sharing the lock file. The LockFile and the directory containing the LockFile should have read/write permissions set. On Operating Systems other than Windows, one can verify the permissions using the ls -l command.

On Windows Operating System, the directory on the gateway machine containing the LockFile should be mapped to a network drive.

Example:

If the directory containing the lock file on Windows, is shared using the samba protocol, this directory should be mapped to a network drive on the Windows machine hosting the server. Let us say the shared directory is mapped to a drive letter 'Z:/' and the lock file is lock.lck , the lock file path now becomes 'Z:/lock.lck'.

This path should be used to set the LockFile parameter while configuring the profile via Fiorano Studio.

On non Windows Operating System, the directory on the gateway machine containing the LockFile should be mounted on the machine hosting the server.

Examples:

If the lock file is present in a windows samba share & is mounted at /home/user/db on the machine hosting the server then the lock file path on the machine hosting the server would be /home/user/db/lock.lck. This absolute path should be used to set the LockFile parameter while configuring the profile via Fiorano Studio.

Example Mount Command:

```
'mount -t cifs –o rw //<gatewayIP>/<sharename>  <path on local machine where the
directory has to be mounted.>'
```

If the lock file 'lock.lck' is shared using NFS v4 and the directory db is mounted at /home/user/db on the machine hosting the server then the lock file path on the machine hosting the server would be /home/user/db/lock.lck. This absolute path should be used to set the LockFile parameter while configuring the profile via Fiorano Studio.

Example Mount Command:

```
'mount -t nfs4 -o rw <gatewayIP>:/  <path on local machine where the directory has to
be mounted.>'
```

Note: The newly added mount point will disappear after system reboot. The user has to make sure that the mount is automated on system reboot. One can refer to the URL 'http://www.brennan.id.au/19-Network_File_System.html#nfs4' on how to share a directory using NFS v4.

6.3.4.2 Configuring the FES HA Profile

Fiorano SOA gives the ability to configure the HA through Fiorano Studio to simplify your configuration in offline mode.

To configure FES HA, perform the following steps:

1. Open the HA profile (replicated)
2. Right-click the profile and select FES Replicated HA from pop-up menu.

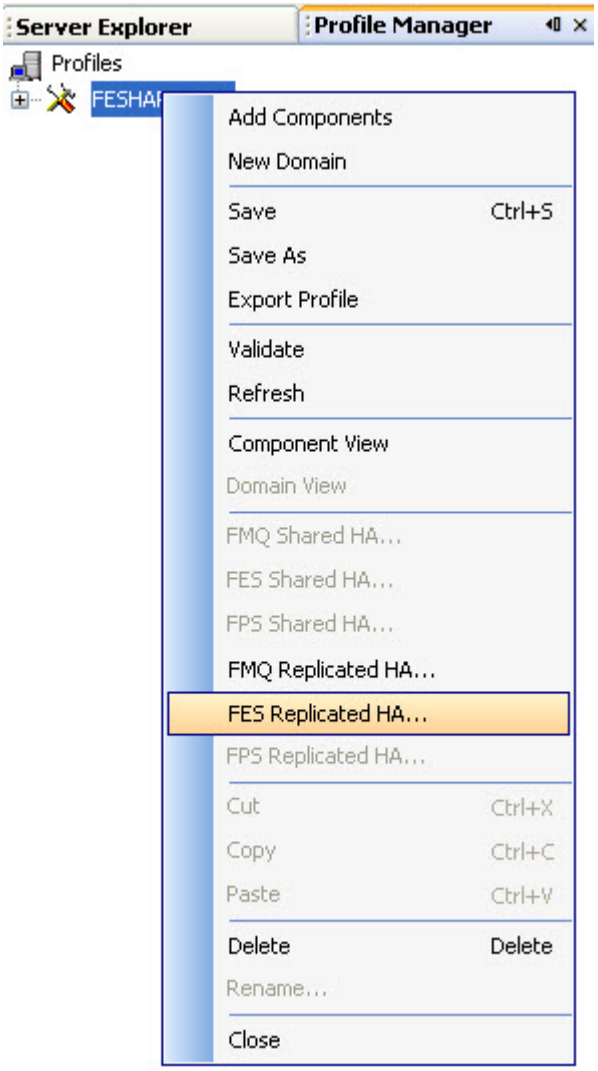


Figure 5: Selection of FES Replicated HA

The FES Replicated HA… dialog box appears as shown in Figure 6.

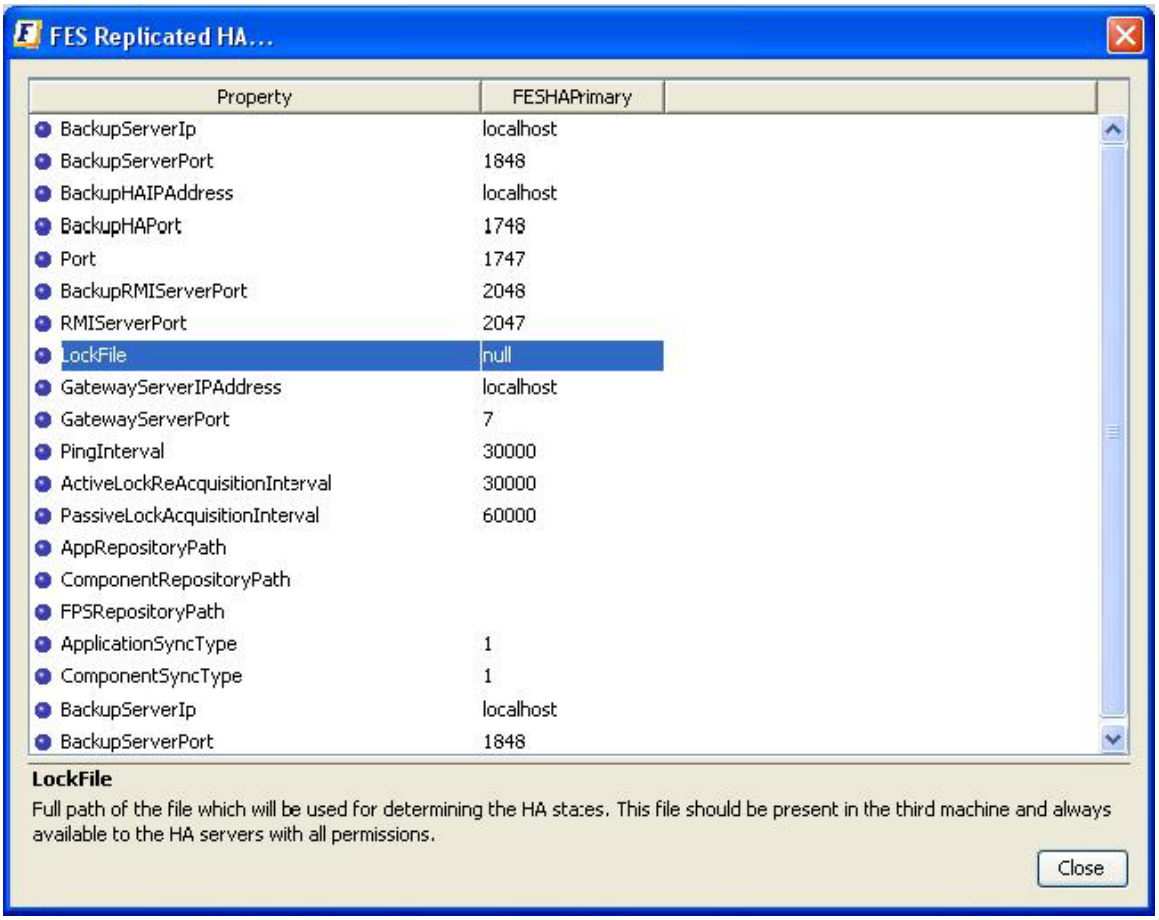


Figure 6: FES Replicated HA... dialog box

Description of Properties (refer to Figure 6)

- **BackupServerIp** - Specifies the Backup Server IP. This IP is configured in backup URL for default connection factories.
- BackUpServerPort - Specifies the Backup Server port. This port is configured in backup URL for default connection factories.

  This can be changed by modifying the Backup Server profile.

  - Open the Backup Server profile in the Fiorano Studio Profile Manager.
  - Navigate to <ProfileName>=>Fiorano=>socketAcceptors=>port-1=>ConnectionManager =>Properties of ConnectionManager and change the port parameter.

The Figure 7 illustrates the configuration of the Fiorano ConnectionManager.

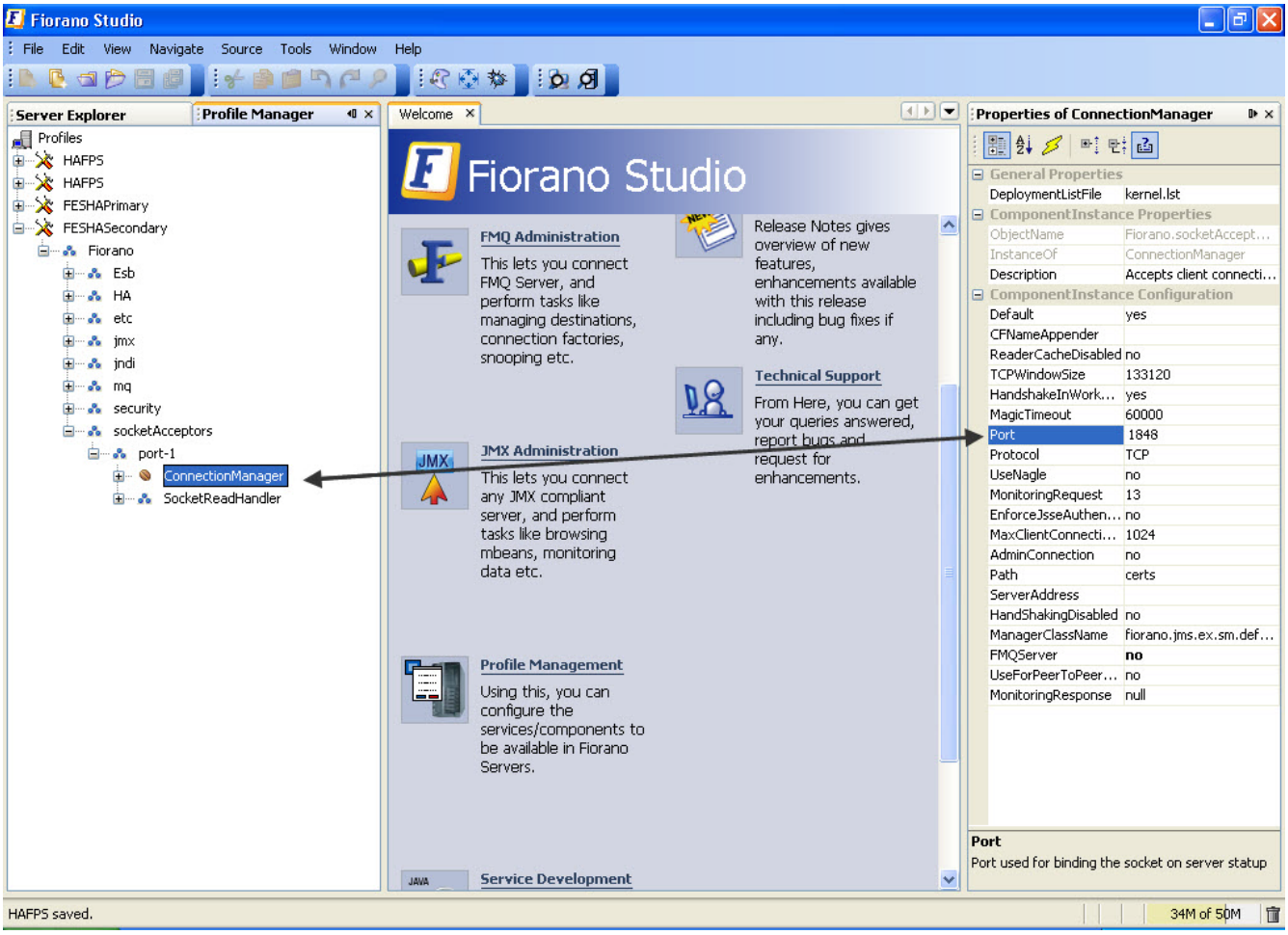


Figure 7: Configuration of the Fiorano ConnectionManager

- **BackupHAIPAddress** - IP Address of backup server in HA mode. This parameter is mandatory to run HA.
- **BackupHAPort** - Port of the Backup server on which peer is listening for status requests send by another server.[This parameter is same as the 'Port' parameter but it specifies the port used by the backup server]
- **Port** - This is the port on which the HA Manager is going to listen for connections from its backup server. Once the connection is established, it starts serving as the back channel for broadcasting the state of the servers to the backup server whenever there is a state transition.
- **BackupRMIServerPort** - Port used by backup server to bind the Mx4J RMI Connector.
- **RMIServerPort** – Port used by the server to bind the Mx4j Rmi Connector.
- **LockFile** - Full path of the file which will be used for determining the HA states. This file should be present in the third machine and always available to the HA servers with all permissions. This parameter is mandatory to run HA. [Also, See 'GatewayServerIPAddress' parameter while configuring.]

- **GatewayServerIPAddress** -IP address of Gateway machine. This is used to detect network failures. It is recommended that the IP specified should be of a machine that is always available on the network. It is mandatory to specify this parameter. This parameter should always be the IP Address of the machine hosting the LockFile.
- **GatewayServerPort** - In Replication HA mode, network failure is detected by using the gateway server machine. Specifies the port on which gateWay machine is listening for incoming requests
- **PingInterval** - Time interval (in ms) after which the remote server is pinged in Replication HA mode.
- **ActiveLockReAcquistionInterval** - This parameter indicates the wait Interval between each attempt to acquire the lock for active server. This value should be in multiples of pingInterval/2 otherwise server may try to acquire the lock on the next multiple of pingInterval/2.
- **PassiveLockAcquistionInterval** - This parameter indicates the wait Interval for the passive server to acquire the lock when the link between active and passive server is down. This value should be greater than '2*ActiveLockReacquisitionInterval', otherwise an exception will be thrown. Server won't start if this value is not set properly.
- **AppRepositoryPath** - Path of the enterprise server application repository. FES stores event process information in this directory. Bydefault its value is $FIORANO_SOA_INSTALL_DIR/server/repository/applications.
- **Component RepositoryPath** - Specifies the Component Repository Path. FES stores services' information in this directory. By default its value is $FIORANO_SOA_INSTALL_DIR/server/repository/components
- **FPSRepositoryPath** - Path of the Fiorano Peer server Configurations repository. FES stores peer server configurations in this directory.By default its value is $FIORANO_SOA_INSTALL_DIR/runtimedata/EnterpriseServers/<ProfileName>/FES/peers.
- **ApplicationSyncType** - Sync Type for Application Repository 0 - FULL SYNC - The Active Application Repository is replaced on the Passive Application Repository 1 - PARTIAL_SYNC - Only the new Applications in the Active Application Repository is updated on the Passive Application Repository 2 - NO_SYNC - No Synchronization will happen between the Active and Passive Repository. Assumes that there is no Application Edited/Removed/Added.
- **ComponentSyncType** - Sync Type for Component Repository 0 - FULL SYNC - The Active Component Repository is replaced on the Passive Component Repository 1 - PARTIAL_SYNC - Only the new Components in the Active Component Repository is updated on the Passive Component Repository 2 - NO_SYNC - No Synchronization will happen between the Active and Passive Repository. Assumes that there is no Component Edited/Removed/Added.The user can configure both Primary and Secondary FES from a single screen.

3. You can also configure both Primary and Secondary FES from a single screen. For this first open both the profiles and select both of them and from popup select FES Replicated HA... as shown in the Figure below:

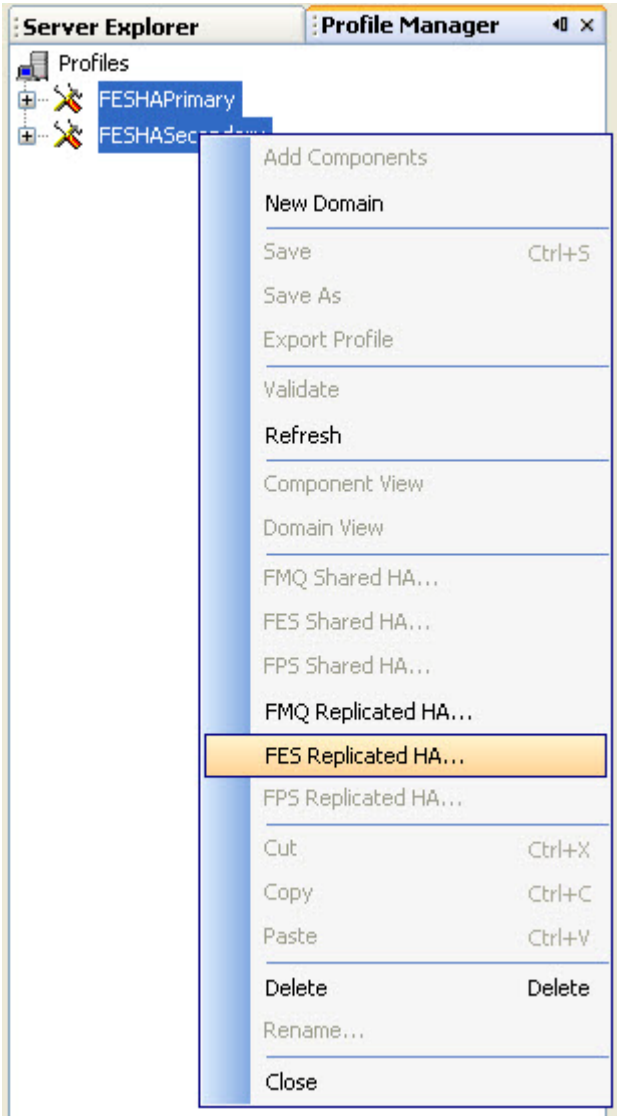


Figure 8: FES Replicated HA... option

The dialog opened contains column for both FES HA Primary and FES HA Secondary as shown in Figure 9

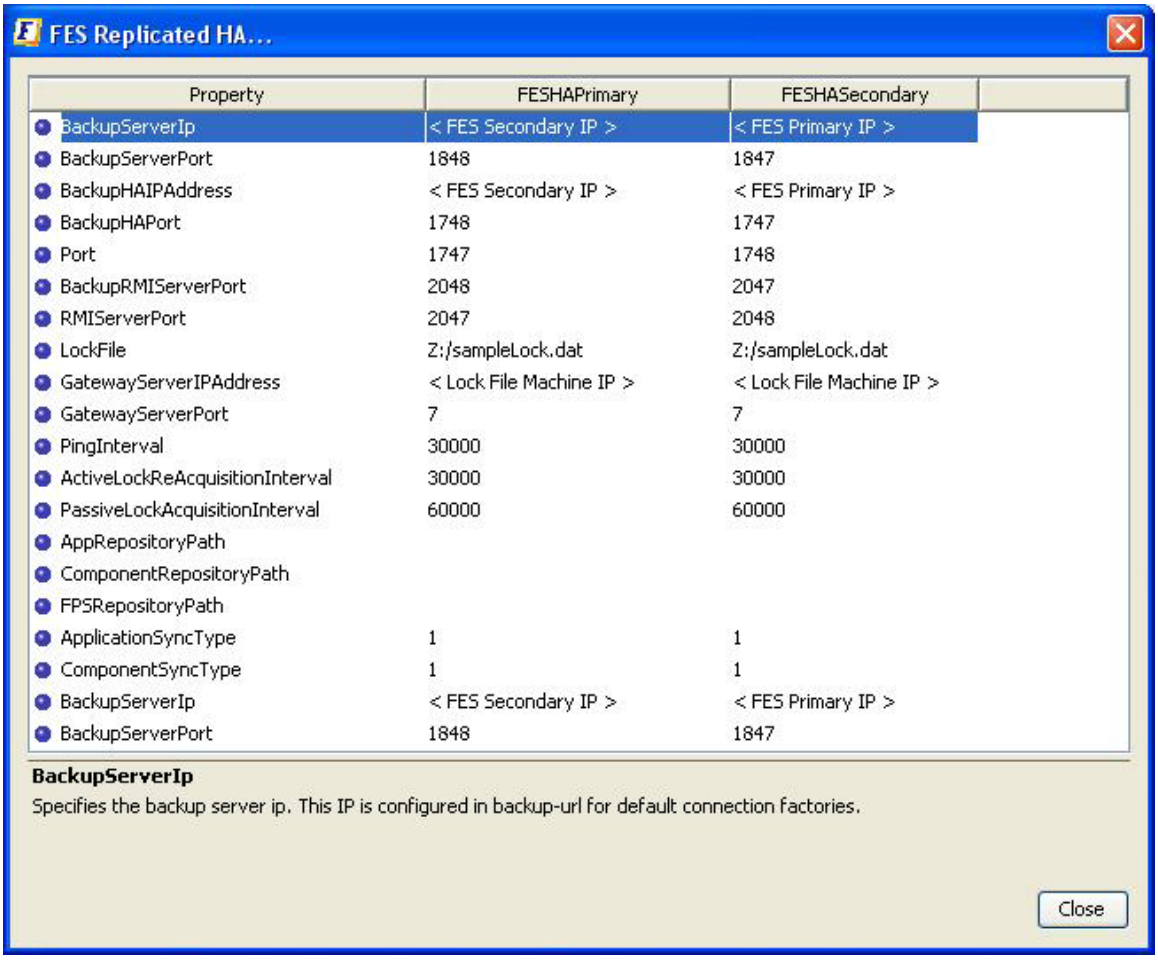


Figure 9: FES Replicated HA.. dialog box

4. Save the profile and we are ready to start the server.

To configure FPS HA, perform the following steps:

1. Open the HA profile (replicated)
2. Right-click the profile and select **FPS Replicated HA..** from the pop-up menu. The **FPS Replicated HA..** dialog appears as shown in figure 10.

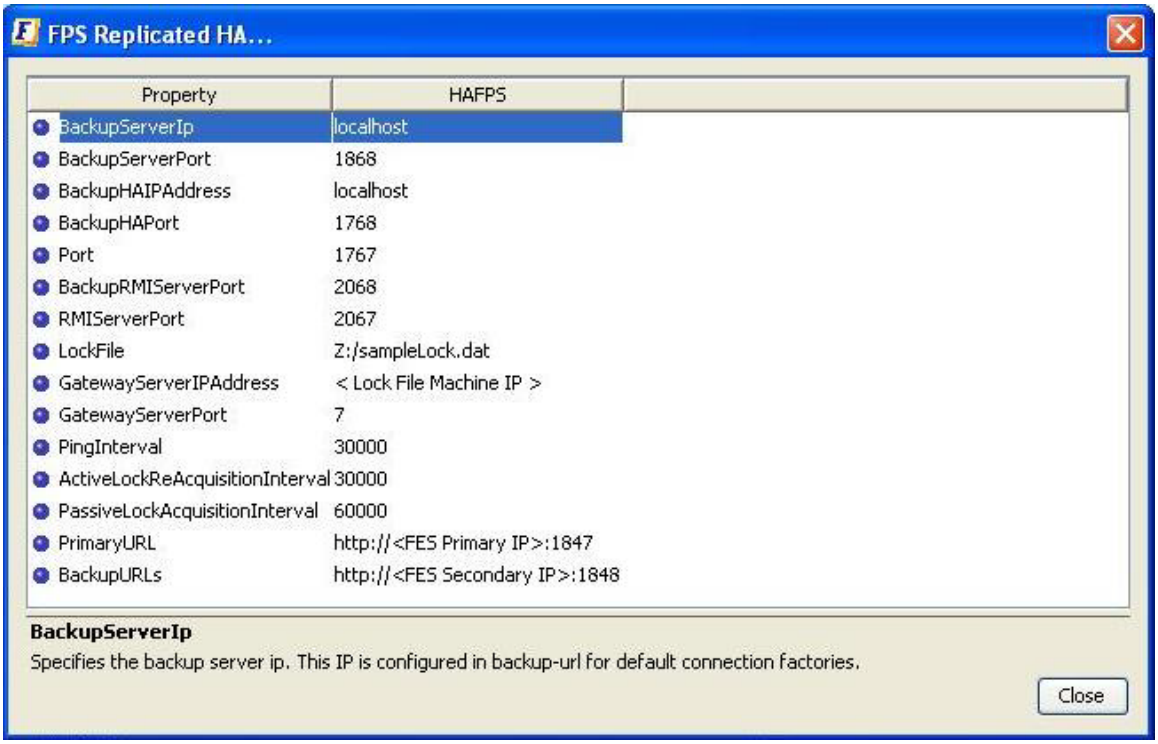


Figure 10: FPS Replicated HA dialog box

Description of Properties (refer to Figure 10)

- **BackupServerIp** - Specifies the backup server ip. This IP is configured in backup-url for default connection factories.
- **BackUpServerPort** - Specifies the backup server port. This port is configured in backup-url for default connection factories.This can be changed by modifying the backup server profile.

  Open the Backup Server profile in the Fiorano Studio Profile Manager. Navigate to <ProfileName>=>Fiorano=>socketAcceptors=>port-1=>ConnectionManager =>Properties of ConnectionManager and change the port parameter.

- **BackupHAIPAddress** - IP Address of backup peer server in HA .
- **BackupHAPort** - Port of the Backup Peer server on which peer is listening for status requests send by another server. This parameter is same as the 'Port' parameter but it specifies the port used by the backup server
- **Port** - This is the port on which the HA Manager is going to listen for connections from its backup server. Once the connection is established, it starts serving as the back channel for broadcasting the state of the servers to the backup server whenever there is a state transition.
- **BackupRMIServerPort** - Port used by backup server to bind the Mx4J RMI Connector.
- **RMIServerPort** – Port used by the server to bind the Mx4j Rmi Connector.
- **LockFile** - Full path of the file which will be used for determining the HA states. This file should be present in the third machine and always available to the HA servers with all permissions. This parameter is mandatory to run HA. [Also, See **GatewayServerIPAddress** parameter while configuring.]
- **GatewayServerIPAddress** - IP address of Gateway machine. This is used to detect network failures. It is recommended that the IP specified should be of a machine that is always available on the network. It is mandatory to specify this parameter. This parameter should always be the IP Address of the machine hosting the LockFile.
- **GatewayServerPort** - In Replication HA mode, network failure is detected by using the gateway server machine. Specifies the port on which gateWay machine is listening for incoming requests
- **ActiveLockReAcquistionInterval** - This parameter indicates the wait Interval between each attempt to acquire the lock for active server. This value should be in multiples of pingInterval/2 otherwise server may try to acquire the lock on the next multiple of pingInterval/2.
- **PassiveLockAcquistionInterval** - This parameter indicates the wait Interval for the passive server to acquire the lock when the link between active and passive server is down. This value should be greater than '2*ActiveLockReacquisitionInterval', otherwise an exception will be thrown. Server won't start if this value is not set properly.
- **PingInterval** - Time interval (in ms) after which the remote server is pinged in Replication HA mode.
- **PrimaryURL** – The primary URL of MQ server [i.e the FES] from which configuration should be loaded
- **BackupURL(s)** – The backup URL(s) of MQ server [i.e the FES] from which configuration should be loaded.

Note: You can also configure both Primary and Secondary FPS from a single screen .This is similar to the procedure adopted while configuring HA FES. Save the profile and we are ready to start the server.

3. Save the profile and we are ready to start the server

6.3.5 Verifying HA Setup

On starting the Fiorano SOA Server that is part of an HA pair, the server prints debug information about its own state (ACTIVE, PASSIVE, and WAITING). It also prints information about its backup server's state whenever it detects a change.

Example Statements on console:

[Sat Feb 07 14:03:50 IST 2009] Old status of remote server = DEAD

[Sat Feb 07 14:03:50 IST 2009] New status of remote server = WAITING

[Sat Feb 07 14:03:50 IST 2009] Old status of remote server = DEAD

[Sat Feb 07 14:03:50 IST 2009] New status of remote server = WAITING

[Sat Feb 07 14:03:50 IST 2009] Old status of remote server = WAITING

[Sat Feb 07 14:03:50 IST 2009] New status of remote server = PASSIVE TRANSITION STATE

The Console includes statements as shown below:

'Primary Server switched to ACTIVE' or 'Secondary Server switched to PASSIVE', which indicate that the pair has successfully connected. Also, a statement gets printed when the lock is successfully acquired over the lockfile on the active servers console.

Example: Successfully acquired lock on:Z:\lock.txt.

The figure below illustrates a successfully started Fiorano HA Peer Server:

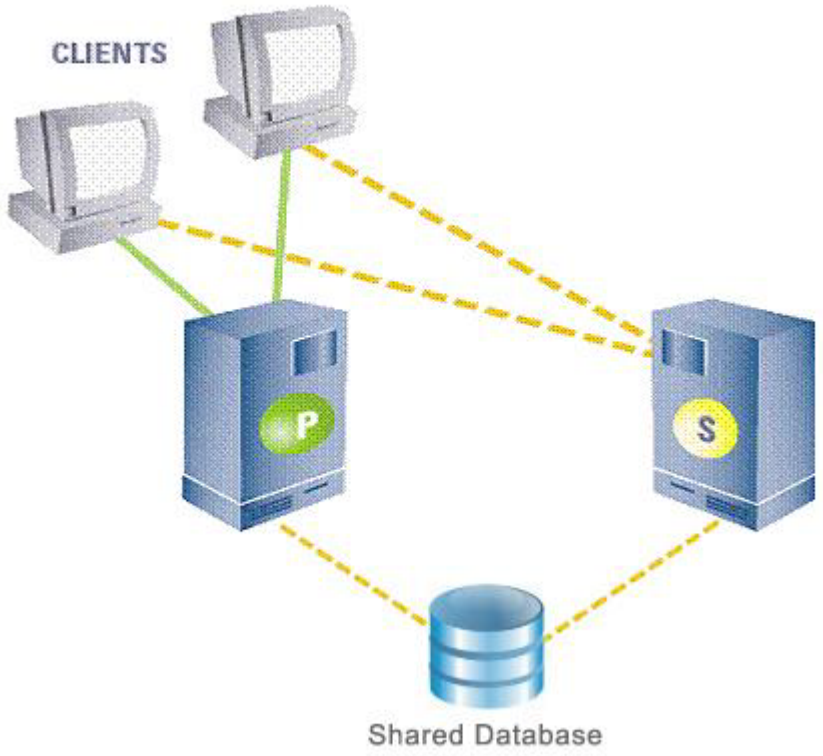
```
C:\WINDOWS\system32\cmd.exe - server -profile haprofile1/primary
C:\PROGRA~1\Fiorano\FMP90~2.0-B\esb\server\bin>server -profile haprofile1/primary

Fiorano Home      : C:\PROGRA~1\Fiorano\FMP90~2.0-B
Java Home         : C:\PROGRA~1\Fiorano\FMP90~2.0-B\jre1.5.0_16
Profile           : C:\PROGRA~1\Fiorano\FMP90~2.0-B\esb\server\profiles\haprofile1\primary

Server's Process ID : 792
[07/Feb/2009 14:03:32]    license      INFO       The fiorano-soa9.lic license for the product FPS
 false, Node locked = false
RMIConnectorServer started at: service:jmx:rmi://localhost/jndi/fmq
RmiConnectorServer Listening Port: 2067
[Sat Feb 07 14:03:48 IST 2009]  Trying to establish connection with localhost:1768 ...
[Sat Feb 07 14:03:48 IST 2009]  Primary Server switched to WAITING
Backup Server not Connected. Check configuration or Start backup Server ...
Profile ..\profiles\haprofile1/primary\FPS successfully deployed on Sat Feb 07 14:03:48 IST 2009
[Sat Feb 07 14:03:50 IST 2009]  Old status of remote server = DEAD
[Sat Feb 07 14:03:50 IST 2009]  New status of remote server = WAITING
[Sat Feb 07 14:03:50 IST 2009]  Old status of remote server = DEAD
[Sat Feb 07 14:03:50 IST 2009]  New status of remote server = WAITING
[Sat Feb 07 14:03:50 IST 2009]  Old status of remote server = WAITING
[Sat Feb 07 14:03:50 IST 2009]  New status of remote server = PASSIVE TRANSITION STATE
[Sat Feb 07 14:03:50 IST 2009]  Old status of remote server = WAITING
[Sat Feb 07 14:03:50 IST 2009]  New status of remote server = PASSIVE TRANSITION STATE
[Sat Feb 07 14:03:51 IST 2009]  Primary Server switched to ACTIVE TRANSITION STATE
[Sat Feb 07 14:03:51 IST 2009]  Old status of remote server = PASSIVE TRANSITION STATE
[Sat Feb 07 14:03:51 IST 2009]  New status of remote server = PASSIVE
[Sat Feb 07 14:03:51 IST 2009]  Old status of remote server = PASSIVE TRANSITION STATE
[Sat Feb 07 14:03:51 IST 2009]  New status of remote server = PASSIVE
Successfully acquired lock on ::C:\lock.txt
Connected to [URL: http://localhost:1847]
Registering with enterprise server
Waiting for the acceptance from Enterprise Server
Waiting for the acceptance from Enterprise Server
Waiting for the acceptance from Enterprise Server
Updating hafps's Configuration in FES Server Repository ...
[Sat Feb 07 14:04:27 IST 2009]  Primary Server switched to ACTIVE
Max Memory Allocated to the Server : 508 MB.
Fiorano Server accepting internal connections at http://192.168.1.133:1867 .
Connected to [URL: http://localhost:1847]
Fiorano SOA 9.0.0, Build # 9009
Server Name ::  hafps
Successfully started Fiorano Peer Server at Sat Feb 07 14:04:47 IST 2009
```

Figure 11: Fiorano HA Peer Server

6.3.6 Shutting down the HA Server

For details on how to shutdown the servers, please refer to sections 2.3.3.2 and 2.4.3.2.

6.4 Fiorano High Availability Working In Shared Mode



In this mode of High availability, the primary-secondary broker pair shares a common database and do not replicate data over the network. If the primary fails, all Fiorano applications fail over from the primary and reconnect to the designated secondary backup broker. The primary and secondary broker-pair use the network channel between them to routinely seek the heartbeat of the other and watch for any break in connection to switch states. A locking mechanism (as explained in section 6.3.1) is also employed to determine the state of the servers. The database which is common to both the servers is referred to as the **shared database**.

6.4.1 Shared HA Precondition

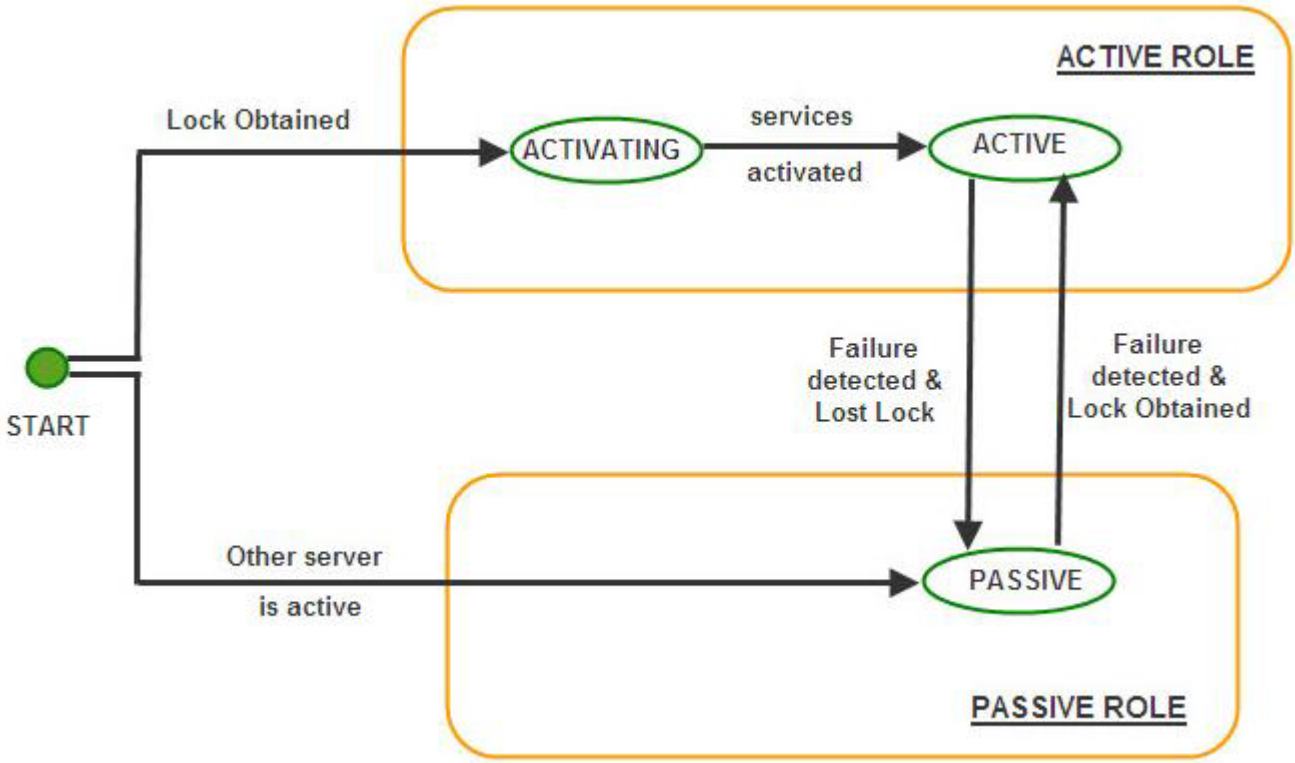
The shared database is critical for the servers to function, as the servers store all data in it. It is mandatory for the **Shared Database** to be always accessible to the servers. Unavailability of the shared database could lead to data loss and data corruption.

6.4.2 Server States

A server at any point of time can be in the following states:

- Active
- Passive
- Activating – The server switches to this state as soon as it acquires the lock. Once all its services are activated, it switches to Active State.

The following diagram explains the transition to various states:



Note:

- failure detected – refers to the link between the servers being broken
- Lock Lost – lock over the LockFile is lost
- Lock Obtained – lock obtained over the LockFile

When the server starts up, the server tries to acquire a lock on the lock file. If it acquires the lock successfully, it switches to the ACTIVATING state. It then switches to ACTIVE state once all its services have been activated. Unlike in replicated HA, where the servers wait for each other to come up (that is, in WAITING state), a server in shared mode does not need to wait for its backup server to come up because they share a common database and no database synchronization is required which is the case for servers working in replicated mode.

After switching to ACTIVE state, the server keeps trying to connect to its backup server. If the backup server starts up, the backup server switches to PASSIVE state.

At this point, if there is a failure of the ACTIVE server, the Hot Standby PASSIVE Server is ready to move into the ACTIVE state and starts accepting requests from the clients.

6.4.3 Configuring Fiorano SOA High Availability Servers

The Fiorano SOA installer comes with a prebuilt profile for shared mode which is preconfigured and ready to run on a single machine.

Table 1: The default profile to be used for Primary and Secondary Servers

Server	Location of Profile
Fiorano Enterprise Server HA Primary	$Fiorano_home/esb/server/profiles /haprofile_shared/primary/FES
Fiorano Enterprise Server HA Secondary	$Fiorano_home/esb/server/profiles /haprofile_shared/secondary/FES
Fiorano Peer Server HA Primary	$Fiorano_home/esb/server/profiles /haprofile_shared/primary/FPS
Fiorano Peer Server HA Secondary	$Fiorano_home/esb/server/profiles /haprofile_shared/secondary/FPS

To launch the server on one of these profiles, the user can use the script server.bat/sh in $Fiorano_home/esb/server/bin.

Since the shared HA pair use a common database, the location of the database has to specified while starting up each server.

The -dbPath command line option is used for specifying the location of the shared database.

Examples:

- to launch the primary server of Fiorano Enterprise shared -HA-profile

  ```
  server.bat -mode fes -profile haprofile_shared/primary -dbPath D:/sharedDb
  ```

- to launch the secondary server of Fiorano Peer shared-HA –profile

  ```
  server.bat -mode fps -profile haprofile_shared/secondary -dbPath E:/sharedDb
  ```

Note: While running the servers on the same machine, both the HA servers should have their databases pointing to the same physical directory.

6.4.4 Configuration Steps

- Setting up the lock file
- Setting up the shared database
- Configuring the profile
- Changing the location of log files.

6.4.4.1 Setting up the Lock File

For more information, refer to section 6.3.4.1 Setting up the LockFile.

6.4.4.2 Setting up the shared database

A directory is shared on a third machine with read/write permissions using the NFS protocol. If the operating system supports NFS - version 4, then we recommend that the shared database be shared using NFS version 4 else it should be shared using NFS –version 3.

Note: The user has to make sure that, the operating system hosting the server supports the protocol used for sharing the shared database.

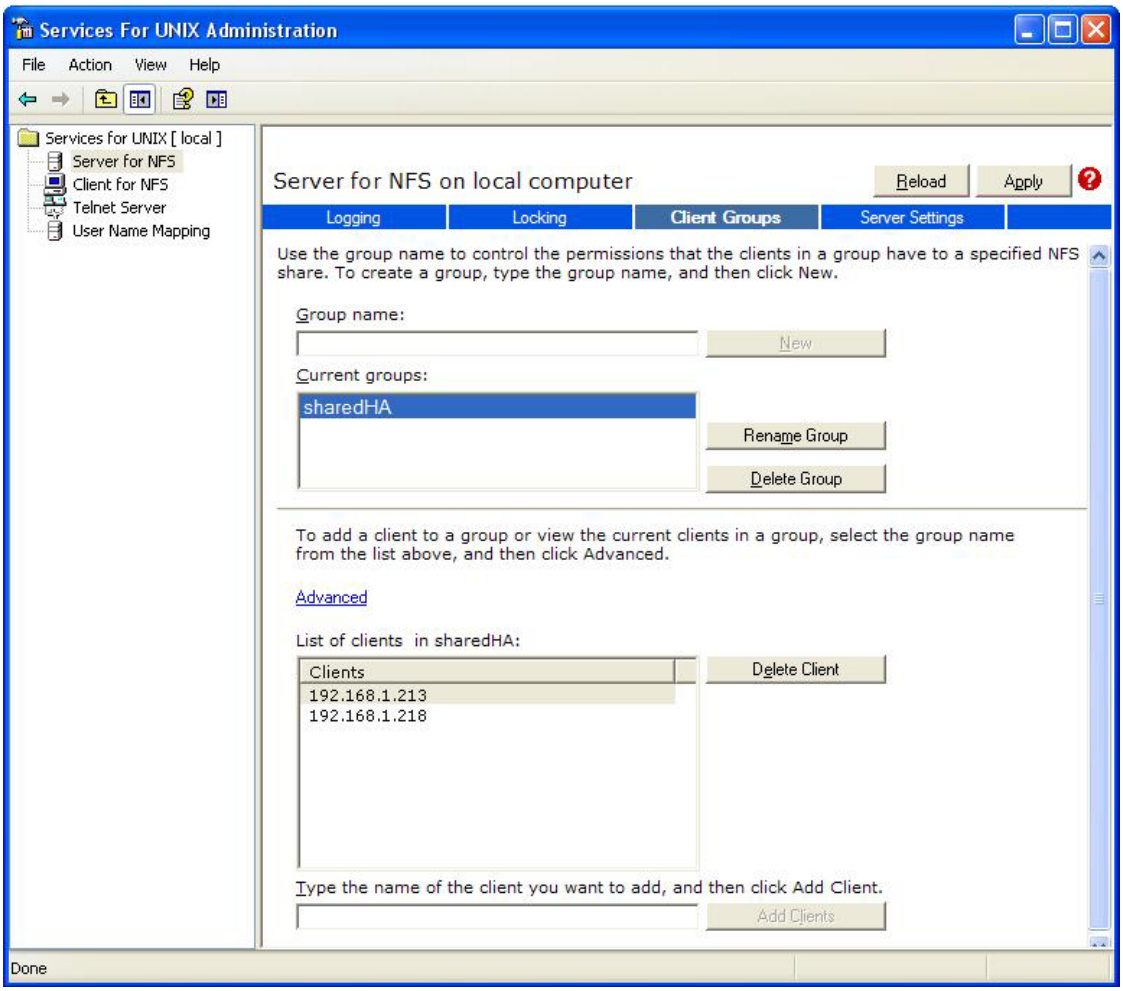
For more information on how to share a directory using NFS v4, refer to http://www.brennan.id.au/19-Network_File_System.html#nfs4.

For sharing a directory on Windows OS using NFS, the user has to download and install **Windows Services for Unix** / **Subsystem for UNIX-based Applications** provided by Microsoft on the machine. The **Client for NFS** and **Server for NFS** packages must be installed. Some Microsoft Operating Systems have these packages by default like **Windows Server 2003 R2** and **Windows Vista Enterprise and Ultimate Editions**.

Following are the steps to share the directory using NFS v3 on Windows OS:

1. Create a Group of hosts that can access the share:

   Open **Services for Unix Administration**. Click on **Server For NFS** node. The **Client Groups** tab appears. Click on it and you get options to add/remove/rename groups. Add a group and add clients (IP Addresses) to the group. Click **Apply**. The screen shot below shows a group **sharedHA** having two hosts in it.



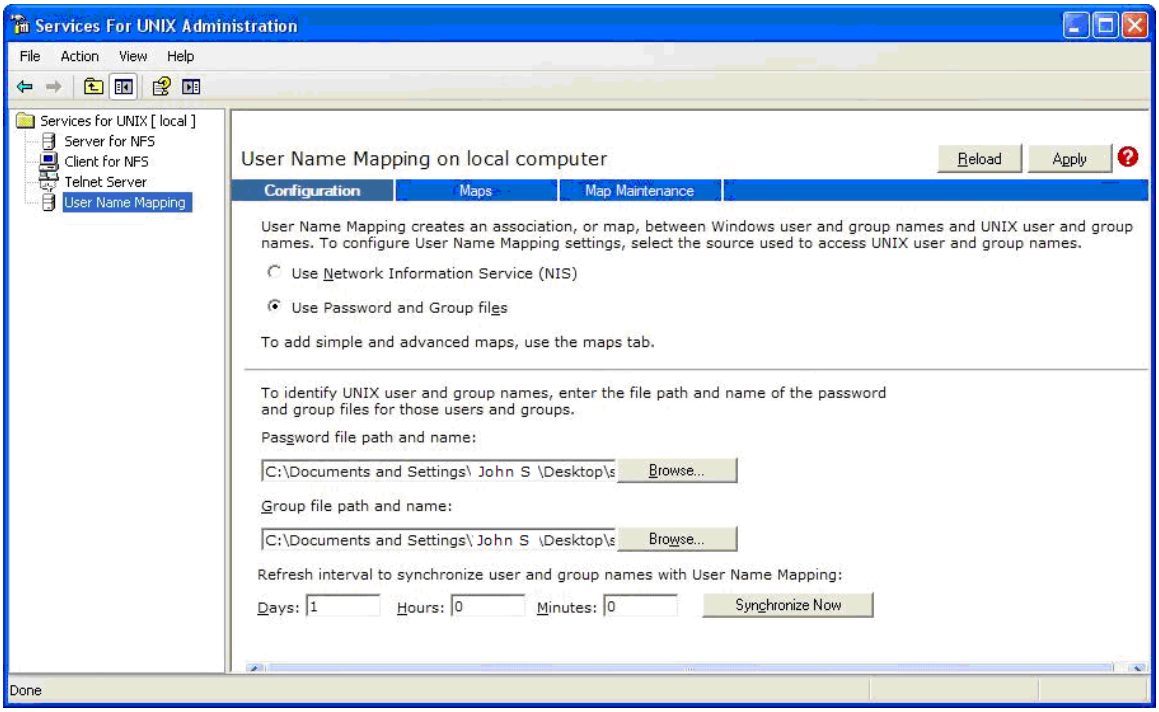
2. Create a user – mapping:

   Let us assume that the name of the windows user to be **John S**. To modify a nfs share, the uid(s) of the user sharing the directory and the uid of the user accessing the share need to be the same. Now, say that the server is running on a UNIX OS, since Windows users have no uid, we need to create a mapping between the Unix user and Windows User, that is, the user **John S** has to be mapped to the Unix user.

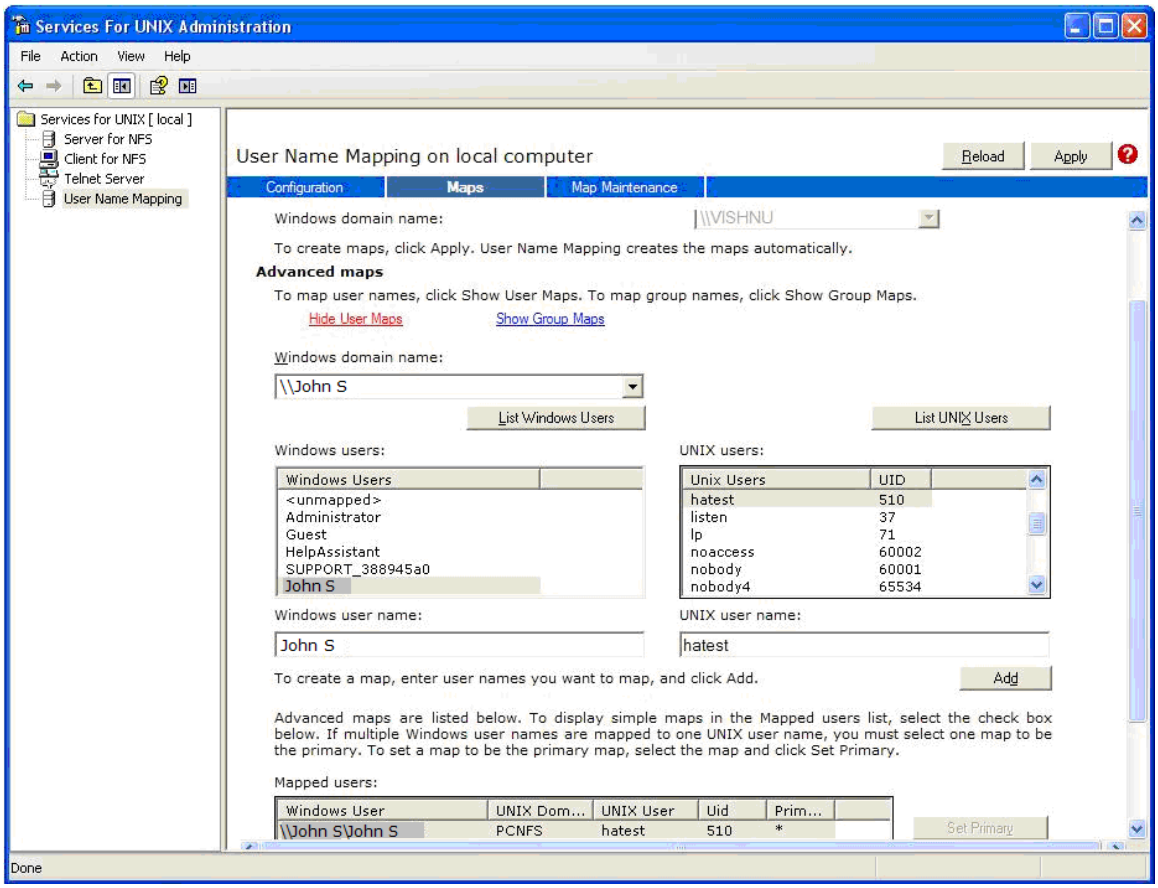
   Following are the steps to create the mapping:

   - Open **Services for Unix Administration**. Click on **User Mapping** node and then click on the **Configuration** node
   - The user can configure the service either using NIS or by using the password/ group files.
   - If the user chooses to create a mapping by using the password and group files, the files **/etc/group** and **/etc/passwd** on the Unix Machine should be copied to the windows machine and their paths should be given in the corresponding boxes and the changes should be applied.

The figure below illustrates the **Configuration** Window.



- Now click on the Maps Node. A user can choose to create simple or advanced maps. In an advanced mapping, the user can list the Windows and UNIX users and create a mapping. The figure below illustrates the **User Mapping** Window having an advanced map between the Windows user **John S** and a UNIX user having uid 510 and name 'hatest'. By doing so, users having uid equal to 510 on machines which belong to the access group of a share will be given read/write access to the share.



3. Share the directory using the 'nfsshare' command:

   Open a command prompt and type the following command

   `nfsshare sharedDB=C:/shared/db -o rw=sharedHA`

   Note:

   sharedDB refers to the share name.

   C: /shared/db refers to the path of the shared directory

   rw=sharedHA specifies that read & write permissions should be given to the group **sharedHA**

   Type the command **nfshare /?** for help on how to use the **nfsshare** command

4. Mounting the shared database on the Machine hosting the server:

   The nfs share can be accessed on UNIX/Solaris OS by mounting it. The **mount** command is used for this purpose. On Windows, the nfs share can be accessed by adding it as a network drive.

   Examples:

- If the shared database is on Windows & the HA server is on UNIX:

  `mount –t nfs –o rw 192.168.1.213:sharedDB /home/testUser/sharedDB`

Note:

The IP Address '192.168.1.213' refers to IP address of the Windows Machine hosting the shared database. 'sharedDB' is the sharename of the directory being shared on the Windows Machine & '/home/testUser/sharedDB' is the path on the local machine where the share will be mounted.

The path on the local machine where the shared database is mounted should given as the value for the '-dbPath' command line option while starting the server.

To start up the FES shared HA Primary profile, we now type the command

```
server.sh –mode fes –profile haprofile_shared/primary –dbPath
/home/testUser/sharedDB
```

- If the shared database is on UNIX & the HA server is on UNIX:

  ```
  mount –t nfs4 –o rw 192.168.1.213:/ /home/testUser/sharedDB
  ```

Note:

The IP Address 192.168.1.213 refers to IP address of the UNIX Machine hosting the shared database which is being shared using NFSv4 and **/home/testUser/sharedDB** is the path on the local machine where the share will be mounted.

The path on the local machine where the shared database is mounted should given as the value for the **-dbPath** command line option while starting the server.

To start up the FPS shared HA Primary profile, we now type the command

```
server.sh –mode fps –profile haprofile_shared/primary –dbPath
/home/testUser/sharedDB
```

- If the shared database is on UNIX/Windows and the HA server is on Windows:

  Suppose the uid of the user sharing the database is 510, a mapping should be created between the windows user & the UNIX user using the **Services for Windows Administration** software. Once, the mapping is created the user should map the shared directory to a network drive. On adding a network drive, a Confirmation dialog box appears as shown in the figure below to display the mapping used for accessing the nfs share.

The added network drive path should given as the value for the **-dbPath** command line option while starting the server.

If the shared database is mapped to a network drive letter say **Z:**, and we want to start up FES shared HA Primary profile, we now type the command

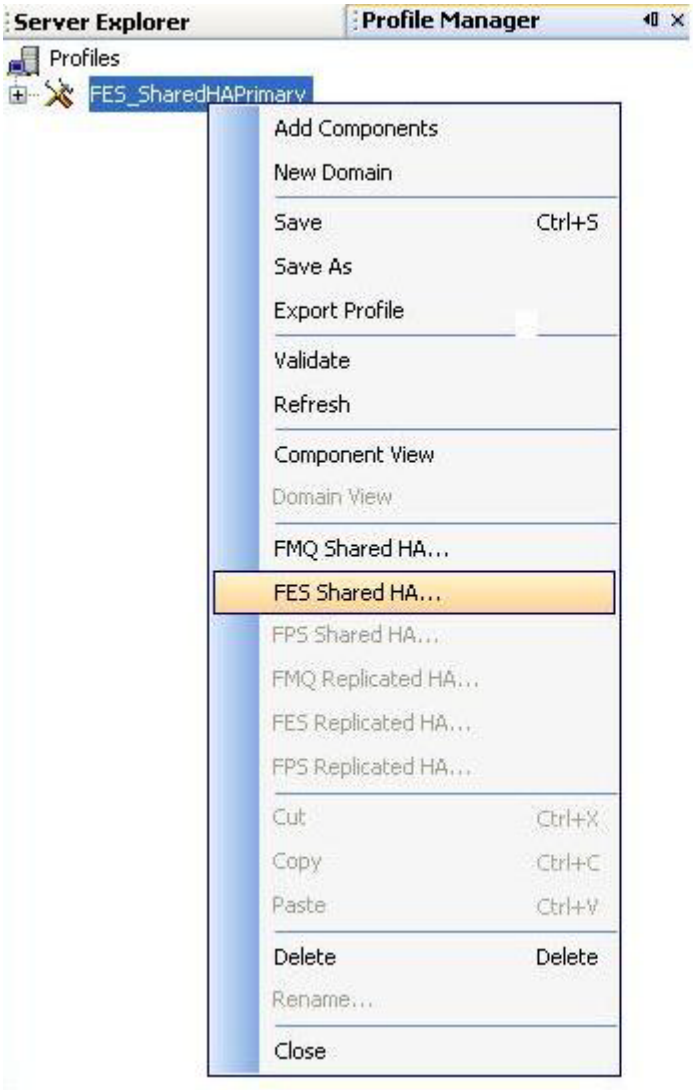
```
server.bat –mode fes –profile haprofile_shared/primary –dbPath Z:/
```

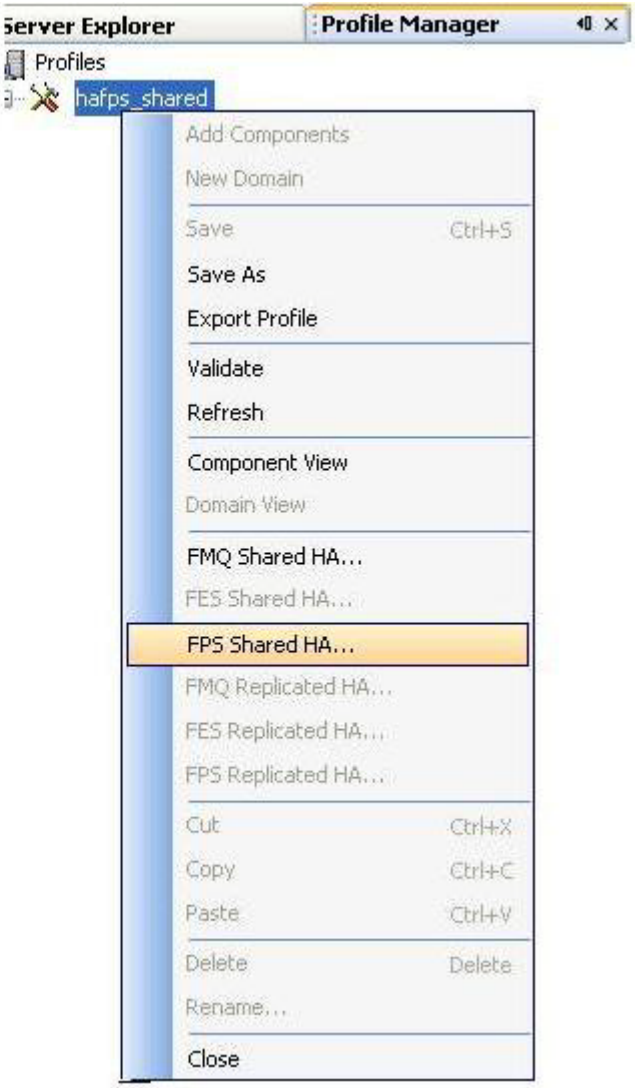
6.4.4.3 Configuring the FES/FPS HA Profile

Fiorano SOA gives the ability to configure the HA through Fiorano Studio to simplify your configuration in offline mode.

To configure FES/FPS HA, perform the following steps:

1. Open the HA profile (shared)
2. Right-click the profile and select FES/FPS Shared HA from pop-up menu.





The FES/FPS Shared HA dialog box appears. You can now configure the various parameters that appear in the dialog box. These parameters are same as the parameters specified for a profile in replicated mode. The user can also configure both profiles in a single dialog. Please refer to section 6.3.4.2 Configuring the FES HA Profile for the description of each parameter and how to configure both profiles in a single dialog box.

3. Save the profile.

6.4.4.4 Changing the location of log files

The log files for the servers running in replicated mode are created in their respective databases. Since, the servers running in shared mode share a common database, the log files by default are also shared by both servers. This configuration has to be changed for the primary & secondary server before startup.

This can be done by modifying the file 'Configs.xml' which is located under the 'conf' directory of the profile location. i.e.
$Fiorano_Home/esb/server/profiles/<profile_name>/<profile_type>/<server_type>/conf

Where

- <profile_name> is the name of the profile.
- <profile_type> is the type of the profile, that is, primary or secondary.
- <server_type> is the type of the server, that is, FES or FPS.

Given below is a screen shot of the file **Configs.xml**. The value of the attribute **FileName** of the child nodes **Appender** in the file (which have been circled in the below screenshot) decides the location of the log file.

```
<Container xmlns="http://fiorano.com/types/common">
    <LOGGER LogLevel="3" ObjectName="Fiorano.Loggers:ServiceType=Logger,Name=Fiorano,type=config">
        <Appender AppenderName="OutAppender" AppenderType="file"
            FileName="mqout.log" IsAppend="true"
            LogPattern="%-5p %d{HH:mm:ss,SSS} %m%n" MaxBackupIndex="4"
            MaxFileSize="1000000" MaxFilterLevel="10" MinFilterLevel="4"
            PrintTarget="System.out" ThresholdLevel="10"/>
        <Appender AppenderName="ErrAppender" AppenderType="file"
            FileName="mqerr.log" IsAppend="true"
            LogPattern="%-5p %d{HH:mm:ss,SSS} %m%n" MaxBackupIndex="4"
            MaxFileSize="1000000" MaxFilterLevel="3" MinFilterLevel="1"
            PrintTarget="System.err" ThresholdLevel="10"/>
    </LOGGER>
    <LOGGER LogLevel="6" ObjectName="Fiorano.Loggers:ServiceType=Logger,Name=Monitoring,type=config">
        <Appender AppenderName="OutAppender" AppenderType="file"
            FileName="monitor.txt" FilterPattern="" IsAppend="true"
            LogPattern="%d{dd/MMM/yyyy HH:mm:ss}%m%n" MaxBackupIndex="4"
            MaxFileSize="100000000" MaxFilterLevel="10"
            MinFilterLevel="4" PrintTarget="System.out" ThresholdLevel="10"/>
        <Appender AppenderName="ErrAppender" AppenderType="file"
            FileName="monitor.txt" FilterPattern="" IsAppend="true"
            LogPattern="%d{dd/MMM/yyyy HH:mm:ss}%m%n" MaxBackupIndex="4"
            MaxFileSize="100000000" MaxFilterLevel="3"
            MinFilterLevel="1" PrintTarget="System.err" ThresholdLevel="10"/>
    </LOGGER>
```

We recommend that the server should have its log files on the same machine as the server. Set the value of the attribute **FileName** to the path of the log file on the local machine.

The screenshot below shows the modified **Configs.xml** where the log files are located at **/home/sharedHA**.

```
<Container xmlns="http://fiorano.com/types/common">
    <LOGGER LogLevel="3" ObjectName="Fiorano.Loggers:ServiceType=Logger,Name=Fiorano,type=config">
        <Appender AppenderName="OutAppender" AppenderType="file"
            FileName="/home/sharedHA/mqout.log" IsAppend="true"
            LogPattern="%-5p %d{HH:mm:ss,SSS} %m%n" MaxBackupIndex="4"
            MaxFileSize="1000000" MaxFilterLevel="10" MinFilterLevel="4"
            PrintTarget="System.out" ThresholdLevel="10"/>
        <Appender AppenderName="ErrAppender" AppenderType="file"
            FileName="/home/sharedHA/mqerr.log" IsAppend="true"
            LogPattern="%-5p %d{HH:mm:ss,SSS} %m%n" MaxBackupIndex="4"
            MaxFileSize="1000000" MaxFilterLevel="3" MinFilterLevel="1"
            PrintTarget="System.err" ThresholdLevel="10"/>
    </LOGGER>
    <LOGGER LogLevel="6" ObjectName="Fiorano.Loggers:ServiceType=Logger,Name=Monitoring,type=config">
        <Appender AppenderName="OutAppender" AppenderType="file"
            FileName="/home/sharedHA/monitor.txt" FilterPattern="" IsAppend="true"
            LogPattern="%d{dd/MMM/yyyy HH:mm:ss}%m%n" MaxBackupIndex="4"
            MaxFileSize="100000000" MaxFilterLevel="10"
            MinFilterLevel="4" PrintTarget="System.out" ThresholdLevel="10"/>
        <Appender AppenderName="ErrAppender" AppenderType="file"
            FileName="/home/sharedHA/monitor.txt" FilterPattern="" IsAppend="true"
            LogPattern="%d{dd/MMM/yyyy HH:mm:ss}%m%n" MaxBackupIndex="4"
            MaxFileSize="100000000" MaxFilterLevel="3"
            MinFilterLevel="1" PrintTarget="System.err" ThresholdLevel="10"/>
    </LOGGER>
```

Save the file and we are ready to start the server.

6.4.5 Verifying HA Setup

On starting the Fiorano SOA Server that is part of an HA pair, the server prints debug information about its own state (ACTIVE, PASSIVE, and ACTIVATING). It also prints information about its backup server's state whenever it detects a change.

Example Statements on console:

[Tue Apr 28 16:57:50 IST 2009] New status of remote server = PASSIVE

[Tue Apr 28 16:58:07 IST 2009] New status of remote server = ACTIVATING

[Tue Apr 28 16:58:23 IST 2009] New status of remote server = ACTIVE

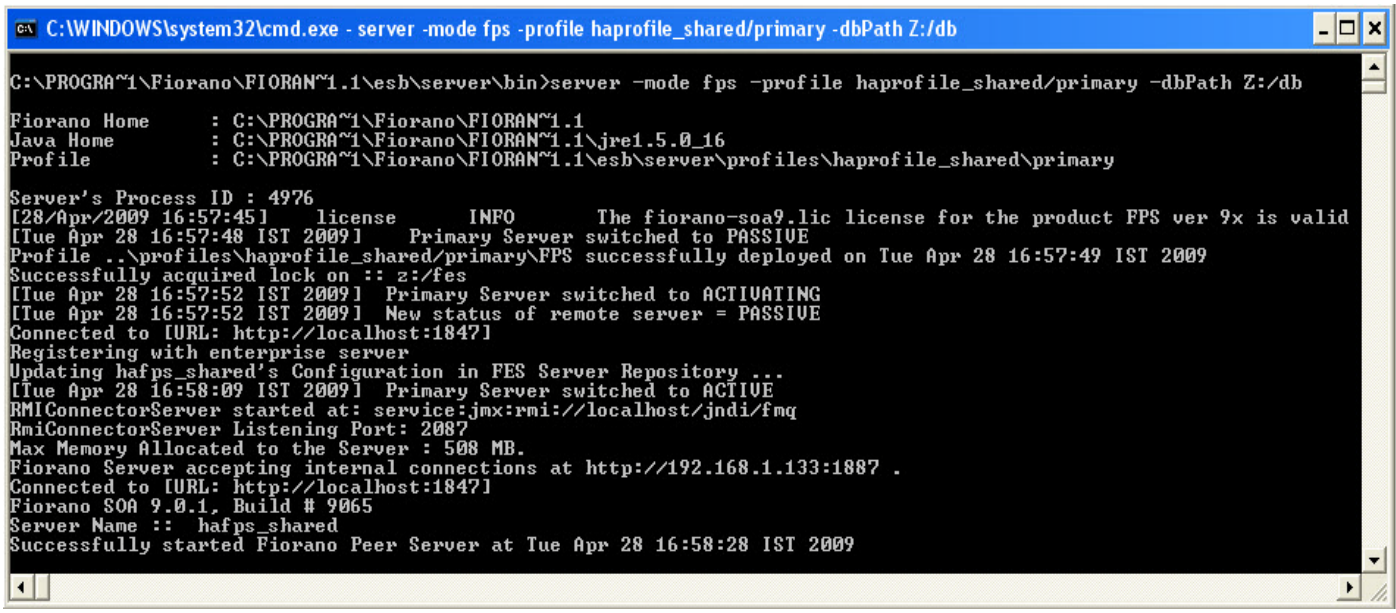
[Tue Apr 28 16:57:52 IST 2009] Primary Server switched to ACTIVATING

The Console includes statements as shown below:

Primary Server switched to ACTIVE or **Secondary Server switched to PASSIVE**, which indicate that the pair has successfully connected. Also, a statement gets printed when the lock is successfully acquired over the lockfile on the active server's console.

Example: Successfully acquired lock on: Z:\lock.txt.

The figure below illustrates a successfully started Fiorano HA Peer Server in shared mode:



6.4.6 Shutting down the HA Server

For more information on how to shutdown the servers, refer to sections 2.3.3.2 and 2.4.3.2.

6.4.7 Troubleshooting Steps

SocketBindException saying that the HA Port is already bound:

This exception means that some other program running on the HA port or the last instance of the server is not properly killed. You can choose to stop or kill the application which is holding up the port and start the server again or choose a different HA port. But changing this means that there needs to be a change in the Backup Servers' configuration for its Backup Server port.

Both servers go to Active state in shared mode respectively if the network link between them is broken. This can happen if the servers do not refer to the same LockFile

The server throws 'log4j: ERROR Failed to flush writer, java.io.IOException: Stale NFS file handle'. This can occur if the log files are present on machine hosting the shared database and they have rolled over. Rolling over of log files sometimes results in having stale file handles (invalid file handles). To avoid this, refer to section 6.4.4.4 Changing the location of log files.

6.5 Limitations of Fiorano SOA High Availability

Following are the limitations of Fiorano SOA High Availability:

1. The current High Availability implementation does not support the PrimaryPreffered Flag. If one wants the Primary Server to be active on its reboot, the Secondary Server has to be manually shutdown.
2. Profiles of Servers prior to Fiorano SOA 9.0.0 release will not work as they do not have the LockFile property in them. One has to copy the profile to $Fiorano_home/esb/server/profiles and open the profile using Studio. The LockFile parameter has to be set and profile has to be saved for the profile to work.

   Also, the LockFile has to be set up on the gateway machine. If the PrimaryPreffered flag has been set in the older profile, the flag will not be respected.
3. The Fiorano SOA High Availability servers can not be run on Mac OS as java does not support locking of files on samba shares on Mac OS.
4. The hot standby Fiorano HA passive server takes approximately 17-20 minutes to become ACTIVE / STANDALONE in the following scenario.
   - Active server on UNIX OS and Lock file on UNIX OS shared using Samba.
   - Network cable of active server pulled out / problem with the network card of machine hosting the active server

Note: This time interval is a characteristic of Samba server on UNIX OS.

The **ActivatePrimaryIfBothStandalone** flag will no longer be respected, as the lock mechanism makes sure that both servers of the HA pair do not be Active/Standalone at any point. This flag has been deprecated from the Fiorano SOA 9.0.0 release onwards.

6.6 Reference Matrix – HA Profile

By default following are the ports that are configured in the given X pattern profiles:

Profiles Port	**FES**	**FPS**	**FPS1**	**FES HA Primary (Rpl)**	**FES HA Secondary (Rpl)**	**FPS HA Primary(Rpl)**	**FPS HA Secondary(Rpl)**	**FPS1 HA Primary(Rpl)**	**FPS1 HA Secondary(Rpl)**	**FES HA Primary(Sh)**	**FES HA Secondary(Sh)**	**FPS HA Primary(Sh)**	**FPS HA Secondary(Sh)**
RMI Port	2047	2067	2077	2047	2048	2067	2068	2077	2078	2047	2048	2087	2088
External Port	1947	NA	NA	1947	1948	NA	NA	NA	NA	1947	1948	Na	Na
Internal Port	1847	1867	1877	1847	1848	1867	1868	1877	1878	1847	1848	1887	1888
Backup Server Port	NA	NA	NA	1848	1847	1868	1867	1878	1877	1848	1847	1888	1887
HA Port	NA	NA	NA	1747	1748	1767	1768	1777	1778	1747	1748	1787	1788
Backup HA Port	NA	NA	NA	1748	1747	1768	1767	1778	1777	1748	1747	1788	1787
Primary FES Port	NA	1847	1847	NA	NA	1847	1847	1847	1847	Na	Na	1847	1847
Backup FES Port	NA	1848	1848	NA	NA	1848	1848	1848	1848	Na	Na	1848	1848
Jetty Port	1980	1880	1890	1980	1980	1880	1880	1890	1890	1980	1980	1900	1900

Where:

- FES: Fiorano Enterprise Server
- FPS: Fiorano Peer Server
- Rpl: Replicated Mode

Note:

- Clients only need to be aware of following ports; details of all other ports are only for informational purpose
- 1947 to connect to Fiorano Server primary from external tools.
- 1948 to connect to Fiorano Server secondary from external tools.
- RMI Ports are used to edit server parameters in online mode.

6.7 Determining Server State

The states of the primary and secondary servers can be determined by running the sample present in $FIORANO_HOME/esb/samples/HAStatus.

The sample needs two parameters to run:

- IP Address of either Primary or the Secondary Server.
- RMI Port of the server whose IP Address is provided.

If the sample fails to connect to a server, it could mean that the server is either Dead or not reachable. In such a case, try giving the IP Address of the other server. If this fails too, this could mean that both servers are Dead or unreachable in the network.

The value of the RMI Port can be found in the profile of the server. It is referred to as the RmiServerPort parameter. For more information on configuring profile, please refer to section 6.3.4.2

Note: This sample can be used to determine the states of the servers running in replicated mode only.

Chapter 7: Scalability, Load Balancing and Memory Optimization

This chapter discusses how Fiorano Composite Applications and Event Processes can be scaled up to increase the overall throughput and to better utilize distributed system resources across the Enterprise Service Grid.

Fiorano's unique peer-to-peer grid-enabled architecture allows hardware resources scattered across the network to be effectively utilized in order to balance the load resulting in scaling the number of processes running concurrently.

7.1 Server-Level Load Balancing

Load balancing is supported at the service level as well as the server levels.

7.1.1 Scaling by adding more peers to the network

At the server level, the Fiorano peer-to-peer architecture enables increasing loads to be seamlessly distributed across the network through the dynamic addition of peer servers. Since data flows between distributed processes are not routed through a central hub, the Fiorano architecture avoids load-related faults that plague most existing integration infrastructure solutions. The Peer-to-Peer architecture also allows dynamic load-balancing to be added to running applications that are already in process.

To scale an Event Process across multiple peers, one needs to add more peers to the network and then re-deploy some existing running components onto these new peers.

This is done easily as follows:

1. Add the new peers to the network using the Administration tools.
2. Stop one or more of the components in the flow, right-click the component and select Kill from the drop-down menu; this stops the components' execution on its current peer. Now change the Peer-Server name on which the component is to be redeployed by selecting the appropriate new Fiorano Peer Server target from the available list of peers in the Properties panel for the Component Instance.

7.1.2 Scaling by distributing load across multiple service instances

At the service level, the Load-Balancing Business Service may be used to distribute the messages to multiple business services for processing.

To enable service-level load balancing on a service, one normally runs multiple instances of the service on the different ESB Peers and routes incoming data to the multiple running instances via a Distribution Service, as illustrated in Figure 7.1.1. The Distribution Service routes data to its output ports depending upon the relative weights associated with the ports. Load-balancing of Services across different nodes as defined earlier can be set up directly by the End-User or Administrator within the application-flow and does not require any programmatic approach. Alternatively, this load balancing can be achieved programmatically as well, since all operations on the Fiorano platform are supported via intuitive APIs.

The load-balancing method described above increases the overall message throughout, since incoming messages are now processed by multiple instances of a service, typically running on different machines (or being processed by separate threads on a single machine). A good example is the fast insertion of data into a database via the Database Business Service. This technique is particularly very useful when a particular business service encounters a bottleneck in the entire business flow. However, the drawback is that in general the messages may not be processed in the order in which they arrive, and this lack of ordering is the trade-off for performance using this load-balancing technique. It is possible to use an event-sequencing flow to ensure message ordering, however, this is not as optimal for increased performance as the general-purpose method outlined above.

7.1.3 An example of Load Balancing

Consider a case of Purchase Order (PO) processing, where a PO generation is very fast compared to the PO processing time which typically requires, among other things, some form of XSLT transformation. In such a process, the PO manipulation becomes the bottleneck in a large PO Processing Business flow.

Since the PO processing step (which typically uses an XSLT transformation) is slower than the PO generation and there is no way to increase its performance, it makes sense to have multiple instances of the PO processing step with each instance sharing the load between them.

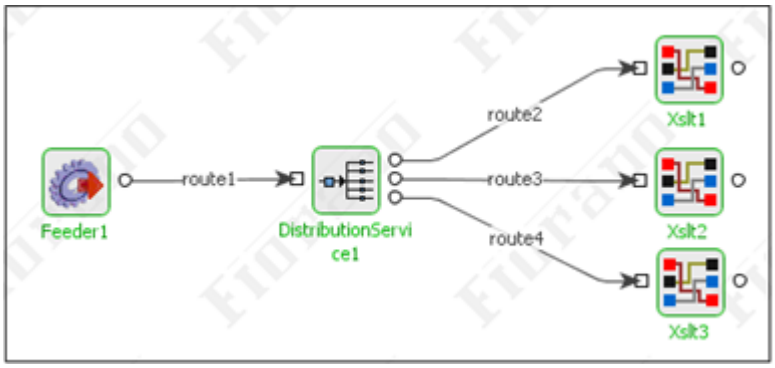


Figure 7.1.1: Load distribution across service instances

By the way of example, load distribution across multiple instances of an XSLT service is displayed in Figure 7.1.1.This figure includes three separate services:

1. **Feeder1 (Feeder Service)** — A Feeder Service sends a large number of PO documents. This step is comparatively fast in comparison to the XSLT Service which forms part of the PO processing step.

2. **DistributionService1 (Distribution Service)** — A Distribution Service evenly distributes the load between three XSLT Services. The Distribution Service can also be configured to distribute load according to the relative weight assigned to each output port. As such, the load can be proportionately distributed between slower and faster machines.

3. **Xslt3 (XSLT Service)** — The example illustrates three instances of the XSLT Service running on same or different peers.

7.2 Thread Count of Components

By default, a service component is single threaded, that is, there is only one document being processed by a service component at any given time. If the CPU is not being fully utilized during a load test, then the performance and throughput can be increased by increasing the # of sessions (that is, threads) for a component. This linearly increases the number of data-elements being processed by the component concurrently. The number of threads per component can be set within the Properties window of the InPort of each component in the Studio.

Always start off with the bottleneck component when optimizing threads this way since there will typically be only one bottleneck component in a given process at a time. Once the first bottleneck component is fixed, the bottleneck typically moves to another component in the flow. This technique helps optimizing the number of sessions/threads for each component based on the ability of the hardware to process data flowing through the Event Process.

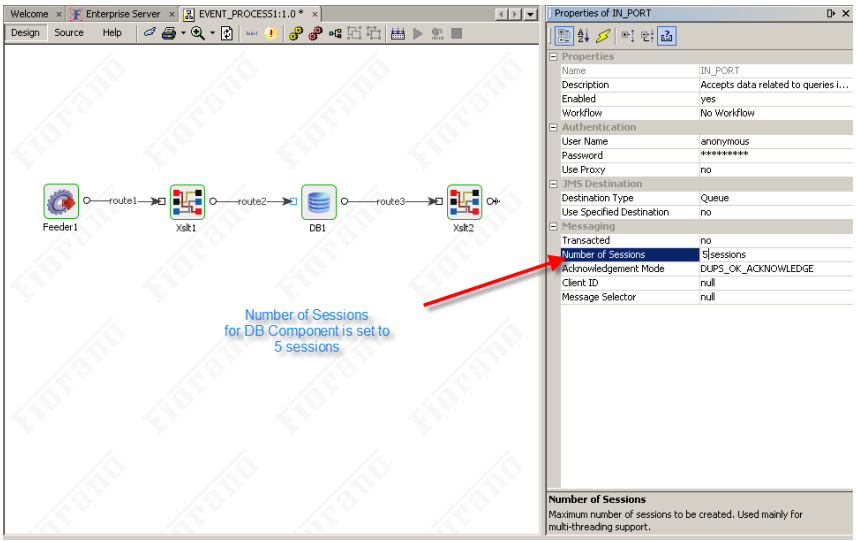


Figure 7.2.1: INPORT Properties Screen

7.3 Scalability

The Distributed Process model implemented by the Fiorano SOA Platform automatically ensures that as many operations as possible are run in parallel. The data flows between applications are represented by graphs, and all independent trees within the application graph represent concurrent computations. Within a single tree, operations are implicitly serialized based on in-built data-flow dependencies.

Scalability is critical to ensure that the platform scales both with current projects (likely in themselves to be highly distributed, probably across company boundaries) and with future growth. The Fiorano platform addresses scalability issues as follows:

7.3.1 Transparent Resource Addition

Fiorano SOA Platform promotes a linear 'build as you grow' model, which allows an enterprise to add software resources in the form of Fiorano Peers at network end-points to absorb additional load on the platform. For instance, if the load on a given set of Peers processing data is determined to be too high, new Peers can be added incrementally to the network at runtime without disrupting existing services and distributed processes. Since Fiorano platform Peers reuse existing enterprise hardware, resource addition typically does not involve additional hardware deployments, unless explicitly required.

7.3.2 Dynamic Change Support

Distributed processes and applications deployed on the Fiorano platform can be extended and modified dynamically at runtime by adding or removing the new services and data routes without stopping or disrupting existing processes in any way. Existing services within an application can be individually controlled via the start/stop/upgrade/modify semantics, allowing incremental, dynamic, and runtime changes to distributed processes.

7.3.3 Parallel Data Flow

With dispersed computation and parallel data flow between nodes, Fiorano Peers scale naturally and seamlessly with the addition of new Peer nodes and Enterprise Services across the network. Information and data flowing between distributed services does not have to pass through a central hub because each Fiorano Peer incorporates a JMS-compliant messaging server, allowing direct Peer-to-Peer connections to be set up on-the-fly between any set of Peers across the network. This on-demand creation of Peer-to-Peer data-flow connections is unique to the Fiorano platform and enables linear scalability and performance as new peers are added to the system. Furthermore, since peers can be hosted on existing (already reasonably powerful) hardware at the end-point of the network, enterprises do not have to purchase expensive hardware each time if there is an increase in performance requirements.

7.4 Memory Optimization

Although Fiorano ESB architecture scales very well in a distributed environment, Fiorano Users and Developers are encouraged to adopt the following strategies:

7.4.1 JVM Parameters

Tune the JVM Parameters according to memory consumption within your environment. The most important parameters are:

- **Xmx:** Denotes the maximum java heap size. You may see the amount of memory you use exceeds the amount specified for the Xmx parameter. While Xmx limits the java heap size, java will allocate memory for other things, including a stack for each thread. It is not unusual for the total memory consumption of the VM to exceed the value of –Xmx.
- **Xms:** Initial java heap size.
- **Xss:** Each thread in the JVM is allocated a stack. The stack size limits the number of threads per JVM. If the stack size it too large, it will result in memory running out as each thread is allocated more memory than required. If the stack space is too small, there will eventually be an exception StackOverflow error. If the stack space is too large, there will eventually be an exception similar to "Unable to create native thread", if the server tries to create more threads.

The default maximum JVM heap size is 64 MB. The Fiorano tools leave the JVM parameters as the default parameter, that is, 64 MB heap memory for each of the components. This JVM parameter can be fine tuned to reduce the memory footprint of individual service component instances. The amount of memory allocated per JVM can and should be reduced for smaller components (such as, flow-control components) or increased for memory-heavy components (such as, XSLT, Database Adapter, and so on).

The default heap size for server (Enterprise and Peer) can be set in the following files (that is, the file applicable based on the script being used to run the server):

- $FIORANO_HOME/esb/server/bin/server.conf
- $FIORANO_HOME/esb/fes/bin/fes.conf
- $FIORANO_HOME/esb/fps/bin/fps.conf

For information on other recommended JVM parameters, please refer to sections 7.6.6 and 7.6.7.

Individual Components have 64 MB max heap size. This can be changed for each component by setting the JVM_PARAMS property in the Studio as shown in Figure 7.4.1.

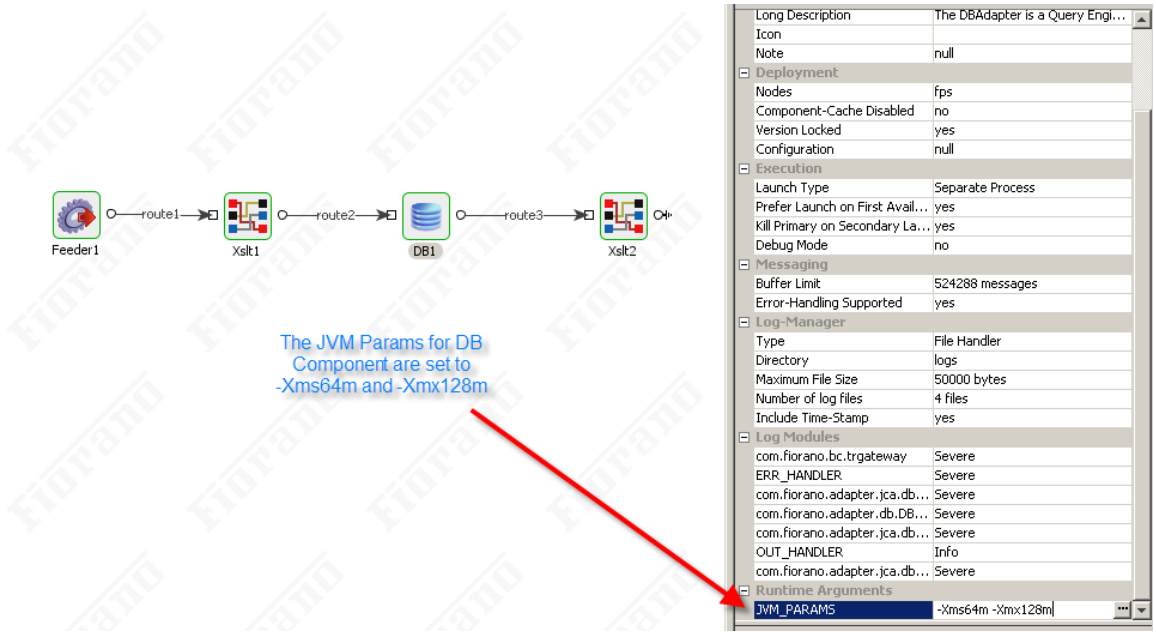


Figure 7.4.1: JVM_PARAMS Properties Screen

7.4.2 Separate Machines for Servers

Run the Enterprise Server and Peer Servers on different machines. This helps in getting additional available memory for components.

7.4.3 Distribute Components

Distribute the flows across all the peers. Group the components inside flows logically so that a single unit of work is done using just one peer.

7.4.4 Inter-Connect Flows

Use port bindings to interconnect multiple flows running on a single Peer (if required) in place of External Business components. For example, if the Business Process definition requires data to flow from EventProcess1 (EP1) to EventProcess2 (EP2), then the OUT_PORT of the last component in the EP1 is bound to a specific destination (for example, MY_DESTINATION). The IN_PORT of the first component of the EP2 Event Process is then bound to the same destination (MY_DESTINATION), as illustrated in Figure 7.4.2.

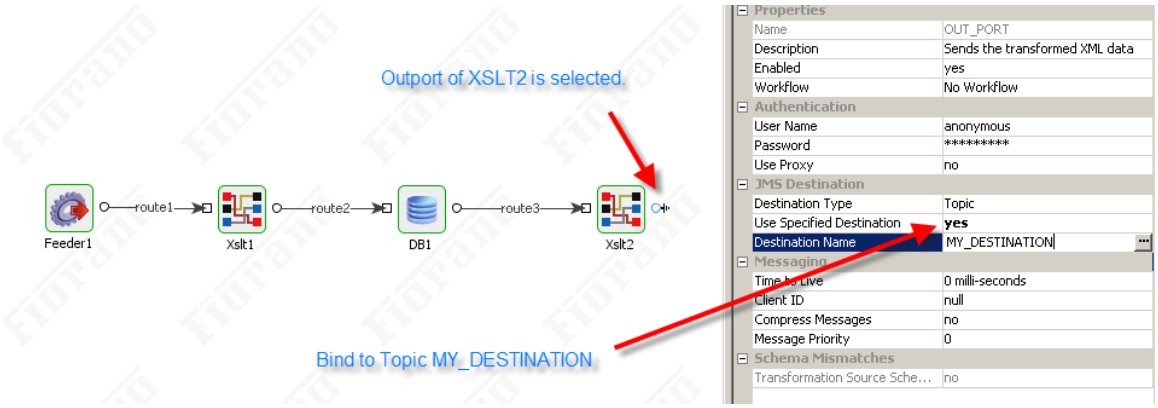


Figure 7.4.2: OutPort Properties Screen

7.4.5 Size of Event Flows

Avoid using large number of component instances per event flow. Keep the flows modular and distribute functionality across multiple flows.

7.4.6 Size of Messages

Keep the sizes of each message flowing through the Fiorano workflow small and not greater than 1 MB. If a given message is too large, then split up the message into smaller messages using the xml-splitter component, or in case of the Database adapter, limit the response size to an appropriate value.

Do not keep large xml's in the Application Context. The Application Context should be used as storage for context and temporary results related to each message. Instead of carrying a binary attachment (for example, PDF, XML, Images, and so on) with Tifosi Document/JMS Message or ESBRecord across the flow, make use of a SAN/NAS to store the attachment and carry forward only references**. Please note** that the above technique works even when the flow executes across distributed machines. The technique is to store large binary attachments in one location (essentially a file somewhere on the network), pass just a reference around the flow and then on the appropriate step to access the (large) binary as needed. Also, the binary attachment in most cases is not parsed at every step of the event process. As such, it is not needed at each step of the process and a simple reference will do.

Keep in mind that not all messages can be split. Splitting works only when each of the final messages become a standalone document after the split. For this reason the XML Splitter cannot be used for all large messages.

7.4.7 DB Adapter Tuning

Use a larger number of SQL queries per DB Adapter instead of using a separate DB adapter per SQL query. Additionally, use the DBQuery/DBProc components wherever possible instead of DB Adapter as the latter is a much heavier component.

7.5 Component Memory Tuning

In the Fiorano Environment, each component is launched (by default, unless specifically configured otherwise) in a separate JVM Process. It is very important to tune the heap memory allocated to each service component and the Fiorano servers to achieve stable, high throughput, and highly optimized memory usage of the Fiorano Environment.

7.5.1 Tuning Memory for Service Components

Each service component has to be tuned to achieve high performance and stability. There cannot be strict recommendations on the heap memory usage for components, as the memory usage of a component is highly dependent on the "context" in which it is used. The Context could be the physical hardware like the number of processors/physical memory size, the configuration with which the component is running, the size of the incoming message, the frequency of service invocation, JVM specific parameters like the version, type of JVM (32 bit/64 bit), the JVM garbage collection algorithm and other memory competing software(s) running on the system and, finally, in some cases the internal implementation of the component itself.

This document makes some recommendations for components running under preset contexts. These recommendations should be considered as guidelines and ideally the tuning process has to be done based on the general guidelines outlined herein along with the specific instructions for each service component instance.

7.5.1.1 Know about Heap sizes

The heap is a section of memory used by the JVM to store the Java objects. You can set constraints on the size of the heap with two parameters passed to the JVM at startup: -Xms sets the initial size of the heap and -Xmx sets the maximum size of the heap. If you set those two parameters to different values, say 32 MB and 256 MB, then it is vital to instruct the JVM to start with a heap size of 32 MB and increase the size up to 256 MB "as necessary". In this case, the JVM has to balance two conflicting constraints: not request too much memory from the operating system (getting to 256 MB too quickly), and not request too little memory as that increases the amount of time the JVM spends on garbage collection which reduces the performance of the application.

Asking the JVM to balance memory usage and performance by setting different values for -Xms and -Xmx is critical in a situation where multiple components are being run on the same hardware system and the components are actively competing for resources - in particular, memory resources.

7.5.1.2 Default Heap size

By default, all components are launched without any JVM Memory settings. The following rules govern the size of the heap allocated by default by JVM if no memory settings were specified explicitly.

From J2SE 5.0, a machine is classified as a server-class machine if it has:

- 2 or more physical processors
- 2 or more GB of physical memory

Java version	Default Garbage Collection Algorithm	Machine class	Initial Heap Size	Maximum Heap Size
1.4 and before	Serial Garbage Collector	Client/Server	4 MB	64 MB
J2SE 5.0 and above	Throughput garbage Collector	Client	4 MB	64 MB
J2SE 5.0 and above	Throughput garbage Collector	Server	1/64 of physical memory up to 1 GB	¼ of physical memory up to 1 GB

The definition of a server-class machine applies to all platforms with the exception of 32 bit platforms running the Microsoft Windows operating system. On all other platforms, the default values are the same as the default values for JDK version 1.4. The following table illustrates the choice of the machine class for common platforms and Operating system combinations.

Platform	Operating System	Default	Default class
Sparc (32-bit)	Solaris	Client	Server
I586	Solaris	Client	Server
	Linux	Client	Server
	Windows	Client	Client
Sparc(64-bit)	Solaris	Server	Server
AMD(64-bit)	Linux	Server	Server

	Windows	Server	Server
IA-64	Linux	Server	Server
	Windows	Server	Server

7.5.1.3 Setting Heap sizes

The -Xmx and –Xms settings can be set for a particular Fiorano component as part of the JVM_PARAMS runtime parameters attributes as illustrated in Figure 1. Multiple JVM runtime parameters can be added and separated with a space as a delimiter. The Heap Size setting should be set in the following format: –Xmx[value][[optional]mem_char]. By default, all specified values are measured in bytes. You can append the letter `k' or `K' to the value to indicate kilobytes, `m' or `M' to indicate megabytes, and `g' or `G' to indicate gigabytes. This attribute is a common attribute for all service components.

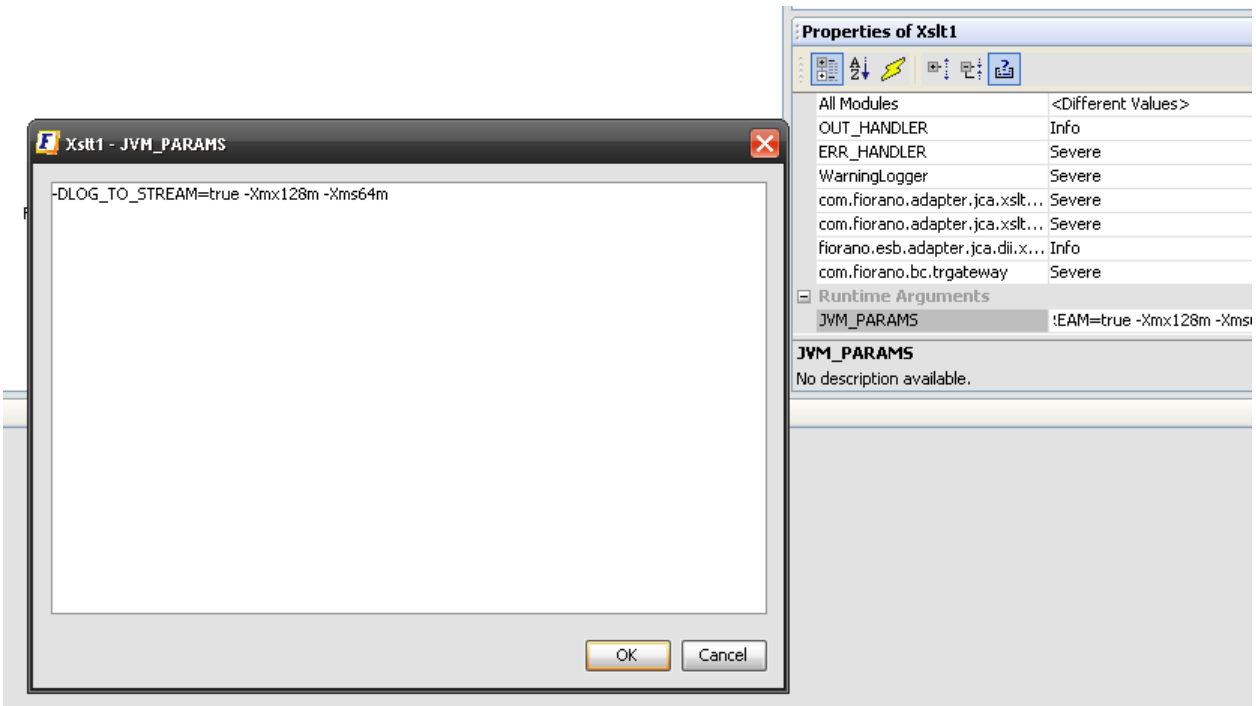


Figure 1: Setting the JVM _PARAMS attributes of a Component Instance

7.5.1.4 Garbage Collection

Garbage collection is a process of releasing the memory currently allocated by unused java objects in the Heap Space. The Heap Space contains all the objects created in the Java program. When an object is no longer used by any of the pointers, the object is garbage and it can be released. The recommended values for -Xmx and -Xms largely depend on the load/size of the messages processed by the component and other component implementation specific parameters. The JVM heap size specified for the component determines how often the VM collects garbage and how much time it spends on each garbage collection sweep. An acceptable rate for garbage collection is application-specific. It is generally recommended that the JVM spends significantly less than 30% of its time on garbage collection; this time should be adjusted after analyzing the actual time and frequency of garbage collections. The goal of tuning the heap size is to minimize the time that it spends doing garbage collection while maximizing the number of messages that the component can handle at a given time. It is recommended that the following factors are analyzed for the optimum setting of the Xmx and Xms parameters:

1. Frequency of garbage collection? If the garbage collection is occurring too frequently, then increasing the maximum heap setting is recommended.
2. How long is garbage collection taking? As a recommendation, full garbage collection should take no longer than 3 to 5 seconds. If it takes more time, then reducing the maximum heap size is recommended.
3. What is the average memory footprint? In other words, what does the heap settle back down to after each full garbage collection? If the heap always settles to 85% free, reducing the maximum heap size is recommended.
4. Typically the -Xmx parameter settings for all the components and the Peer Server should account for 80% of the available physical memory.
5. Typically the -Xms parameter settings for all the components should be much less than 50% of the available physical memory.

7.5.1.5 Monitoring Component JVM Statistics

There are several JVM profilers available for monitoring the heap size, the garbage collection statistics, the thread-count and the count of the number of classes loaded for the component JVM. Java has an inbuilt tool to profile the JVM using JConsole as part of the Java SDK distribution. Fiorano recommends the free Netbeans profiler for monitoring the component JVM statistics. The exact set of steps depending on the version of Netbeans used and can be found at www.Netbeans.org.

1. Run the Netbeans Profiler Calibration
2. Configure the Netbeans profiler
    a. The Profiler task should be to analyzing memory usage
    b. It is recommended that the Profiler is setup on a different machine than the machine on which the Fiorano Peer Server is installed, because profiling in itself is a CPU-intensive, memory consuming task.
    c. Netbeans provides the runtime arguments to be added to the external application on launch. Copy the parameters as specified and as required into a text editor.

3. Configure the Fiorano component runtime parameters. Add the copied runtime arguments from step 2 (c) above to the service component JVM_PARAMS runtime argument as shown in figure 2.

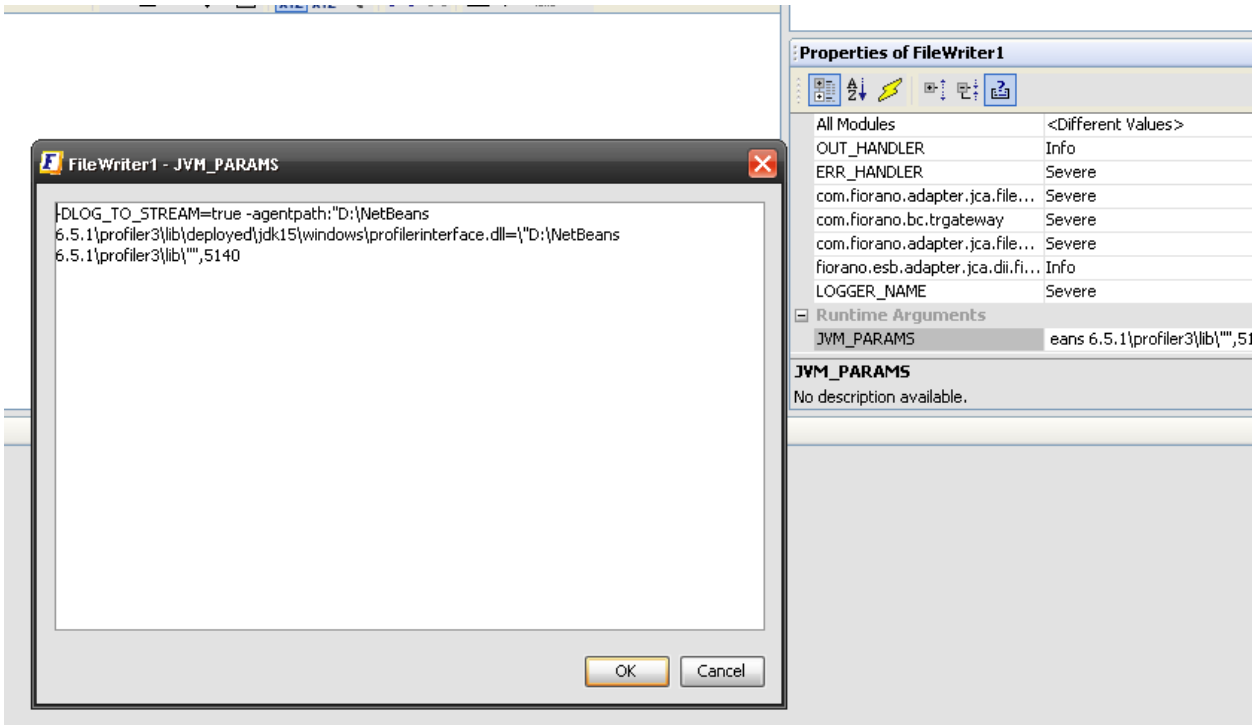


Figure 2: Adding the profiler settings to the JVM_PARAMS

4. Launch the component / application. **Note**: the component will not start until the Profiler is attached to the Java process.
5. Attach the Netbeans Profiler. The profiler should start collecting the statistics on heap memory usage together with the garbage collection statistics. Switching to VM telemetry view is needed.

7.5.1.6 Tuning the memory settings

It is recommended that the production environment be simulated to determine how the component / application is typically used before one starts tuning memory settings. Once the component is running and the profiler is attached an analysis can be performed to determine whether a component is stable, suffocated, dangerous or well tuned, as described in the sections that follow.

Stable Component

The graph of figure 3 below suggests that the memory utilization of a component after the initialization of a very minimum heap size, shoots up to a specified level and remains at that level even after you have requested a Full System Garbage Collection (Using Netbeans). (**Note:** Java does not guarantee that Garbage collection will happen if the command is executed, but there is a good change it does for the most part). Such a level is a good recommendation for the Xms setting of the service component. The service component memory usage is stable under the tested scenario and it is recommended that the Xmx and Xms settings be set to values on the upper bound and lower bound of the graph respectively.

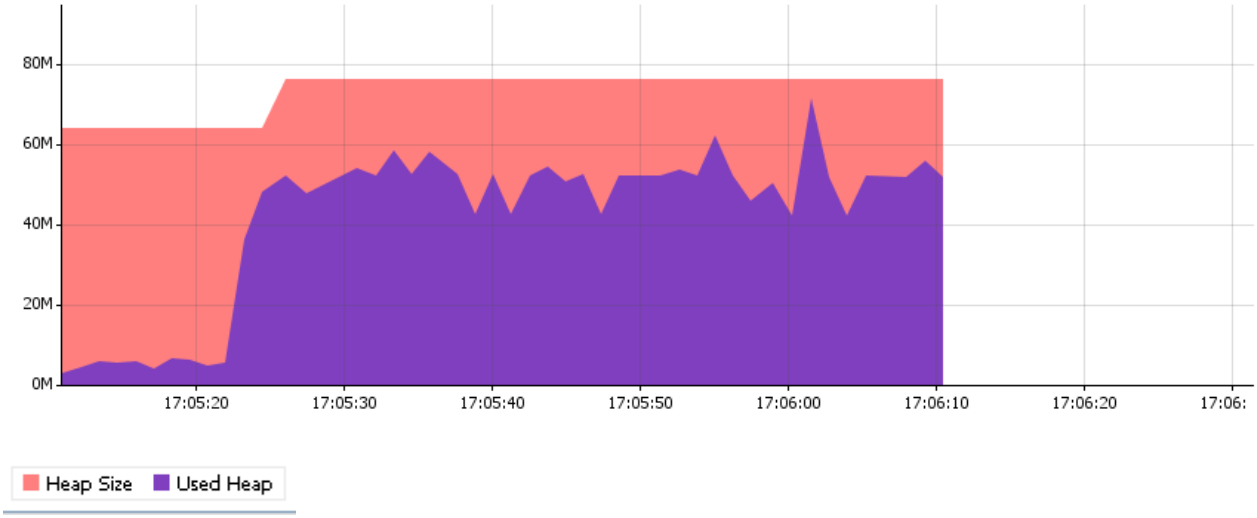


Figure 3: Stable component memory utilization

Suffocated Component

Figure 4 graph suggests that the actual used heap memory reaches the allocated heap memory too often and when the allocated heap memory is below the Maximum memory setting, a partial garbage collection is done on the Eden space (younger generation) of the heap space; this is fine as long as the Eden space is kept at low number (default not modified), but if the allocated heap memory in the above case is set to the Maximum memory, a full System garbage collection is issued, which ideally should not occur too often. When the graph shows steep peaks too often as illustrated in figure 4 and if the allocated heap size is close to the maximum memory utilization of the component JVM, then it is recommended that the allocated heap space should be increased.

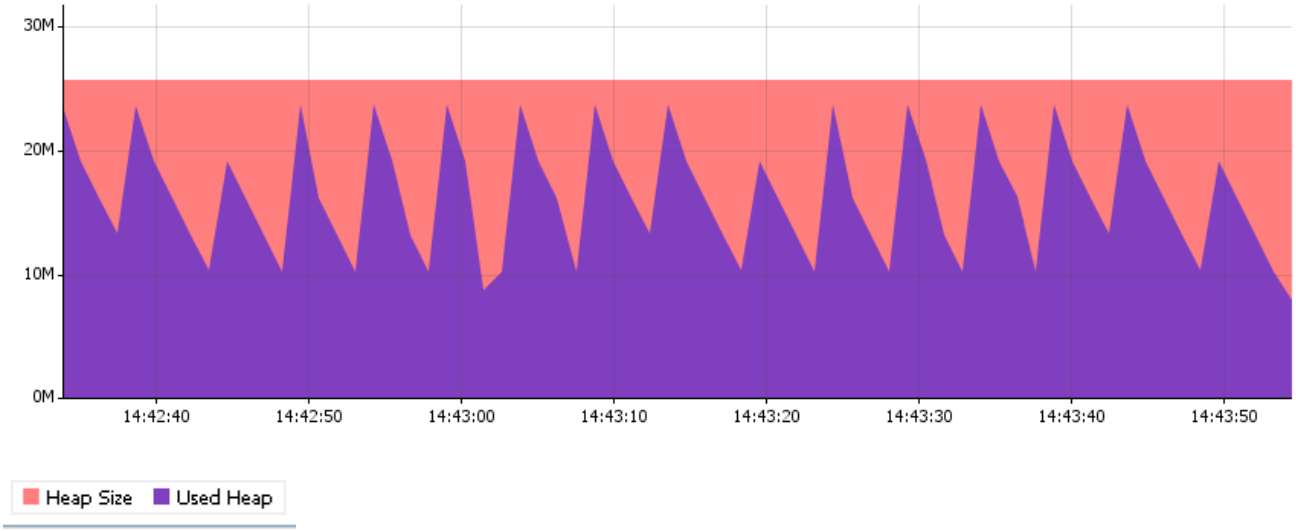


Figure 4: Suffocated component memory utilization

Typically for "suffocated" components, the garbage collection statistics are also very important to observe. If the component is spending more than 30% of the time on Garbage collection, then there is a good chance that the component is underperforming significantly. It is therefore recommended that the maximum memory heap space should be increased in such cases. The GC static graph in figure 5 below definitely suggests that the component needs more heap space.

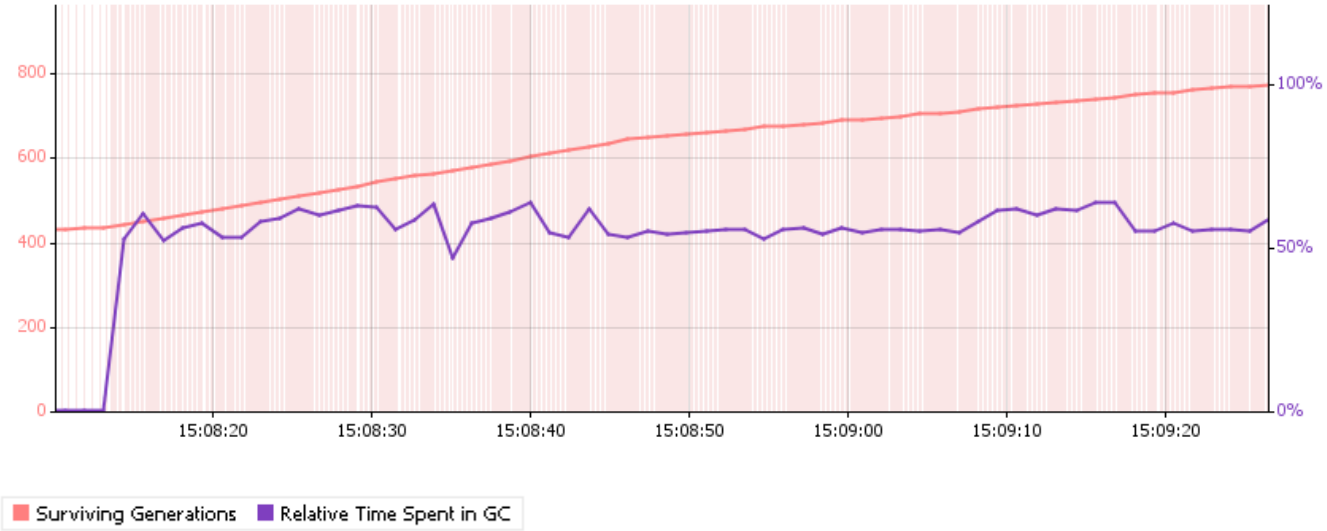


Figure 5: Memory utilization of a Component instance that needs more heap space

Dangerous Component

It is typically the case that most users tune a component with a very high heap Memory setting for the best performance since the frequency with which garbage collection happens will then be very low and most processor time for the JVM is dedicated to the component. This is a good option when it is possible to calculated or known for certain the component does not reach the maximum memory limit or, if it does reach the maximum memory limit, that the time then taken for garbage collection is acceptable. Because the heap memory limit is very high, the Garbage Collection process takes proportionately longer. So a downside of setting a high upper limit on the heap space for a component instance is that the Garbage Collection Process can sometime take an inordinate amount of time. This can have some repercussions as discussed below:

The distinct plateaus in the garbage collection statistics of figure 5 above suggest that for the length of each plateau a full GC has been running which is generally a "Stop the World" process, making the component unresponsive for that length of time. A long garbage collection process can lead to several complications as there can be a random/ considerable amount of delay in the processing time of the component. More importantly, the JMS connection made by the component to the Fiorano Peer Server to which it is connected could be invalidated depending on the ping timeout set on the connection parameter which might cause the component to stop execution or kill itself.

In such cases, it is recommended that the Minimum memory or the maximum memory settings are reduced as appropriate:

- If the time taken on the GC done over the Eden space of the component JVM Heap when the used heap space reaches the minimum memory setting is unacceptable (which is less likely), then it is recommended to reduce the minimum heap setting.
- If the time taken on the Full System GC over the entire Heap space including perm gen of the component JVM Heap when the used heap space reaches the maximum memory setting is unacceptable (which is more likely), then it is recommended to reduce the maximum heap setting and perform the tuning again.

Well-tuned component

A well-tuned component, as illustrated in figure 6, neither performs garbage collection very frequently nor does it take an unacceptable amount of time performing garbage collection. The heap memory size does not shoot up or come down drastically at any point. It is generally quite stable increasing from the minimum heap size to the maximum heap size and then back to the minimum heap size, which should be the amount of memory needed by the component.

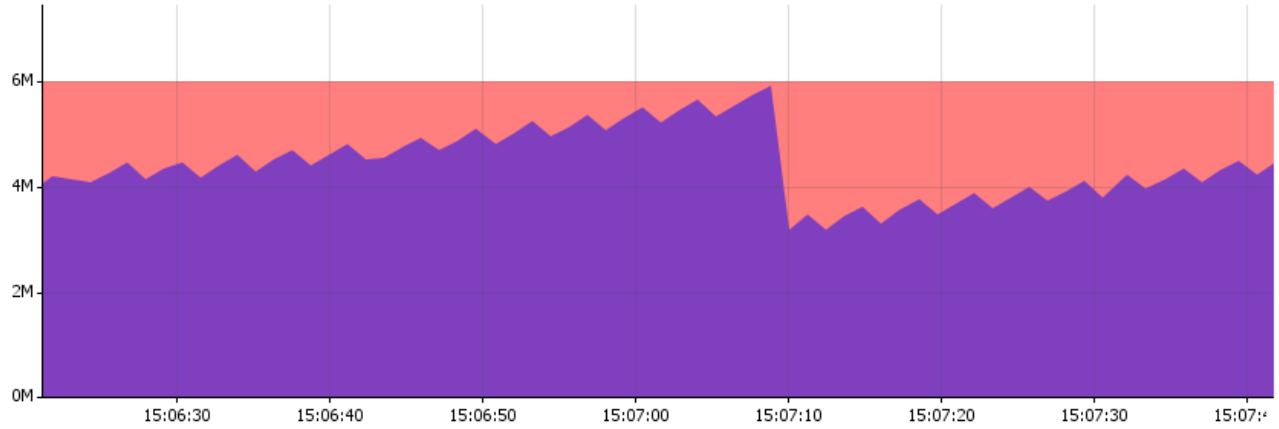


Figure 6: A well tuned component

As shown in figure 7, the garbage collection graph displays uniform behavior below the 30 % mark and there are no high peaks for an extended period of time. This balance is very important to strike for every Fiorano service component instance.

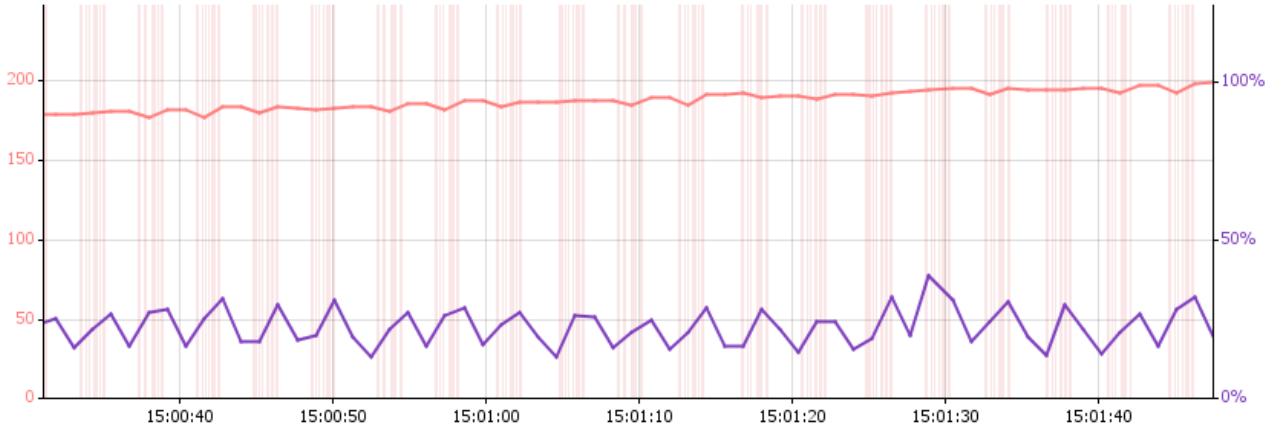


Figure 7: Garbage collection graph for a well-tuned component

7.5.2 Recommendations

7.5.2.1 Component Overloading

During the application design/architecture phase, the following factors are to be considered regarding the number of components to be launched on a Fiorano Peer Server.

1. Component Maximum Heap Size and the Minimum Heap sizes after performing the tuning for heap size memory as discussed in the previous sections.
2. List of memory and CPU intensive programs running on the same system and the Maximum memory required for these programs.
3. Fiorano Peer Server Heap setting after it is tuned with all the in-memory components in a simulated production scenario.

Fiorano recommends the following:

1. The Fiorano Peer Server and Fiorano components should be the only memory and CPU intensive applications running on the production system.
2. Fiorano Peer Server Xms and Xmx parameters are set to the same value or to values that are close to each other. This sort of setting is generally recommended for servers to achieve optimal performance. The value which needs to be set for the Xmx and Xms parameters should be deduced by performing the tuning operations with simulated production runs.
3. For maximum performance, Fiorano recommends that the server system should have "enough" RAM so that there is never a shortage and that page faults essentially never happen.

4. The Maximum heap size for the Fiorano Peer Server and all the components should not exceed 75% of the virtual memory space. It is recommended to have the total of the maximum heap sizes equal to 85% of the available physical memory. Virtual memory is the sum of the Physical Memory and the Page File Memory.

   So if your computer has 4 GB RAM, and 2 GB RAM is being set as the page file size, then available Virtual memory is 6 GB RAM and 75% of this value is 4.5 GB. If the Fiorano Peer Server and Fiorano components total of the maximum heap sizes is close to 4.5 GB, then this indicates that the "Architectural Limit" has been reached. Loading memory over this limit does not guarantee a stable Fiorano environment.

7.5.3 Components

The following sections provide some guidelines on how different contextual parameters affect the heap memory utilized by some of the most commonly used Fiorano components. These parameters can sometimes be categorized into buckets of common usage; some parameters cannot be categorized as each distinct value leads to a non-extrapolative distinct behavior. For such parameters, the values are fixed based on the most common usage and these values are specified in the recommendations. For all other parameters, based on different categorizations, some memory setting recommendations are made. These numbers are only indicative and should not be used as very strict guideline; they only indicate a rough estimate of the memory settings under the specified mode of usage and can be considered a good starting point for specific tuning based on your own environment and parameter settings.

If the component is launched within the same JVM as the Fiorano Peer Server, the libraries and classes loaded by the peer servers and other components launched in-memory are shared. Since there is no separate process, the JVM does not need to create a separate stack space as the memory spaces are shared between the component and the Fiorano Peer Server. In this case there will only be an increase in the heap space of the Fiorano peer server. The Peer Server JVM has to be re-tuned each time a new component is launched in-memory. For Fiorano Peer Server, use the same procedure to tune the memory settings as discussed in the previous sections for components.

7.5.3.1 File Reader

The File Reader component reads files from the file system and sends their contents to the output port, typically as XML documents. The source file can either be text or binary depending on the value of the "Is File binary" attribute. The following factors determine the setting of an optimum Heap Space for the component.

1. If the file reader is configured to read text files, the Size of the text file, as one single FioranoTextMessage is constructed from the contents of the entire text file.The entire size of the file characters is allocated at the same time.
2. The frequency of reading the files and publishing the message. If Scheduling is enabled, the interval at which the files are being read or the number of files in the working directory (depending on POST_PROCESS_ACTION) or the frequency of messages in the input port of the File reader.

Recommendation 1

Initialization Context:

- Is Input File Binary – no
- Frequency of operation – > 1 sec
- In-memory – no
- Hardware - Dual core 2.1 GHz, 4 GB Ram
- OS - Windows XP Professional 32 Bit
- JVM - version 1.5.0_18, 32 bit

Message Size		Operating Range		Recommended Memory Setting		Time spent in GC	
Minimum	Maximum	Maximum (MB)	Minimum (MB)	Maximum (MB)	Minimum (MB)	Maximum (%)	Minimum (%)
0	5KB	5	4	32	4	5	0
5KB	500KB	15	10	32	4	15	10
500KB	1MB	20	10	64	4	20	15
1MB	5MB	45	28	128	32	15	5
5MB	10 MB	75	50	128	64	20	10
10 MB				256	128		

Recommendation 2

Initialization Context:

- Is Input File Binary – no
- Frequency of operation – ~ 100 milliseconds
- In-memory – no
- Hardware – Dual core 2.1 GHz, 4 GB Ram
- OS – Windows XP Professional 32 Bit
- JVM – version 1.5.0_18, 32 bit

Message Size		Operating Range		Recommended Memory Setting		Time spent in GC	
Minimum	Maximum	Maximum (MB)	Minimum (MB)	Maximum (MB)	Minimum (MB)	Maximum (%)	Minimum (%)
0	5KB	6	4	32	4	5	0

Message Size		Operating Range		Recommended Memory Setting		Time spent in GC	
5KB	500KB	25	10	32	16	15	10
500KB	1MB	30	15	64	32	20	15
1MB	5MB	45	30	128	64	15	5
5MB	10 MB	75	50	128	64	20	10
10 MB				256	128		

Recommendation 3

Initialization Context:

- Is Input File Binary – yes
- Chunk size – 1024 KB
- Frequency of operation – ~ 100 milliseconds
- Hardware – Dual core 2.1 GHz, 4 GB Ram
- OS – Windows XP Professional 32 Bit
- JVM – version 1.5.0_18, 32 bit

Message Size		Operating Range		Recommended Memory Setting		Time spent in GC	
Minimum	Maximum	Maximum (MB)	Minimum (MB)	Maximum (MB)	Minimum (MB)	Maximum (%)	Minimum (%)
0		30	15	64	16	20	15

Note:

- Frequent text file reading (100 ms or less) of 500 KB messages (or more) results in the JVM spending almost 60% of time performing garbage collection. Under conditions such as these, the component under-performs drastically and if the component has a Graphical User Interface, then the interface will be very sluggish or even non-responsive for the period till the garbage collection is being done.
- If the file reader is configured for binary files, then the file size is not a factor anymore; only the chunk size is.

7.5.3.2 File Writer

The File Writer component writes the received data from its input port to the specified output File. The received data can either be plain text or binary data. The following factors determine the setting of an optimum Heap Space for the component:

- The Size of the input message.
- The frequency of the requested operation. If Scheduling is enabled, then the interval at which the files are being read or the number of files in the working directory (depending on POST_PROCESS_ACTION) or the frequency of messages arriving at the input port and written to the output port of the File reader.

Recommendation 1

Initialization Context:

- Frequency of operation – > 1 second
- Hardware – Dual core 2.1 GHz, 4 GB Ram
- OS – Windows XP Professional 32 Bit
- JVM – version 1.5.0_18, 32 bit

Message Size		Operating Range		Recommended Memory Setting		Time spent in GC	
Maximum	Minimum	Maximum (MB)	Minimum (MB)	Maximum (MB)	Minimum (MB)	Maximum (%)	Minimum (%)
0	5KB	4	3	32	4	5	0
5KB	500KB	14	10	32	4	20	10
500KB	1MB	20	15	64	4	20	15
1MB	5MB	60	40	128	64	25	20
5MB	10 MB	110	65	128	64	15	10
10 MB				256	128		

Recommendation 2

Initialization Context:

- Frequency of operation – ~ 100 milliseconds
- Hardware – Dual core 2.1 GHz, 4 GB Ram
- OS – Windows XP Professional 32 Bit
- JVM – version 1.5.0_18, 32 bit

Message Size		Operating Range		Recommended Memory Setting		Time spent in GC	
Maximum	Minimum	Maximum (MB)	Minimum (MB)	Maximum (MB)	Minimum (MB)	Maximum (%)	Minimum (%)

Message Size		Operating Range		Recommended Memory Setting		Time spent in GC	
0	5KB	10	8	32	4	5	0
5KB	500KB	35	25	64	32	10	10
500KB	1MB	40	30	64	32	20	15
1MB	5MB	65	55	128	64	15	5
5MB	10 MB	130	95	256	64	20	10
10 MB				256	128		

7.5.3.3 XSLT

XSLT is a component that executes a XSL and converts XML from one format to another. The memory utilization of this component depends on:

- Transformation Complexity
- Message Size

7.5.3.4 CBR

The Content based router is used to route messages to different routes based on the XML content of the message. The routing logic is based on applying an XPath selector over the incoming message. The following factors determine the setting of an optimum Heap space for the CBR component:

- Complexity of the XPath evaluations
- Number of XPath evaluations and the number of output ports configured
- Size of the input message on which the XPaths are evaluated
- Frequency of input messages

Recommendation 1

Initialization Context:

- Frequency of operation – > 1 Second
- No of Output Ports – ~ 3
- Complexity level of XPaths – Simple (Simple function and operator)
- Hardware – Dual core 2.1 GHz, 4 GB Ram
- OS – Windows XP Professional 32 Bit
- JVM – version 1.5.0_18, 32 bit

Initial component size: 3 MB

Threads loaded: 8 and are constant throughout the execution

Approximate number of classes loaded: 2500

Message Size		Operating Range		Recommended Memory Setting		Time spent in GC	
Maximum	Minimum	Maximum (MB)	Minimum (MB)	Maximum (MB)	Minimum (MB)	Maximum (%)	Minimum (%)
0	5KB	7	4	32	4	0	0
5KB	500KB	12	8	32	4	20	15
500KB	1MB	30	15	64	32	15	10
1MB	5MB	90	55	128	64	10	5
5MB	10 MB	120	100	128	64	10	5
10 MB				256	128		

Recommendation 2

Initialization Context:

- Frequency of operation – ~ 100 Milliseconds
- No of Output Ports – ~ 3
- Complexity level of XPaths – Simple (Simple function and operator)
- Hardware – Dual core 2.1 GHz, 4 GB Ram
- OS – Windows XP Professional 32 Bit
- JVM – version 1.5.0_18, 32 bit

Initial component size: 3 MB

Threads loaded: 8 and are constant throughout the execution

Approximate number of classes loaded: 2500

Message Size		Operating Range		Recommended Memory Setting		Time spent in GC	
Maximum	Minimum	Maximum (MB)	Minimum (MB)	Maximum (MB)	Minimum (MB)	Maximum (%)	Minimum (%)
0	5KB	6	4	32	4	10	0

Message Size		Operating Range		Recommended Memory Setting		Time spent in GC	
5KB	500KB	30	10	64	32	20	15
500KB	1MB	40	20	64	48	10	5
1MB	5MB	100	60	128	64	10	5
5MB	10 MB	120	70	128	64	10	5
10 MB				256	128		

Recommendation 3

Initialization Context:

- Frequency of operation – > 1 Second
- No of Output Ports – ~ 8
- Complexity level of XPaths – Simple (Simple function and operator)
- Hardware – Dual core 2.1 GHz, 4 GB Ram
- OS – Windows XP Professional 32 Bit
- JVM – version 1.5.0_18, 32 bit

Initial component size: 3 MB

Threads loaded: 8 and are constant throughout the execution

Approximate number of classes loaded: 2500

Message Size		Operating Range		Recommended Memory Setting		Time spent in GC	
Maximum	Minimum	Maximum (MB)	Minimum (MB)	Maximum (MB)	Minimum (MB)	Maximum (%)	Minimum (%)
0	5KB	8	6	32	4	0	0
5KB	500KB	12	8	32	4	15	10
500KB	1MB	45	30	64	32	25	10
1MB	5MB	120	80	128	64	15	10
5MB	10 MB	120	100	256	128	10	5
10 MB				256	128		

Recommendation 4

Initialization Context:

- Frequency of operation – ~ 100 Milliseconds
- No of Output Ports – ~ 8
- Complexity level of XPaths – Simple (Simple function and operator)
- Hardware – Dual core 2.1 GHz, 4 GB Ram
- OS – Windows XP Professional 32 Bit
- JVM – version 1.5.0_18, 32 bit

Initial component size: 3 MB

Threads loaded: 8 and are constant throughout the execution

Approximate number of classes loaded: 2500

Message Size		Operating Range		Recommended Memory Setting		Time spent in GC	
Maximum	Minimum	Maximum (MB)	Minimum (MB)	Maximum (MB)	Minimum (MB)	Maximum (%)	Minimum (%)
0	5KB	6	4	32	4	10	0
5KB	500KB	35	20	64	32	20	15
500KB	1MB	50	30	64	48	25	5
1MB	5MB	100	80	128	64	15	10
5MB	10 MB	125	100	256	64	10	10
10 MB				256	128		

7.5.3.5 Aggregator

This component collects and aggregates messages received on its IN_PORT based on a specified Completeness Condition. The collected messages are then forwarded as an aggregated bundle of messages to a component connected to its OUT_PORT. The Aggregator is a special message filter that receives a stream of messages and identifies messages that are correlated. When a complete set of messages has been received, the Aggregator collects information from each of these messages and publishes a single, aggregated message to the output port for further processing.

The Aggregator is not a stateless component unlike other simple routing components (like Content Based Router (CBR)) that are generally stateless. Stateless components process incoming messages one by one and are not required to maintain any information between messages. The Aggregator component also has an option to persist the aggregated message into an RDBMS.

The memory usage of the Aggregator component is heavily dependent on the Aggregation size. The Aggregation size is defined as the total size of all messages to be aggregated. This depends on the completeness condition.

Fiorano does not recommend turning the persistence off if the aggregation size is going to be more than 5 MB. The Aggregator uses DOM based data structures for internal processing of XMLs, and these structures typically utilize memory approximately 3-10 times the size of the input file. The memory required by the component in case of message persistence is set to no, thus directly proportional to the Aggregation size. The following factors determine the amount of heap memory taken up by an Aggregator Component:

- Aggregation Size
- Frequency of Aggregation happening
- Number of messages in the aggregation.

Recommendation 1

Initialization Context:

- Frequency of aggregation – ~ 5 Seconds
- Number of messages aggregated – 5 Messages
- Message Persistence – off
- Hardware – Dual core 2.1 GHz, 4 GB Ram
- OS – Windows XP Professional 32 Bit
- JVM – version 1.5.0_18, 32 bit

Initial component size: 5 MB

Threads loaded: 9 and are constant throughout the execution

Approximate number of classes loaded: ~ 2750

Aggregation Size		Operating Range		Recommended Memory Setting		Time spent in GC	
Maximum	Minimum	Maximum (MB)	Minimum (MB)	Maximum (MB)	Minimum (MB)	Maximum (%)	Minimum (%)
0	10KB	7	3	32	4	0	0
10KB	1MB	25	20	64	16	15	10
1 MB	3MB	45	35	64	32	20	15

Aggregation Size		Operating Range		Recommended Memory Setting		Time spent in GC	
3MB	5MB	120	60	128	64	25	20
5MB		-	-	-	-	-	-

Recommendation 2

Initialization Context:

- Frequency of aggregation – ~ 500 Milliseconds
- Number of messages aggregated – 5 Messages
- Message Persistence – off
- Hardware – Dual core 2.1 GHz, 4 GB Ram
- OS – Windows XP Professional 32 Bit
- JVM – version 1.5.0_18, 32 bit

Initial component size: 5 MB

Threads loaded: 9 and are constant throughout the execution

Approximate number of classes loaded: ~ 2750

Aggregation Size		Operating Range		Recommended Memory Setting		Time spent in GC	
Maximum	Minimum	Maximum (MB)	Minimum (MB)	Maximum (MB)	Minimum (MB)	Maximum (%)	Minimum (%)
0	10KB	7	3	32	4	0	0
10KB	1MB	25	20	64	16	15	10
1 MB	3MB	60	35	128	32	15	10
3MB	5MB	180	130	256	128	15	10
5MB		-	-	-	-	-	-

Recommendation 3

Initialization Context:

- Frequency of aggregation – ~ 5 Seconds
- Number of messages aggregated – 5 Messages
- Message Persistence – on

- Hardware – Dual core 2.1 GHz, 4 GB Ram
- OS – Windows XP Professional 32 Bit
- JVM – version 1.5.0_18, 32 bit

Initial component size: 5 MB

Threads loaded: 10 and are constant throughout the execution

Approximate number of classes loaded: ~ 3300

Aggregation Size		Operating Range		Recommended Memory Setting		Time spent in GC	
Maximum	Minimum	Maximum (MB)	Minimum (MB)	Maximum (MB)	Minimum (MB)	Maximum (%)	Minimum (%)
0	10KB	8	6	32	4	0	0
10KB	1MB	30	20	64	16	15	10
1 MB	3MB	45	35	64	32	25	20
3MB	5MB	-	-	-	-	-	-
5MB		-	-	-	-	-	-

7.5.3.6 Distribution

This component is used for distributing a workload of N Jobs amongst M flow processors. Typically this component is used before multiple instances of the same component and the load balancing mechanism in the component is used to distribute the messages received by this component. The component uses a weighted round robin mechanism. The following factors determine the heap memory used up by the Distribution component:

- Incoming message size
- Message Frequency

Recommendation 1

Initialization Context:

- Frequency of operation – > 1 seconds
- Hardware – Dual core 2.1 GHz, 4 GB Ram
- OS – Windows XP Professional 32 Bit
- JVM – version 1.5.0_18, 32 bit

Initial component size: 3 MB

Threads loaded: 8 and are constant throughout the execution

Approximate number of classes loaded: 2400

Message Size		Operating Range		Recommended Memory Setting		Time spent in GC	
Maximum	Minimum	Maximum (MB)	Minimum (MB)	Maximum (MB)	Minimum (MB)	Maximum (%)	Minimum (%)
0	5KB	3	3	16	4	10	0
5KB	500KB	10	9	32	4	20	15
500KB	1MB	20	10	32	4	20	15
1MB	5MB	80	60	128	32	15	10
5MB	10 MB	90	70	128	64	15	10
10 MB				256	128		

Recommendation 2

Initialization Context:

- Frequency of operation – ~ 100 Milliseconds
- Hardware – Dual core 2.1 GHz, 4 GB Ram
- OS – Windows XP Professional 32 Bit
- JVM – version 1.5.0_18, 32 bit

Initial component size: 4 MB

Threads loaded: 8 and are constant throughout the execution

Approximate number of classes loaded: 2400

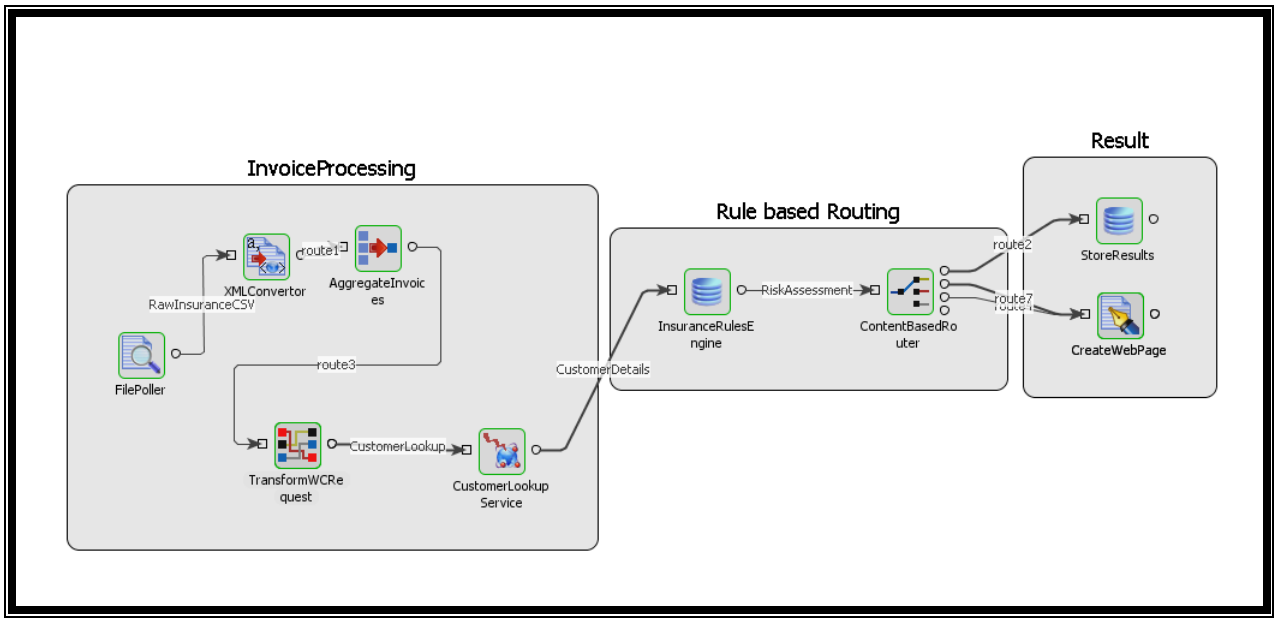
Message Size		Operating Range		Recommended Memory Setting		Time spent in GC	
Maximum	Minimum	Maximum (MB)	Minimum (MB)	Maximum (MB)	Minimum (MB)	Maximum (%)	Minimum (%)
0	5KB	6	4	16	4	5	0
5KB	500KB	15	10	32	4	15	10
500KB	1MB	35	25	64	16	10	5

Message Size		Operating Range		Recommended Memory Setting		Time spent in GC	
1MB	5MB	80	60	128	32	25	15
5MB	10 MB	100	70	128	64	25	20
10 MB				256	128		

7.5.4 Walkthrough

In this section, we consider a sample application walk-through the process for tuning the memory that is allocated to all the component instances.

7.5.4.1Application



The above application performs the following actions:

1. Pick up a CSV (Comma Separate Variable, text file) Insurance Request file from a Directory location and send the file contents to a Text2XML Converter.
2. The Text2XML Converter converts the incoming file into XML and then sends the XML packet to an aggregator component.
3. The aggregator component aggregates 100 messages (which represent invoices) and then the Mapping component stores some context data (invoice numbers and customer ID) in the Application context, along with the request for the Web service invoker component is framed and forwarded.
4. The Web Service component invokes a web service to get some customer details and looks up to a database to make some decision based on a Content Based Router component.

5. Results are then written as a webpage and also stored into a database.

7.5.4.2 Tuning Process

Step 1:

Estimate the configurations which will be used in the production scenario for the components and also the hardware on which the application will be hosted for production. Determine the production system load (with reference to the applications which may be already running in the system and available CPU cycles and Physical memory).

The following servers run on the test system:

1. Fiorano Peer Server
2. MY SQL Community Edition
3. Tomcat Container with Axis hosting Web services for use by the application.

Hardware Configuration:

Windows XP Professional x86

JRE 1.5.0_18 with HOTSPOT off

Core2Duo processor 210GHz

Total Physical Memory - 4GB

Available physical memory - 2 GB

Virtual Memory - 2GB

CPU Usage - LOW below 15%

Step 2:

Estimate the following metrics for each of the service components used in the Event Process. Define a logical time period to collect the metrics, so that the standard deviation is not too high for the averages computed. It could per hour, per minute or per day depending on the customer usage pattern. Let us refer to this time period as MC_TIME_INTERVAL in the notes below.

1. Average number of messages [ANM] per MC_TIME_INTERVAL during peak usage hours.
2. Maximum number of messages [MNM] per MC_TIME_INTERVAL during peak usage hours.
3. Average size of the messages. [AVG]
4. Maximum message size. [MAX]

When flow components like Aggregator, XMLSplitter, and Content Based Router are used, metrics such as the Average number of messages and maximum number of messages vary from component to component depending on the position they are in the application flow. When using other components that expect Input in one format and send output in another format, metrics like average message size and maximum message size tend to vary.

The number of messages for a particular service component can be found by applying the debugger (message interceptor) on the incoming routes on all the incoming ports. The debugger shows the count of messages coming into the component. The message size has to be determined by applying the debugger and copying the XML content (or binary content) into a temp file.

It is ideal to tune the application for the average settings, provided the standard deviation is not too high. The maximum memory settings should be high enough to accommodate stressful uses of the application.

For this particular sample walk-through, let us define these parameters as:

MC_TIME_INTERVAL = 1 minute

Parameter	FilePoller	XMLConverter	AggregateInvoices	TranformWCRequest	Customer Service Lookup
ANM	10	10	10	3	3
MNM	30	30	30	10	10
AVG	1KB	1KB	5KB	500KB	30KB
MAX	2KB	2KB	10KB	1MB	100KB

Parameter	Insurance Rules Engine	ContentBasedRouter	StoreResults	CreateWebPage
ANM	3	3	2	1
MNM	10	10	7	3
AVG	300KB	100KB	100KB	500KB
MAX	500KB	500KB	500KB	500KB

With the above metrics, you can calculate the operation Frequency from the number of messages given in the MC_TIME_INTERVAL.

Step 3:

Once you have the above metrics, if any of the recommendations on the specific component provided by Fiorano fits the scenario under which the component is being used, it is possible to use these recommendations; for others, you can manually tune the memory settings for the component as described earlier in the documentation, observing the GC behavior and the Memory allocation graph.

Component Name	Type	Max Recommended Heap size	Min Recommended Heap size
FilePoller	File Reader	32	4
XMLConvertor	Text2XML	32	4
AggregateInvoices	Aggregator	64	16
TransformWCRequest	XSLT	128	32
Customer ServiceLookup	WS Invoker	64	16
InsuranceRulesEngine	DB	128	32
ContentBasedRouter	CBR	64	16
StoreResults	DB	128	32
CreateWebPage	FileWriter	64	32
		704 MB	184 MB

Recommended Load on the test system:

85% of the physical available memory which is 1740 MB.

7.6 Memory Management of Fiorano Peer Server

Memory management is a process of observing the Server behavior and tuning the JVM hotspot parameters so that the Fiorano Peer Server is be able to perform at its highest throughput with the lowest risk of running into Memory related problems. This is not an optional step to just improve performance; memory tuning and management has to be done at the server to avoid a class of serious problems that could occur because of improper configurations of the Virtual Machine (VM) on which the Server is running. Problems of this nature are very hard to reproduce and can create a lot of confusion with regards to the server behavior.

For example, if one of the "Stop the world garbage collection" calls stops or pauses the Peer VM for more than the time the Enterprise Server waits for a response to an action command, the request times out and there will be an error thrown in Fiorano Studio. If the Primary Peer VM is paused for an interval greater than the ping interval, the backup HA Server might think the primary server is down which can lead to problems.

Tuning the memory is also very important for achieving throughput goals. A badly tuned server may spend almost all of its time doing garbage collection sparing only a few CPU cycles for the server execution.

The following sections describe the factors considered when the Peer Server VM is tuned.

7.6.1 Physical Machine Configuration

One of the most important factors in tuning the Peer VM is the actual machine hardware configuration. The amount of physical memory, the number of CPUs or cores, the Operating System, type of the system (32-bit or 64-bit), and so on, all have a bearing on performance. The server can afford an efficient concurrent garbage collection if it has more Processor cycles to spare. A few GC algorithms run best on a single core and while others have to be run only on multi-core machines. The native process stack size can be reduced for some hardware and OS combinations while for others it is not safe to do so. If the heap space allocated to the peer is much larger than the actual physical memory, a lot of swap occurs between the cache and memory, severely degrading the server performance. A larger memory allocated for the server would also mean a larger pause time for the garbage collector to do a full system cleanup.

7.6.2 Java Virtual Machine

The version of JVM used and the type of JVM (32-bit or 64-bit) is probably the next most important factor in the tuning of parameters for the server VM. A newer version of the JVM may have new Garbage collection algorithms implemented and the existing algorithms can be better tuned or may behave differently in certain cases. A server running with a 64-bit JVM can take more heap space than one running with a 32-bit JVM. More heap memory can be allocated to the server with 64-bit JVM than one with a 32-bit JVM which has a physical limit of 4 GB on the Process Address Space.

7.6.3 Machine Setup

Although it is preferable to have a dedicated server setup for each Fiorano Peer Server, but due to hardware limitations, this often is not possible. Other programs may have to run simultaneously with the Peer Server which competes for available CPU cycles and available memory. The Peer Server has to be tuned differently depending on the number of other programs running on the deployment machine.

7.6.4 Processing Message Size

This is one of the most important factors deciding the amount of memory required by the Peer VM. The Peer VM is a JMS broker which transfers messages from the components to other components. The memory required by the Peer Server VM depends on JMS context settings like persistence, in-memory buffer size, number of connections/sessions/ created and more importantly the message sizes handled. The components running in-memory can require additional memory typically equivalent to four or five times the size of the message to parse the XML and process it.

7.6.5 Peer Server Load

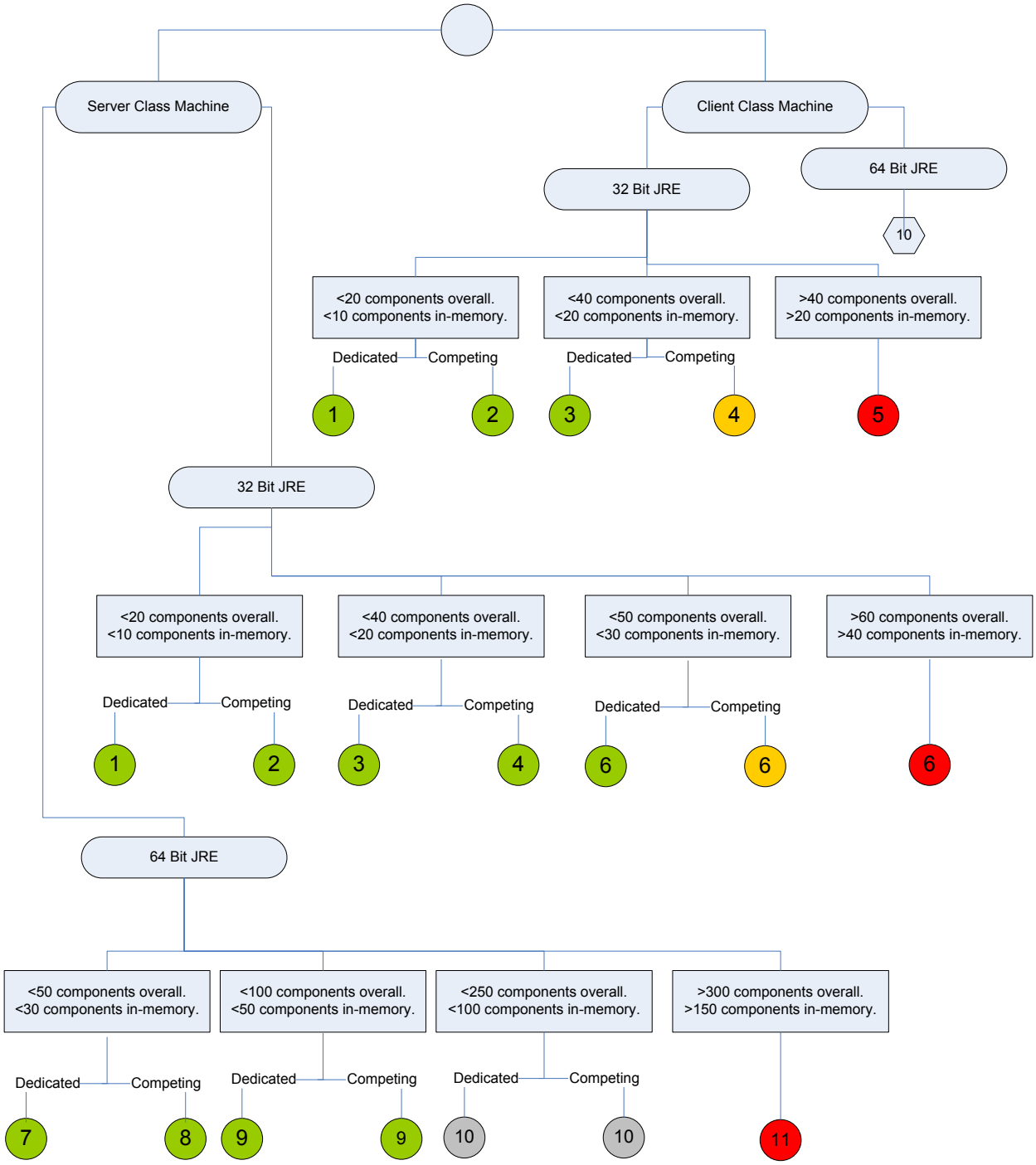
The Peer Server load is the number of components which are launched by it. Components can be launched in-memory - which means the component runs in the same VM as the Peer Server. For each component, a set of JMS resources are created which consume Heap Memory and a certain number of threads are created per JMS session spawned. For components launched in-memory, the classes loaded for the components to be launched affect the VM perm gen space as well as the memory required by the component to run.

7.6.6 Recommendations

The following tree gives you a guideline of the JVM hotspot settings for a set of combinations for the above factors. These recommendations are only guidelines and not stead-fast rules. It is recommended to tune the server starting with one of the recommended settings which match closely with the setup. These recommendations have consistent system behavior as their top priority, followed by performance.

These recommendations do not incorporate the dimension associated with processing messages of a certain size. These recommendations assume that the message sizes are below 1 MB and will not actually affect the memory required by the Peer Server to process the messages.

Note: The JVM hotspot setting is recommended below especially for those which starts with '-XX' keyword, are non-standard hotspot options and are not guaranteed to be supported by all JAVA VM implementations. For example, IBM JRE does not support many of the these options. The recommendations below have been made assuming that Sun JRE is being used to run the servers. In case some other JAVA VM is used, users are advised to find out corresponding options as provided by VM vendor.



Each leaf in the path of the tree represents a recommended setting and the color of the node indicates the risk level associated with the particular set up.

●	Green represents Low Risk and falls into recommended methods

	Yellow represents Medium Risk and has to be used under caution
	Red represents High Risk and the system is highly unstable.
	Grey represents Low Risk with other factors assumed. The corresponding recommendations list the assumptions on which this represents Low Risk.

7.6.7 Interpreting and Applying Recommendations

Each recommendation is identified by a unique number written over the leaf circle in the above tree. The recommended settings are listed below along with their description and a rationale as to why the setting is recommended. To apply these recommendations, find the tuning section in the server.conf (or fes.conf / fps.conf) and just add the texts found in the property names in that section.

{FIORANO_HOME}/esb/server/bin/server.conf
{FIORANO_HOME}/esb/fes/bin/fes.conf
{FIORANO_HOME}/esb/fps/bin/fps.conf

Table 1

Property Name	Description	Rationale
`-Xms512m`	Minimum Heap size	Dedicated servers can have the minimum and maximum Heap size to save the JVM from allocating and reallocating memory thereby increasing throughput.
`-Xmx512m`	Maximum Heap size	
`-XX:+UseParallelOldGC`	Garbage Collection Algorithm	Parallel GC reduces pause time and works well on small heap spaces with not many processors to spare.
`-XX:MaxGCPauseMillis=100000`	JVM Pause limit during GC	Limit the amount of time the garbage collector can pause the VM to be lower than the ESB call timeout (def: 120 seconds).

Table 2

Property Name	Description	Rationale
`-Xms128m`	Minimum Heap size	Competing servers must have

-Xmx512m	Maximum Heap size	the lowest Xms which is safe for the server to execute. The VM is responsible for allocating more memory upto the maximum memory as and when required and shrink back when needed.
-XX:+UseParellelOldGC	Garbage Collection Algorithm	Parallel GC reduces pause time and works well on small heap spaces with not many processors to spare
-XX:MaxGCPauseMillis=100000	JVM Pause limit during GC	Limit the amount of time the garbage collector can pause the VM to be lower than the ESB call timeout (def: 120 seconds).

Table 3

Property Name	Description	Rationale
-Xms1024m	Minimum Heap size	Dedicated servers can have the minimum and maximum Heap size to save the JVM from allocating and reallocating memory, thereby increasing throughput.
-Xmx1024m	Maximum Heap size	
-XX:+UseConcMarkSweepGC	Garbage Collection Algorithm	Works well for long running servers with a large heap memory that can afford to share CPU cycles with the garbage collector. Would result in the lowest pause times.
-XX:MaxGCPauseMillis=100000	JVM Pause limit during GC	Limit the amount of time the garbage collector can pause the VM to be lower than the ESB call timeout (def: 120 seconds).
-XX:+CMSIncrementalMode	Optimization option	This optimizes the algorithm when it is run with 2 or less cores.
-XX:MaxPermSize=128	Perm Gen size	When the number of components launched in-memory increases, so do the number of classes loaded. Perm Gen space stores the

		meta data of the JVM and is not part of the Heap space.

Table 4

Property Name	Description	Rationale
-Xms512m	Minimum Heap size	Competing servers must have the lowest Xms which is safe for the server to execute. The VM is responsible for allocating more memory upto the max memory limit and shrink back when needed.
-Xmx1024m	Maximum Heap size	
-XX:+UseParellelOldGC	Garbage Collection Algorithm	Works with constrained amounts of CPU cycles and limits the pauses under the specified limit by a parallel collection.
-XX:MaxGCPauseMillis=100000	JVM Pause limit during GC	Limit the amount of time the garbage collector can pause the VM to be lower than the ESB call timeout (def: 120 seconds).
-XX:MaxPermSize=128	Perm Gen size	When the number of components launched in-memory increases, so do the number of classes loaded. Perm Gen space stores the meta data of the JVM and is not part of the Heap space.

Table 5

Property Name	Description	Rationale
-Xms1024m	Minimum Heap size	Maximum heap space of 1280 is recommended. More memory allocated to the heap would suffocate other process memory regions and the number of threads that can be created would be dangerously low if this limit is exceeded.
-Xmx1280m	Maximum Heap size	
-XX:+UseParellelOldGC	Garbage Collection	Works with constrained

Property Name	Description	Rationale
	Algorithm	amounts of CPU cycles and limits the pauses under the specified limit by a parallel collection.
-XX:MaxGCPauseMillis=100000	JVM Pause limit during GC	Limit the amount of time the garbage collector can pause the VM to be lower than the ESB call timeout (def: 120 seconds).
-XX:MaxPermSize=128	Perm Gen size	When the number of components launched in-memory increases, so do the number of classes loaded. Perm Gen space stores the meta data of the JVM and is not part of the Heap space.

Table 6

Property Name	Description	Rationale
-Xms1280m	Minimum Heap size	Maximum Heap space of 1280 is recommended. More memory allocated to the heap would suffocate other process memory regions and the number of threads that can be created would be dangerously low if this limit is exceeded.
-Xmx1280m	Maximum Heap size	
-XX:+UseConcMarkSweepGC	Garbage Collection Algorithm	Works well for long running servers with a large heap memory that can afford to share CPU cycles with the garbage collector. Would result in the lowest pause times.
-XX:MaxGCPauseMillis=100000	JVM Pause limit during GC	Limit the amount of time the garbage collector can pause the VM to be lower than the ESB call timeout (def: 120 seconds).

-XX:MaxPermSize=256	Perm Gen Space	When the number of components launched in-memory increases, so do the number of classes loaded. Perm Gen space stores the meta data of the JVM and is not part of the Heap space.

Table 7

Property Name	Description	Rationale
-Xms1536m	Minimum Heap size	With 64-bit JVMs, the limit of 4GB Address space per process is removed and hence a large Heap memory can be allocated to a process without any side-effects.
-Xmx1536m	Maximum Heap size	
-XX:+UseConcMarkSweepGC	Garbage Collection Algorithm	Works well for long running servers with a large heap memory that can afford to share CPU cycles with the garbage collector. Would result in the lowest pause times.
-XX:MaxGCPauseMillis=100000	JVM Pause limit during GC	Limit the amount of time the garbage collector can pause the VM to be lower than the ESB call timeout (def: 120 seconds).
-XX:MaxPermSize=256	Perm Gen Space	When the number of components launched in-memory increases, so do the number of classes loaded. Perm Gen space stores the metadata of the JVM and is not part of the Heap space.

Table 8

Property Name	Description	Rationale
-Xms512m	Minimum Heap size	With 64-bit JVMs, the limit of

-Xmx1536m	Maximum Heap size	4GB Address space per process is removed and hence a large heap memory can be allocated to a process without any side-effects.
-XX:+UseConcMarkSweepGC	Garbage Collection Algorithm	Works well for long running servers with a large heap memory that can afford to share CPU cycles with the garbage collector. Would result in the lowest pause times.
-XX:MaxGCPauseMillis=100000	JVM Pause limit during GC	Limit the amount of time the garbage collector can pause the VM to be lower than the ESB call timeout (def: 120 seconds).
-XX:MaxPermSize=256	Perm Gen Space	When the number of components launched in-memory increases, so do the number of classes loaded. Perm Gen space stores the metadata of the JVM and is not part of the Heap space.
-XX:ParallelGCThreads=<no_of_cpu / no_of_competing_servers>	Option to limit monopolizing CPU resources	If no single application is to monopolize the CPU for a long period of time, it is recommended to limit the number of parallel threads which can be spawned.

Table 9

Property Name	Description	Rationale
-Xms1024m	Minimum Heap size	With 64-bit JVMs, the limit of 4GB Address space per process is removed and hence a large heap memory can be allocated to a process without any side-effects.
-Xmx2048m	Maximum Heap size	
-XX:+UseConcMarkSweepGC	Garbage Collection Algorithm	Works well for long running servers with a large heap memory that can afford to share CPU cycles with the garbage collector. Would result in the lowest pause times.

-XX:MaxGCPauseMillis=100000	JVM Pause limit during GC	Limit the amount of time the garbage collector can pause the VM to be lower than the ESB call timeout (def: 120 seconds).
-XX:MaxPermSize=256	Perm Gen Space	When the number of components launched in-memory increases, so do the number of classes loaded. Perm Gen space stores the metadata of the JVM and is not part of the Heap space.
-XX:ParallelGCThreads=<no_of_cpu / no_of_competing_servers>	Option to limit monopolizing CPU resources	If no single application is to monopolize the CPU for a long period of time, it is recommended to limit the number of parallel threads which can be spawned.

Table 10

Property Name	Description	Rationale
-Xms1536m	Minimum Heap size	With 64-bit JVMs, the limit of 4GB Address space per process is removed and hence a large Heap memory can be allocated to a process without any side-effects.
-Xmx3072m	Maximum Heap size	
-XX:+UseConcMarkSweepGC	Garbage Collection Algorithm	Works well for long running servers with a large Heap memory that can afford to share CPU cycles with the garbage collector. Would result in the lowest pause times.
-XX:MaxGCPauseMillis=100000	JVM Pause limit during GC	Limit the amount of time the garbage collector can pause the VM to be lower than the ESB call timeout (def: 120 seconds).

-XX:MaxPermSize=512	Perm Gen Space	When the number of components launched in-memory increases, so do the number of classes loaded. Perm Gen space stores the meta data of the JVM and is not part of the Heap space.
-XX:DisableExplicitGC	Diabling explicit GC requests	It is dangerous for a misbehaving in-memory component to continuously issue System.GC calls when the Heap memory allocated is large. This option will disable actions against explicit calls. The memory of the server is best managed by the VM. Garbage collection calls from within software code can be extremely dangerous.
-XX:ParallelGCThreads=<no_of_cpu / no_of_competing_servers>	Option to limit monopolizing CPU resources	If no single application is to monopolize the CPU for a long period of time, it is recommended to limit the number of parallel threads which can be spawned.
Assumption: The Physical RAM is at least 8 GB. However 16 GB is recommended.		

Table 11

Property Name	Description	Rationale
-Xms2048m	Minimum Heap size	With 64-bit JVMs, the limit of 4GB Address space per process is removed and hence a large Heap memory can be allocated to a process without any side-effects.
-Xmx3072m	Maximum Heap size	
-XX:+UseConcMarkSweepGC	Garbage Collection Algorithm	Works well for long running servers with a large Heap memory that can afford to share CPU cycles with the garbage collector. Would result in the lowest pause times.
-XX:MaxGCPauseMillis=100000	JVM Pause limit during	Limit the amount of time the

	GC	garbage collector can pause the VM to be lower than the ESB call timeout (def: 120 seconds).
`-XX:MaxPermSize=512`	Perm Gen Space	When the number of components launched in-memory increases, so do the number of classes loaded. Perm Gen space stores the metadata of the JVM and is not part of the Heap space.
`-XX:DisableExplicitGC`	Disabling explicit GC requests	It is dangerous for a misbehaving in-memory component to continuously issue System GC calls when the Heap memory allocated is large. This option will disable actions against explicit calls. The memory of the server is best managed by the VM. Garbage collection calls from within software code can be extremely dangerous.
`-XX:ParallelGCThreads=<no_of_cpu / no_of_competing_servers>`	Option to limit monopolizing CPU resources	If no single application is to monopolize the CPU for a long period of time, it is recommended to limit the number of parallel threads which can be spawned.

7.6.8 Handling Memory Problems

One common issue that Users face is the server JVM exiting with a *java.lang.OutOfMemoryError* exception. This error is thrown when there is insufficient space to allocate an object. That is, garbage collection cannot make any further space available to accommodate the new object, and the heap cannot be expanded further. An *OutOfMemoryError* does not necessarily imply a memory leak; the issue might simply be a case of allocating more memory for the server to perform its operations.

The first step in diagnosing an OutOfMemoryError is to examine the full error message. Generally, in the exception message, additional information is supplied which hints at a reason as to why the JVM ran out of Memory. The following contains a list of some common examples of what that additional information may be, what it may mean, and what to do about it.

7.6.7 Java Heap Space

This indicates that an object could not be allocated on the heap. This issue may just be a configuration problem related to assigning more Heap space to the server. This can be done so using the –Xmx option in the server.conf file. Following the recommendations above, if enough memory has been assigned to the server and the server still runs out of Java Heap space, then there could be a memory leak at the server end given that the possibility of an in-memory running component causing the damage is ruled out. Custom components should be checked for any memory leak with profiling tools available before they are run in-memory.

7.6.7.1 Debugging

In this case, a Reproducible set of steps can be provided to the Fiorano tech support and if its not reproducible every time, the GC Logging can be enabled and the logs can be sent for analysis. This can also indicate that a profiling tool analysis is required at the server. Profiling tools help in monitoring the number of objects pending finalization and to view all reachable objects while understing which references are keeping each one alive.

7.6.8 Permgen Space

This indicates that the permanent generation is full. This is the area where the JVM stores its meta-data, data about classes loaded and threads running. If the Peer Server runs a lot of components in-memory then the perm gen space needs to be increased. Please follow the recommendations to find a suitable value for the permgen space.

7.6.8.1 Debugging

Increasing the perm gen space generally solves this problem since there cannot be any memory leak in this region as it is not available to the programs. However, if the server misbehaves in loading a huge number of classes when it launches components in-memory, there could be a problem. Using a Java profiling tool to determine the number of classes loaded by the Peer Server and the components running in-memory should solve this situation.

7.6.9 Requested Array size exceeds VM Limit

This indicates that the application attempted to allocate a continuous array whose size is larger than the available contiguous memory in the heap. In most cases the problem is likely to be either that the heap size is too small or set incorrectly or that a bug results in the server attempting to create an array whose size is incorrectly calculated. If the Xmx is set to more than 256 MB and if this error occurs repeatedly, steps to solve the problem should be reported to the Fiorano team.

7.6.10 Swap Space

Another possibility is that the server might run out of Swap Space even though the Heap space may not be near to the specified Xmx value. There is a restriction on 32 bit machines where a process can only have up to 4 GB of addressable memory. Each java process have several segments of memory for the java memory and the native memory (memory used the operating system to run the process). Native code competes with the JVM to use the 4GB of addressable space in the application. Such problems generally occur when the heap sizes are specified to more than 1.6 GB on a 32 bit machine not allowing enough memory for the native code to run. A simple solution to this may be to reduce the Xmx setting to a lower value. Please follow the recommendations above for maximum memory guidelines. This problem can also be solved using a 64 bit JVM on a 64 bit server which has practically no restrictions on the reference-able address spaces for a Process.

7.6.11 Unable to Create New Native Thread

This means that there is not enough memory to create a new thread. This could be due to JVM heap, other native memory or Perm Gen memory sections taking up all the memory and leaving nothing for Stack Memory. It could also mean that the server has already spawned the maximum number of threads allowed by the Operating System, though this is rare. If the former is the reason, it is generally solved by reducing the heap size and/or PermGen space. If this solution does not solve the problem, a last option to solve the problem is to try to optimize the stack size of a thread.

The stack size can be set using –Xss. If the threads complain that the stack size is too small then this number needs to be increased and the operation retried. Setting the stack size to half of the original would allow twice the number of threads to be created and this may solve the problem. If the above suggestions do not solve the problem, then the thread count, the no of components running in-memory, no of JMS clients connected and the frequency of messages should be observed and reported.

7.6.12 Enabling GC logging

One of the easiest ways to get initial information about garbage collections is to specify the options –XX:PrintGCDetails, -XX:PrintGCTimeStamps. For every collection, this option will result in the output of the information such as the size of live objects, before and after garbage collection for various generations, the total available space for each generation and the length of time the collection took. This also outputs a timestamp at the start of each collection which helps in correlating GC logs with the Server logs.

The verbose logging will be present in the server logs and in the output console of the server. This console can be sent to the Fiorano team for further analysis.

7.7 In-Memory Execution and Load Balancing of Components Across Peer Servers

This section provides guidelines on the load balancing of components across Peer Servers together with recommendations on in-memory execution. The number of components running on a Peer Server must be optimized to optimize the load on that Peer Server and to utilize machine resources effectively. Both the Separate Process and In-memory Launch options are discussed below.

7.7.1 Separate Process

The number of components that can be launched on a Peer Server depends on the RAM availability. Users can launch more components on a machine with greater RAM than on one with less RAM.

However, there is a practical limit on the number of components that can be safely launched on a Peer Server since all the components are launched as child processes of the Peer Server process. As the number of components on a particular Peer Server increases, it becomes difficult for the Peer Server to launch and stop the components. This can result in Request Timeout exceptions.

To avoid these timeout exceptions there is a need to distribute components across different Peer Servers.

The default timeout for application launch/stop in the Fiorano Studio is 100 seconds. Any server call is timed out after this duration. This value is configurable and can be increased to avoid frequent timeouts.

Fiorano recommends launching 60-75 components per Peer Server on a machine with 8 GB of RAM. The limit can be slightly greater on a machine with more RAM.

7.7.2 In-memory

In case of In-memory Launch, components is launched with the JVM of the Peer Server of the local machine. The default Heap memory allocated for servers in the Fiorano environment is 512 MB. Server heap memory can be increased by changing the –Xmx parameter value by editing the server.conf file located at FIORANO_HOME\esb\server\bin. For In-memory Launch, it is recommended that the server Heap memory should be increase to the maximum limit, though this has the affect of allowing less threads within the overall Peer Server process.

In case of 32-bit machines, a java process can span a maximum of 1.6 GB of memory. Assuming that the heap-size is set to this 1.6 GB limit, the number of In-memory components can be increased until the Peer server's memory usage reaches this point (or within safe limits of this point, say approximately 70% of 1.6 GB).

In case of 64-bit machines, the java heap memory is relatively high (approximately 4 GB) and a larger number off components can be launched in In-memory. However, one needs to be careful in assigning more memory to the Peer Server process since more Heap memory has an impact on the number of threads that can be created by the JVM. As the heap memory of a process increase, the number of threads that can be created is less; this can have an adverse affect on Peer Server performance. As such, the precise heap-memory allocated to the Peer Server JVM has to be determined by testing that particular configuration being run (that is, by running the full application to ensure there are no adverse affects on the system).

Note:

- Components that are launched In-memory should preferably have a lower memory footprint and lower CPU utilization. Components that have a large memory footprint or which take up greater amounts of CPU usage should not be launched In-memory as they will have a negative impact on the Peer Server functionality. For instance, if a component launched in a separate process continuously consumes, say, 70% of CPU then it cannot be launched In-memory.
- Fiorano suggests keeping the components as Separate Processes unless there's an absolute need to change the launch mode to In-memory. Running components In-memory dramatically decreases the overall memory utilization of an application since the number of JVM's launched is decreased. Precise recommendations about which components from the palette can be safely run In-memory are discussed in a separate document.

Chapter 8: Security

This chapter discusses the security policies and service-governance features supported by the Fiorano SOA Platform.

The Fiorano SOA Platform security policy interface gives you the necessary control to administer and manage groups and users on the entire Fiorano Network as well as the ability to control process deployment to QA, Staging and Production environments.

8.1 Authentication

The Fiorano SOA Platform users and groups can operate from all available nodes in the Fiorano Network. The Fiorano ESB Dashboard can be used to configure and manage user authentication.

A group is identified by a unique name and contains a list of users who inherit all rights assigned to a group. Each user is assigned a unique user name, password, and a group membership. Information pertaining to users and groups are utilized while authenticating and enables to determine the resources of a user or a group that is allowed to access.

The Fiorano ESB Dashboard allows you to manage all principals in a Fiorano Network. To manage principals, you need to view the list of principals in the right pane of Fiorano ESB Dashboard. To view the list of principals, click the Groups node in the left pane. Figure 8.1.1 illustrates the principals list.

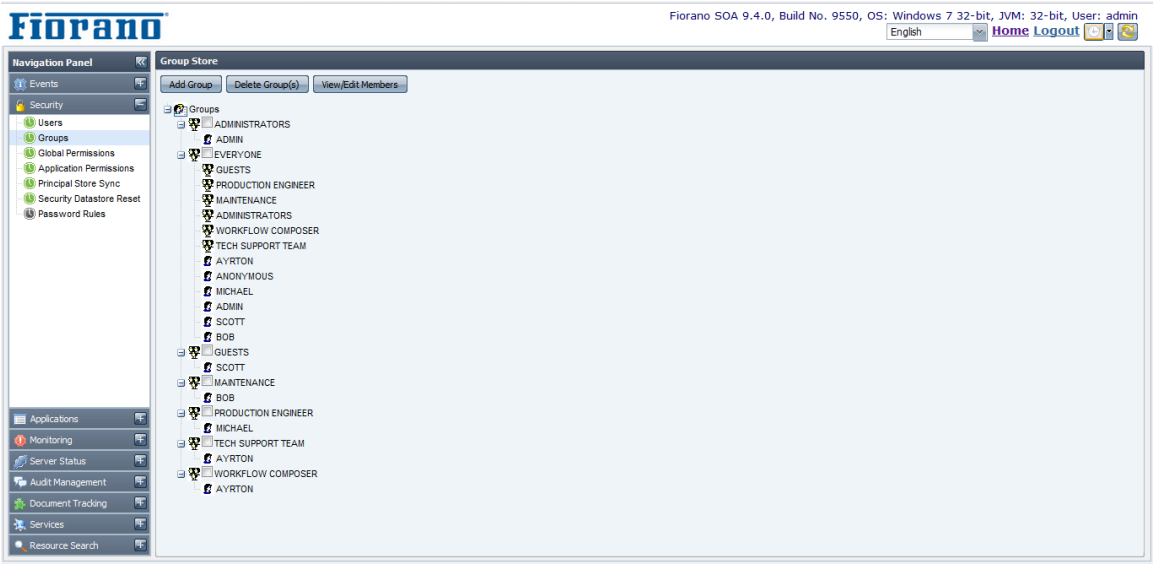


Figure 8.1.1: Users List

You can perform the following management tasks:

- Creating user accounts
- Deleting user accounts

- Changing the password of a user

Similarly the Groups node allows the administrator to club users into specific groups and provide appropriate levels of authorization.

8.2 Authorization

The ESB Dashboard allows you to assign rights to users and groups, rights may be understood as rules associated with the Fiorano Network granted to users and groups. Rules allow users and groups to perform specific tasks on a Fiorano Network. The Fiorano SOA Platform has a well-defined security policy to protect your network against data loss or corruption due to malicious or accidental access. This policy is implemented by assigning the appropriate permissions to groups as well as their users.

When you select Global Permissions shown in the left panel, a list of all available permissions are displayed in the right panel as shown in the Figure 8.2.1.

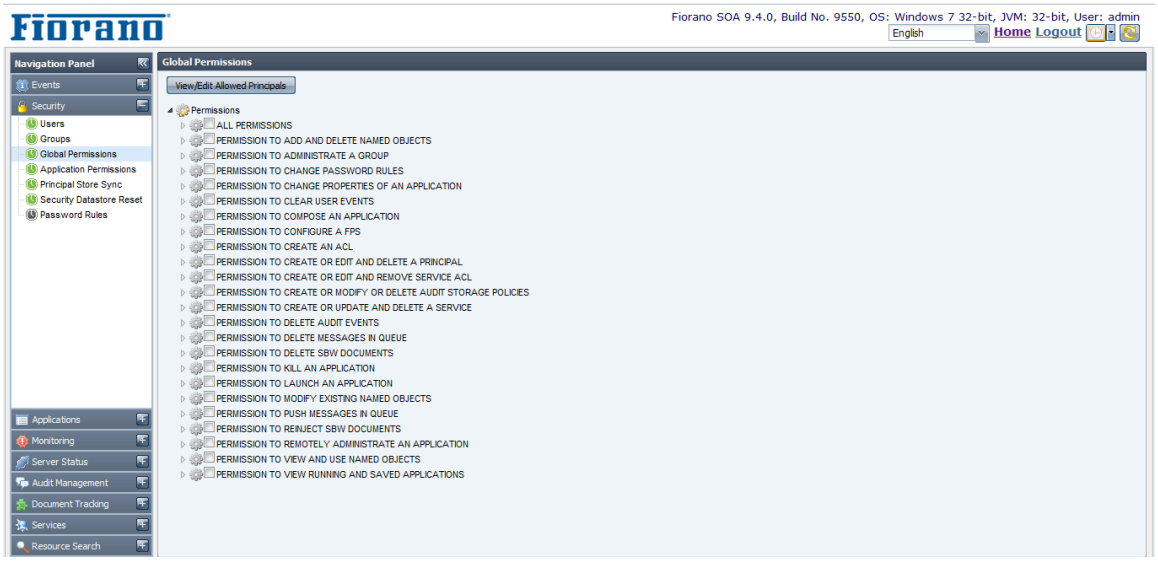


Figure 8.2.1: Access Right Assignment

To set permission for any user and group, select a permission entry and click on View/Edit Allowed Principals to configure. Figure 8.2.2 illustrates the Permission to Create an ACL is restricted to Administrators. Other users and groups can be authorized to perform this operation by adding to the Assigned To list.

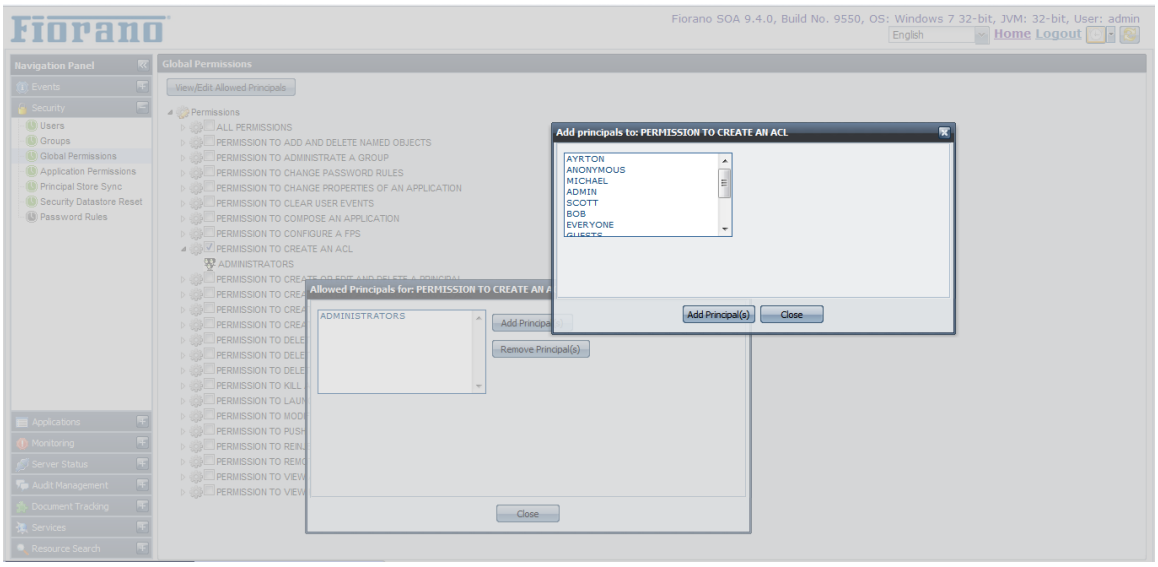


Figure8.2.2: Permission to Create an ACL

8.3. Password Rules

For security purpose, Fiorano Enterprise Server provides feature to define the password validity and strength check rules for accepting password while adding a new user or changing the password of existing user.

8.3.1 Password Strength

Strength of a password is defined on a logical scale and suggests the user that how secure is the password provided by him. Password Strength can be categorized into one of the following categories:

- VERY WEAK
- WEAK
- AVERAGE
- ABOVE AVERAGE
- STRONG
- VERY STRONG

8.3.2 Password Validation

Whereas Password Strength Check defines guidelines for choosing a good password, Password Validation checks put restrictions on which passwords are valid and which ones are invalid. Invalid passwords will not be accepted by the system and the user will be prompted to provide a new password.

Password strength and validation rules are defined per profile basis, that is, the rules can be changed per Enterprise Server profile.

Note: To know more about how password validation and strength check rules can be added or deleted refer to section 2.5.1.2 Security.

8.3.3 Custom Password Rule Implementation

Fiorano Enterprise Server provides the user the flexibility to override the default implementation of password rules and write his own custom implementation, that is, the user can write his own Java Class for checking both password strength and validity.

Steps to define custom implementation for checking password strength and validity

Users who wish to override the default implementation and provide their own custom implementation should implement the following interface:

- "IStrengthChecker.java" for password strength check
- "IValidationChecker.java" for password validation check

Both of these interfaces are defined in package "fiorano.tifosi.dmi.rule.passwd.api". For more information please refer to the javadoc of these APIs in *$FioranoHome/javadoc/esb/dmi/fiorano/tifosi/dmi/rule/passwd/api/*

After defining your own custom implementation for one or both of password strength and validation check, please follow the below mentioned steps to configure Enterprise Server for overriding the default implementation with your custom implementation:

1. Compile the custom class/classes and create a jar of them. Refer link http://java.sun.com/docs/books/tutorial/deployment/jar/build.html to see how jar files are created.
2. Include the jar in classpath of Enterprise Server by adding the full path of jar in server.conf or fes.conf (depending on you are using server.bat/server.sh or fes.bat/fes.sh for starting Enterprise Server) under the tag <java.classpath> in conf file.
3. Start nStudio, and open FES profile by navigating to Profile Management -> Fiorano SOA -> <*EnterpriseServerProfile*> -> fes. In FES Profile, navigate to Fiorano->ESB->Security->User->UserSecurityManager as shown in figure below:

Figure: UserSecurityManager option

4. Click on the ellipsis of "**PasswdStrengthImpl**" in the **Properties of UserSecurityManager** panel on the right hand side of Studio as shown in figure below, provide the fully qualified classname (<*package_name*>.<*class_name*>) of the custom password strength check class. Similarly for password validity check, click on the ellipsis of "**PasswdValidityImpl**" and provide the fully qualified classname of the custom password validity check class.

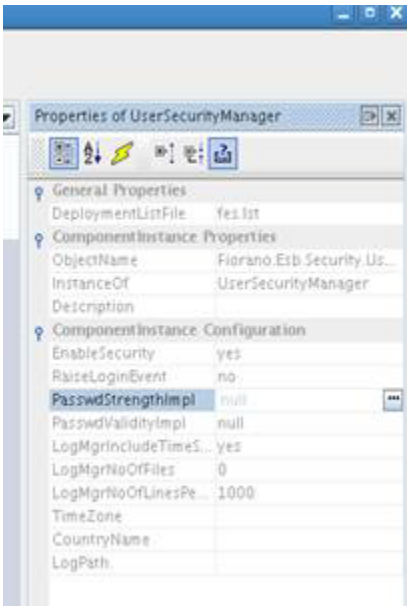


Figure: PasswdValidityImpl

5. Save the profile as shown in the figure below:

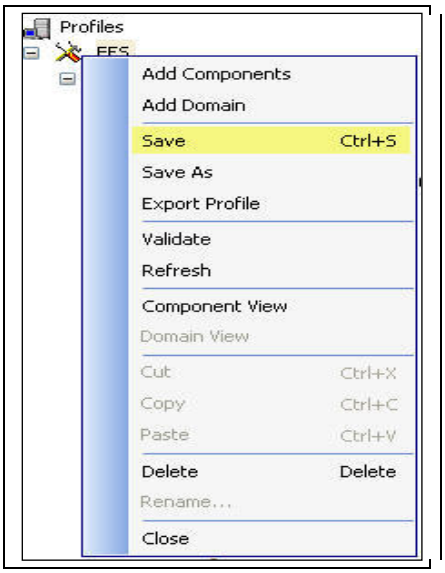


Figure: Saving the profile

Now, the enterprise server will use your custom classes for checking password strength and validity

Note: All the information described above regarding the password rules is valid only if the user is added through Dashboard. If Studio is used for adding new user, the password rules will not imply as the Studio by default sets the password for the new user same as the username.

8.4 Deployment Manager

The Management and Governance of the distributed applications is a challenging task. The On-demand business requirements mean that new services have to be frequently added to existing business processes. The administrator needs to ensure that the service components being added are tested thoroughly and do not compromise the stability and performance of the system. To satisfy this requirement, the administrator needs to implement reliable configuration management and governance policies to prevent unstable components from being added to the system.

Deployment Manager is a powerful rule engine that simplifies configuration management and governance in the Fiorano network. The Deployment Manager uses the concept of Labels and Rules to achieve the above mentioned goals.

8.5 Labels

The Fiorano SOA Platform enables users to label the various components of their distributed architecture. All the Service Components, Event Processes, and nodes (peer servers) can be labeled.

The labels currently supported by the tool are:

- Development
- QA
- Staging
- Production

You can use a combination of labels and other identifiers (GUIDs, version numbers, and node names) to create comprehensive and powerful rules to control the deployment of processes. For example, when a new process has been developed, then it can be marked with a **Development** label. After appropriate levels of testing, it can then be upgraded to **Staging**. The labeling support at the service component level provides similar functionality at a much more fine grained level.

8.6 Rules

A rule in the Fiorano SOA Platform context is a conditional statement that controls the launch of Business Components. The rules have two important aspects; identifiers and precedence.

The identifiers are values assigned to the various configurable elements of the Fiorano Network. The configurable elements are the fixed elements of the Fiorano Network, such as Event Processes, Business Components, and Peer Servers. The rules are constructed by combining identifiers of the various Fiorano SOA Platform elements. The Deployment Manager has been configured with the following identifiers:

- GUID
- Version
- Label

The identifiers allow you to control the ambit and complexity of a rule. The fewer the identifiers, the simpler the rule and the greater its ambit. For example, a simple rule can be created to disallow all instances of a particular Business Component from being launched. This rule will typically contain one identifier, the GUID of the Business Component. Despite being a simple rule, it applies to all Business Components in the Fiorano Network. A complex rule, on the other hand, can be created to prevent a particular Business Component from being launched on a Peer Server when it is part of a particular Event Process. This rule will contain at least three identifiers, the GUIDs of the Event Process, the Business Component, and the label of the Peer Server. The Version identifier is present only for components and event processes and not for the Peer Server.

Precedence: All rules are processed in the order in which they are stored by the Deployment Manager. By default, rules are stored in the order in which they were created. As a result, you will have to ensure that one rule does not interfere with the other. You can alter the precedence of the rules by using the controls provided in the tool.

The syntax of a rule is illustrates in the Figure 8.5.1.

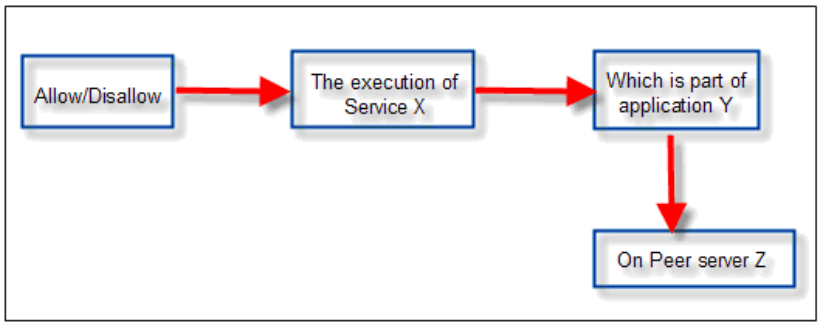


Figure 8.5.1: Rule syntax

Figure 8.5.2 illustrates a sample rule that disallows deployment of any non-production event process on a peer with the Production Label.

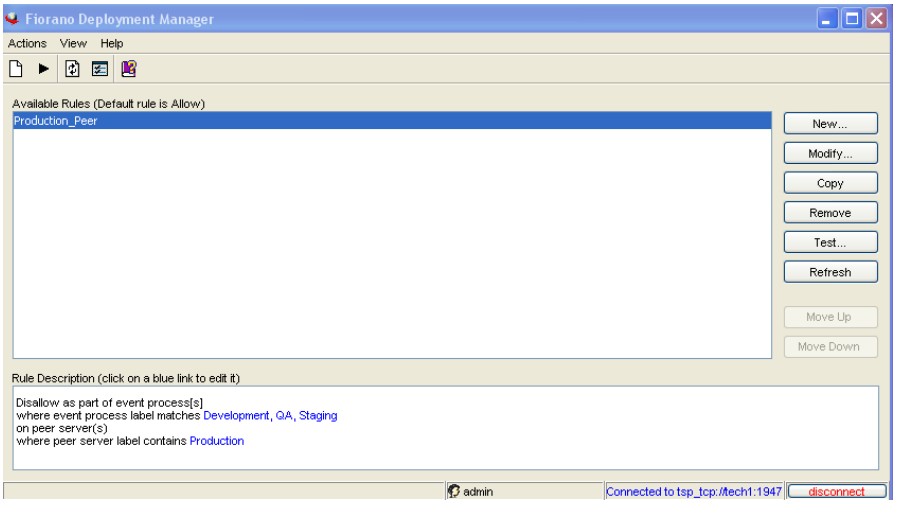


Figure 8.5.2: Sample Rule

The complex business rules can be tested and verified before implementing them in your environment. The Deployment Manager provides the ability to test the rules at design time as shown in Figure 8.5.3

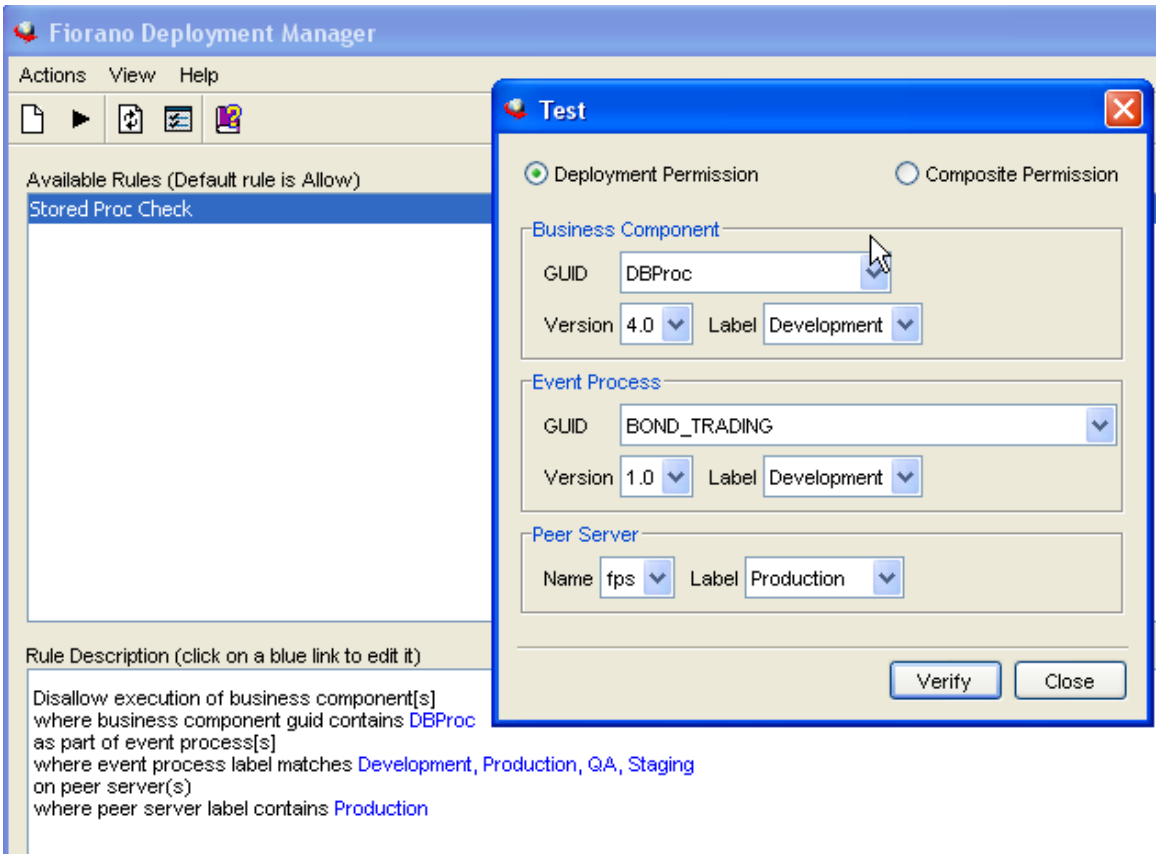


Figure 8.5.3: Verifying Rule

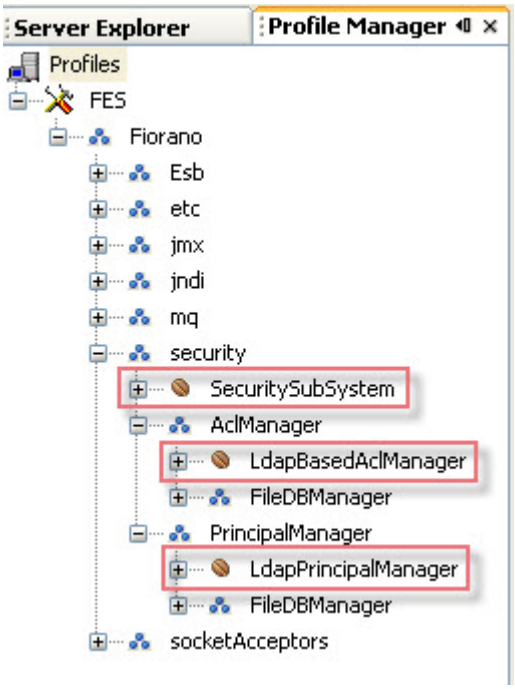
8.7 Changing Security Database Implementation

Security related information (user, groups and ACLs) is by default stored in file based data store inside runtimedata directory of the server. Using Fiorano Studio Profile Manager, the implementation can be changed before starting the server to LDAP, RDBMS or XML type storage. This section explains the configuration steps required to change the implementation.

8.7.1 Security Related MBeans

Security related components are found in default ESB profiles in **Fiorano->Security** domain. Object Names for these components are:

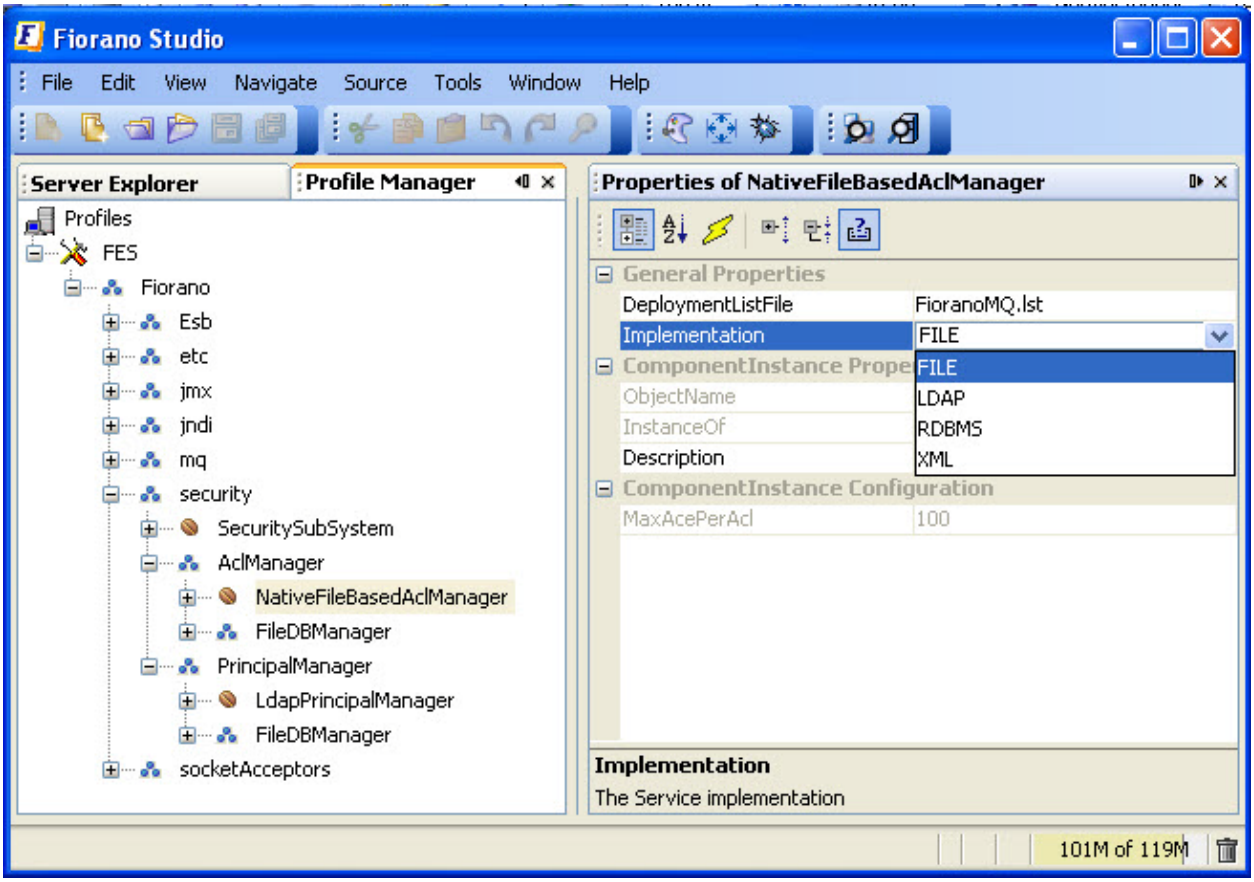
- Fiorano.security:ServiceType=RealmManager,Name=SecuritySubSystem
- Fiorano.security.AclManager:ServiceType=AclManager,Impl=FILE,Name=NativeFileBasedAclManager
- Fiorano.security.PrincipalManager:ServiceType=PrincipalManager,Impl=FILE,Name=NativeFilePrincipalManager



The figure above shows the position of these components in the component tree as seen from an off-line configuration tool (the **Profile Manager**).

8.7.2 Modifying ACLManager Implementation

1. Open the profile for off-line editing through the **Profile Manager**
2. Browse the tree to reach the node **Fiorano -> security -> AclManager**. Click on the current ACL Manager MBean.
3. In the properties panel, click on the value of the Implementation property and choose the desired value from the drop-down menu.

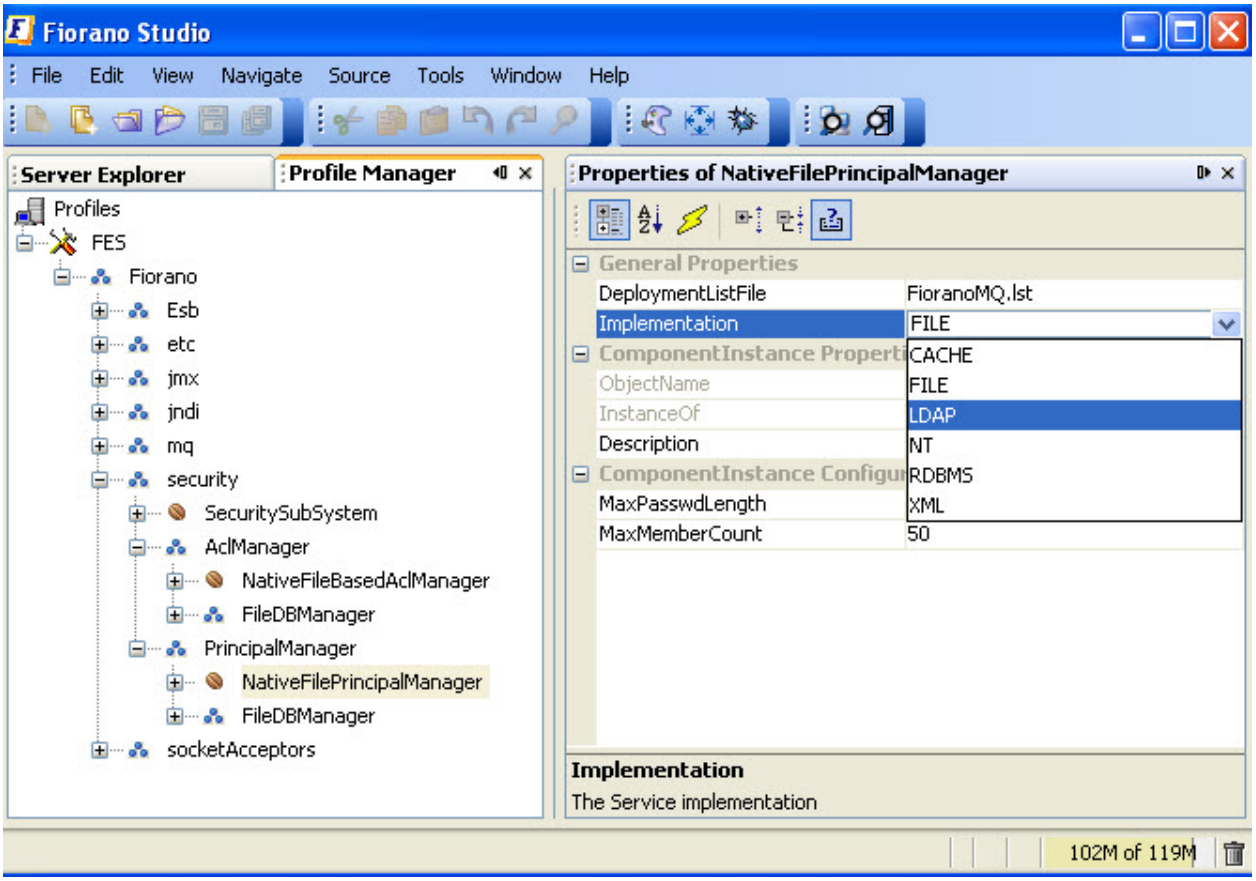


4. Right-click on the FES node and select **Save** from the pop-up menu.

8.7.3 Modifying Principal Manager Implementation

1. Open the desired profile for off-line editing through the **Profile Manager**.
2. Browse the tree to reach **Fiorano -> security -> PrincipalManager** and click on the current Principal Manager Mbean as shown in the figure below.

3. In the property panel click on the value of the Implementation property and choose a desired value from the drop down list.

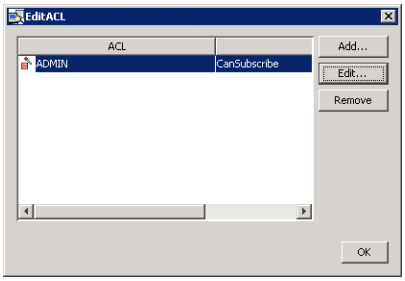


4. Right-click on the FES node and select **Save** from the pop-up menu

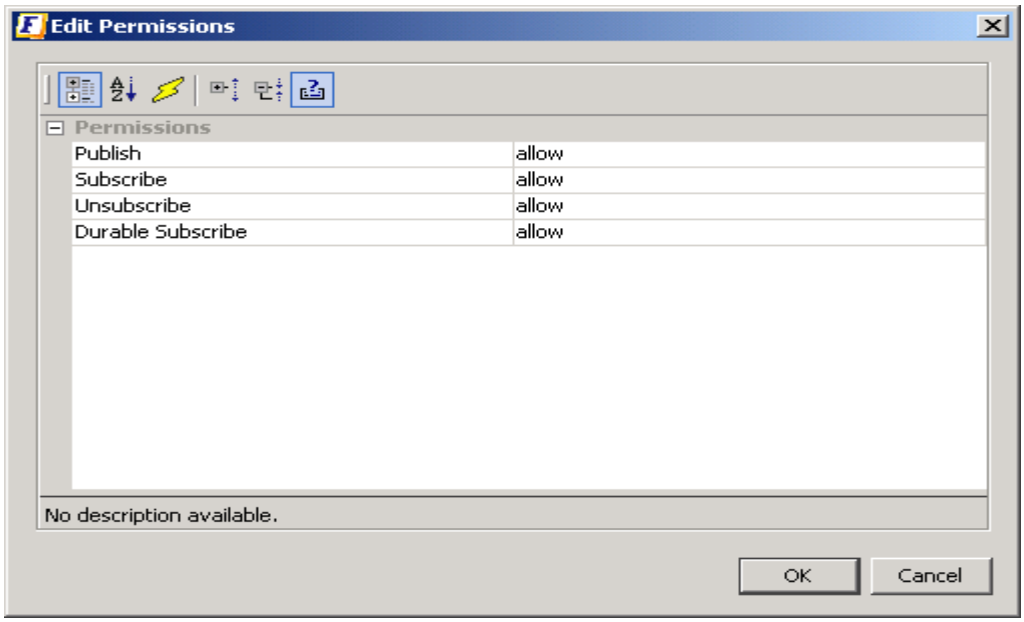
8.7.4 Editing Destination Level Security Through ACL's

The administrator can grant users access permissions to work on different destinations (Topics and queues). Permissions to users and groups can be edited by executing the following steps:

1. Launch Studio and click on the **Server Explorer** pane. Right-click on the desired server and select **Login** from the pop-up menu.
2. Navigate to the required destination through **Destinations > Topics/Queues**.
3. Right-click the required topic/queue and select the **EditACL** option from the pop-up menu. The **EditACL** dialog box is displayed as shown in the figure:



4. You can add permissions for a new principal by clicking on the **Add** button.
5. An existing entry for a principal can be removed by clicking on the **Remove** button.
6. Select the ACL Entry (for any principal) in this dialog box and click the **Edit** button. The **Edit Permissions** dialog box as shown below is displayed.

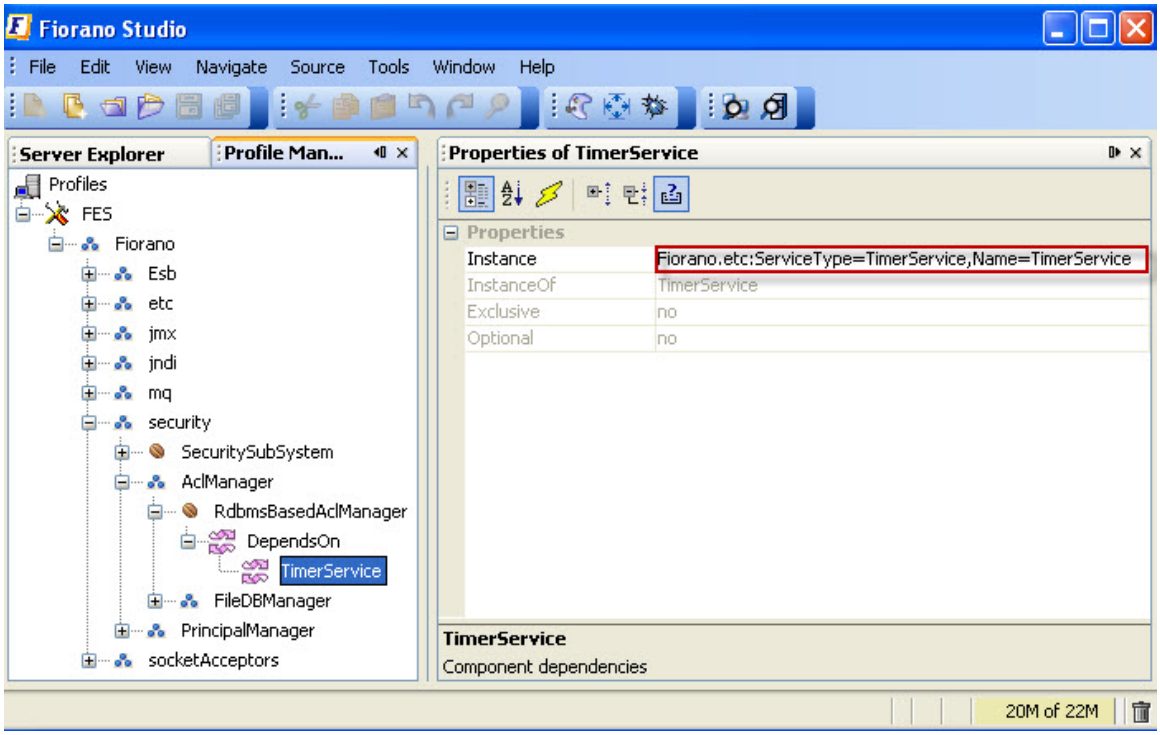


7. Modify the permissions for the various actions like Publish, Subscribe, Unsubscribe, and Durable Subscribe as desired and click on the **OK** button.

8.7.5 RDBMS Realm

8.7.5.1 Setting up

1. Open the profile for off-line editing through the **Profile Manager**
2. Modify Principal Manager [see section **7.4 Modifying Principal Manager Implementation of FioranoMQ Handbook**] and ACL Manager implementation [see section **7.5 Modifying ACLManager Implementation of FioranoMQ Handbook**] to RDBMS
3. Configure these components as per the desired database. Sample configuration for some common databases is provided in later sections.
4. RDMS based ACL manager depends on Timer Service. Specify the instance of TimerService as specified in figure below:



5. Right-click on the FES/FPS node and select **Save** from the pop-up menu.

8.7.5.2 Additional Configuration

The database driver has to be added to the Container classpath.

To force the Fiorano FES/FPS servers to create default destinations and users again over the newly configured RDBMS Server even if the server has already been started using any other configuration option, clear the existing database (run folder of profile) and then restart the server.

8.7.5.3 Sample Configurations

The list below provides sample configuration for various databases. These parameters can be specified for both ACLManager as well as PrincipalManager.

Oracle

URL: jdbc:oracle:thin:@164.164.128.113:1521:orcl

Database Driver: oracle.jdbc.driver.OracleDriver

Username: scott

Password: tiger

MySQL

URL: jdbc:mysql://localhost/mysql

DatabaseDriver: com.mysql.jdbc.Driver

Username: <user name>

Password: <password>

HSQL

URL: jdbc:hsqldb:d:\FMQDB

DatabaseDriver: org.hsqldb.jdbcDriver

Username: sa

Password: <password>

MSSQL

URL: jdbc:microsoft:sqlserver://qalab01:1433

DatabaseDriver: com.microsoft.jdbc.sqlserver.SQLServerDriver

Username: sa

Password: <password>

Note: The MS-SQL driver has to be added to the Container classpath(msutil.jar,mssqlserver.jar,msbase.jar)

DB2

URL: jdbc:db2://localhost:7777/sample

DatabaseDriver: COM.ibm.db7.6.jdbc.net.DB2Driver

Username: user

Password: passwd

The parameter named PropertiesFile should point to principalsqlstatements properties when configuring PrincipalManager and aclsqlstatements properties when configuring ACLManager. These files can be found in the conf folder of the desired profile.

8.7.5.4 Verifying

Use any Query tool provided by your database vendor, such as SQLWorksheet for Oracle, and verify the creation of the following tables with default values:

Principal Manager

TableName - users (stores all users related information)

TableName - groupmembers (stores all Group related information)

ACL Manager

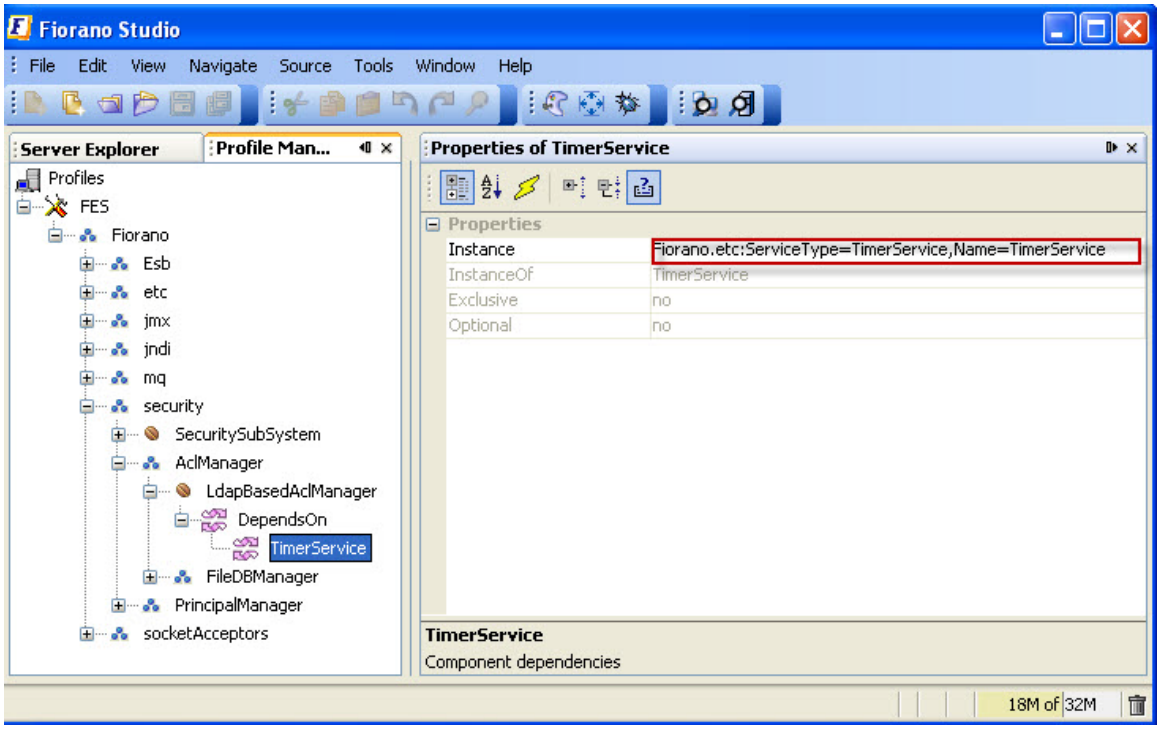
TableName - aclentries (stores ACL Entries for all users)

The following is a sample SQL query executed on SQLWorksheet:

- SQLWKS> select * from users;
- SQLWKS> select * from aclentries;

8.7.6 LDAP Security Realm

1. Open the profile for off-line editing through the **Profile Manager** using Studio by clicking on the **Profile Manager** pane.
2. Modify the Implementation property of ACL Manager and Principal Manager and to LDAP. For more information on how to modify ACL Manager and Principal Manager refer section **7.4 Modifying ACL Manager Implementation** and refer to section **7.5 Modifying Principal Manager Implementation** of *FioranoMQ Handbook*.
3. Configure the Principal Manager as per the desired Directory Server. Sample configuration for Netscape Directory Server is as shown in the figure below in section LDAP Provider DN.
4. LDAP based ACL manager depends on Timer Service. Specify the instance of TimerService as specified in figure below:



5. Right-click on the FES/FPS node and select **Save** from the pop-up menu.

8.7.6.1 Sample Configuration – Netscape Directory Server

Setting the Name

This name has to be the admin of the LDAP server, as you have to start the Initial Context with the Admin.

PRINCIPAL = uid=admin, ou=Administrators, ou=TopologyManagement, o=NetscapeRoot

Setting the password

Enter the password of the Admin of the LDAP Server to whom you want to connect as shown in the Figure below

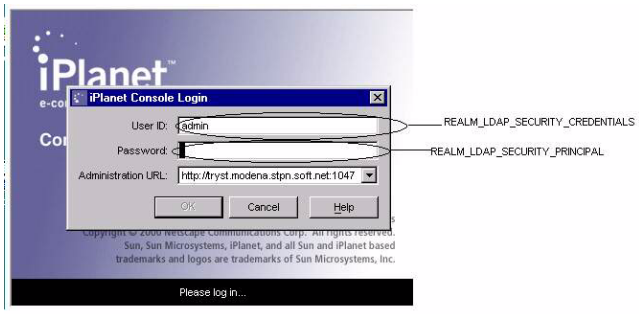


Figure: iPlanet Console Login Dialog Box

LDAP Initial Context Factory

The Initial Context Factory to be used, corresponding to the directory server.

LdapInitialCtxFactory = com.sun.jndi.ldap.LdapCtxFactory

LDAP Provider URL

This can be set according to the directory server being used.

LdapProviderUrl = ldap://ldapserver:389

LDAP Provider DN

This variable needs to be set to the suffix variable that you have set up while installing the LDAP Server as shown in the figure:

LdapProviderDn = dc=modena, dc=stpn, dc=soft, dc=net

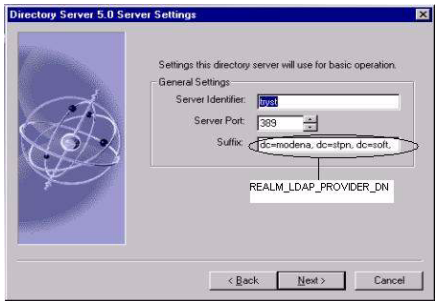


Figure: Directory Server Settings

LDAP security authentication

Set this variable to:

LdapSecurityAuthentication = Simple

LDAP User and Group Object classes

8.7.6.2 Sample Configuration – ApacheDS1.5.4

Note : The steps mentioned here require the installation of Apache Directory Studio.

Setting up the Directory Service

1. Stop any running instance of apacheds.
2. Take backup of server.xml

   /var/lib/apacheds-1.5.4/default/conf/server.xml (DEFAULT PATH. If the DS instances were installed in a different location then server.xml will be available inside that directory.)

3. Modify server.xml as follows, add the following line within the tag *</partitions> ... </partitions>*

   <jdbmPartition id="fiorano" cacheSize="100" suffix="o=fiorano,c=US" optimizerEnabled="true" syncOnWrite="true"/>

4. Run apacheds.

   /etc/init.d/apacheds start

5. Login through Apache Directory Studio.

User	: uid=admin,ou=system.	(Default)
Password	: secret.	(Default)

6. Import the following LDIF content using Apache Directory Studio. (Menu: *LDAP -> New LDIF File*)

   dn: o=fiorano,c=us

   objectclass: top

   objectClass: organization

   o: fiorano

   dn: cn=FMQServerConfigFiles,o=fiorano,c=us

   objectclass: top

   objectClass: organizationalRole

   cn: FMQServerConfigFiles

   dn: cn=FioranoMQUsers,o=fiorano,c=us

   objectClass: top

objectClass: organizationalRole

cn: FioranoMQUsers

dn: cn=FioranoMQGroups,o=fiorano,c=us

objectClass: top

objectClass: organizationalRole

cn: FioranoMQGroups

dn: cn=ACL,o=fiorano,c=US

objectclass: top

objectClass: organizationalRole

cn: ACL

dn: cn=FMQRoot,o=fiorano,c=us

objectClass: inetOrgPerson

objectClass: organizationalPerson

objectClass: person

objectClass: top

cn: FMQRoot

cn: system administrator

sn: administrator

displayname: Directory Superuser

userpassword:: c2VjcmV0

7. Relogin through Apache Directory Studio to see the added children.

Setting up the profile for use with ApacheDS1.5.4

Note : Make sure that the steps mentioned in section 8.7.6.2 have been completed before moving on to these steps.

1. Open the profile for off-line editing through the Profile Manager using Studio by clicking on the Profile Manager pane
2. Make sure that all the properties except the LdapProviderUrl are reset to their original value.
3. In the LDAP Provider URL, the port number has to be **10389** and the ip address has to be that of the server that is running ApacheDS.

8.7.7 XML Security Realm

1. Open the desired profile for off-line editing through the **Profile Manager** using Studio
2. Modify the Implementation property of Principal Manager and ACL Manager to XML. For more information on how to modify ACL Manager and Principal Manager refer to section **7.4 Modifying ACL Manager Implementation** and section **7.5 Modifying Principal Manager Implementation** of *FioranoMQ Handbook*.
3. Configure **Principal Manager** and **ACL Manager**.
4. Right-click on the FES/FPS node and select **Save** from the pop-up menu.

8.7.7.1 Configuring Principal Manager

UserFileName

The name and path of the xml file containing user information. Default is user.xml.

GroupFileName

The name and path of the xml file containing group information. Default is group.xml.

Path

The absolute or relative path where user and group files are stored. On specifying a absolute path, user and group files are saved to that location, while specifying a relative path saves user and group files to [FIORANO_DB_PATH]\[relative path entered], which in default cases for FES are %FIORANO_HOME%\runtimedata\%selectedProfile%\FES\run\realm\principal.

8.7.7.2 Configuring ACL Manager

FileName

The name and path of the xml file containing user information. Default is acl.xml.

MaxAcePerACL

Maximum number of entries that an ACL can store. Default is 100.

Path

The absolute or relative path where the xml files are stored. On specifying an absolute path, user and group files are saved to that location, while specifying a relative path saves user and group files to [FIORANO_DB_PATH]\[relative path entered], which in default cases are %FIORANO_HOME%\runtimedata\%selectedProfile%\FES\run\realm\principal.

8.7.7.3 Sample xml files

User.xml

```
<?xml version="1.0"?>

<UserManager>
<User>
      <Name>ADMIN</Name>
      <Password></Password>
      </User>
     <User>
     <Name>Anonymous</Name>
     <Password></Password>
     </User>
     </UserManager>
```

Where:

<UserManager> Root element of the UserManager.

<User> The UserManager may consist of one or more users.

<Name> The name of the user. This is used to identify the user entry and is used in the ACLS and Groups.

<Password> The password of the user. This is stored in encrypted form. Hence, it cannot be specified from outside the system(?).

8.7.7.4 Group.xml

```
<?xml version="1.0"/>

<GroupManager>
<Group>
                <Name>EVERYONE</Name>
                </Group>
                </GroupManager>
```

Where:

<GroupManager> Root element of the GroupManager.

<Group> The GroupManager may consist of one or more groups.

<Name> The name of the group. This is used to identify the group entry and is used in the ACL table.

<Member> A group can consist of one or more members. These members must exist in the user table.

8.7.7.5 acl.xml

```
</AclManager>
        <ACL>
                </Name>LOOKUP</Name>
                <AclEntry Type="POS">
                <Principal>EVERYONE</Principal>
                <Permission>LOOKUP</Permission>
                </AclEntry>
                </ACL>
                </AclManager>
```

Where:

<AclManager> Root element of the ACL dtd.

<ACL> The AclManager consists of one or more ACLs, which consists of all the information about ACL.

<Name> It specifies the name of the ACL.

<AclEntry> An ACL consists of one or more AclEntry, which can be negative (NEG) or positive (POS).

<Principal> An AclEntry consists of a Principal, which can be a user or a group.

<Permission> An AclEntry consists of 0 or 1 permission to perform certain tasks

8.8 Event Process Security

This feature enables administrator to define the permissions created for the destinations used by the Event Process. This allows administrators to control the set of users allowed to access destinations created/used by the components within an Event Process, ensuring that unauthorized users do not get access to these destinations.

8.8.1 How ACLs Work

All major actions taken by the users on an Event Process are mapped with one or more permission which determines whether the user has the permission to take that particular action or not. If the user is not allowed to take the action, the server throws a Security Exception. For an Event Process, the following ACLs control user-actions on the process.

- Application Level Permission
- Global/System Level Permission(s)

8.8.1.1 Application Level Permission

Each Event Process can be assigned an ACL that determines the set of users and actions allowed for that Event Process. These permissions, if set, over-ride the permission set at the system level.

Note: With default installation, none of the Event Process has any application level ACL set. All behavior is governed by ACLs set at System/Global level, refer to section 8.7.1.2 Global/System Level Permission(s) for more information. The following permissions are available at the Event Process Level.

PERMISSION TO COMPOSE AN APPLICATION

This permission controls whether a user is allowed to compose/update/delete an Event Process from the Server repository.

PERMISSION TO CHANGE PROPERTIES OF AN APPLICATION

This permission controls whether a user is allowed to change the following properties of a running Event Process:

- Logging properties of a service instance
- Document Tracking property at ports of service instances
- Route Transformation at routes connecting service instances

This permission thus controls run-time changes in the above properties.

Note: For a user to successfully modify the above properties of an Event Process, the user should also have the permission to ***compose that Event Process*** *(Permission described above)*.

PERMISSION TO LAUNCH AN APPLICATION

This permission controls whether a user is allowed to launch and/or synchronize the Event Process.

PERMISSION TO KILL AN APPLICATION

This permission controls whether a user is allowed to kill the Event Process.

PERMISSION TO VIEW RUNNING AND SAVED APPLICATIONS

This permission controls whether a user is allowed to view the application from the server repository. If a user tries to view an application without the appropriate permission, a Security Exception is thrown and the user is denied access to the information.

PERMISSION TO REMOTELY ADMINISTRATE AN APPLICATION

This permission controls the set of users who can use service instances from this Event Process as a remote service instance in their own Event Processes. By default, all users are allowed to remotely access the service instances of all Event Processes. This behavior can be controlled by the same permission when set at system level (as described in the Global/System Level Permission(s) section).

8.8.1.2 Global/System Level Permission(s)

The permissions set at system level control the overall behavior of the system. For permissions related to an Event Process, if Event Process Level permissions are not set, Global/System Level permissions govern the set of actions allowed for a user. As explained earlier, the default behavior of the Event Processes is governed by Global/System Level permissions only. The following permissions are available at this level.

ALL PERMISSIONS

This permission over-rides all other permissions set at system level. If this particular permission is granted the user/grout will be allowed to perform all actions unless the Application Level Permission specifically denies the action.

PERMISSION TO COMPOSE AN APPLICATION

This permission serves the same purpose as at the Application Level (see previous section) except that it applies to all Event Processes.

PERMISSION TO CHANGE PROPERTIES OF AN APPLICATION

This permission serves the same purpose as at the Application Level (see previous section) except that it applies to all Event Processes.

PERMISSION TO LAUNCH AN APPLICATION

This permission serves the same purpose as at the Application Level (see previous section) except that it applies to all Event Processes.

PERMISSION TO KILL AN APPLICATION

This permission serves the same purpose as at the Application Level (see previous section) except that it applies to all Event Processes.

PERMISSION TO VIEW RUNNING AND SAVED APPLICATIONS

This permission serves the same purpose as at the Application Level (see previous section) except that it applies to all Event Processes.

PERMISSION TO CREATE OR EDIT AND REMOVE SERVICE ACL

This permission has been deprecated as it is no longer used within the system.

PERMISSION TO CREATE OR UPDATE AND DELETE A SERVICE

This permission controls whether a user is allowed to make modifications in a particular service e.g. adding/removing a resource to/from a service, changing icon, adding/removing service dependencies, etc.

PERMISSION TO CREATE AN ACL

This permission controls whether a user is allowed to create/change a System Level or Application Level permission.

PERMISSION TO CREATE OR EDIT AND DELETE A PRINCIPAL

This permission controls whether a user is allowed to create/delete a user/group on the system. The permission also determines whether a user is allowed to change the password for an existing user.

PERMISSION TO CONFIGURE A FPS

This permission has been deprecated as it is no longer used within the system.

PERMISSION TO REMOTELY ADMINISTRATE AN APPLICATION

This permission serves the same purpose as at the Application Level (see previous section) except it applies to all Event Processes.

PERMISSION TO DELETE MESSAGES IN QUEUE

This permission has been deprecated as it is no longer used within the system.

PERMISSION TO CLEAR USER EVENTS

This permission has been deprecated as it is no longer used within the system.

PERMISSION TO PUSH MESSAGES IN QUEUE

This permission has been deprecated as it is no longer used within the system.

PERMISSION TO ADMINISTRATE A GROUP

This permission controls whether a user is allowed to add/remove members from an existing group.

8.8.3 Default Allowed Set Of Users

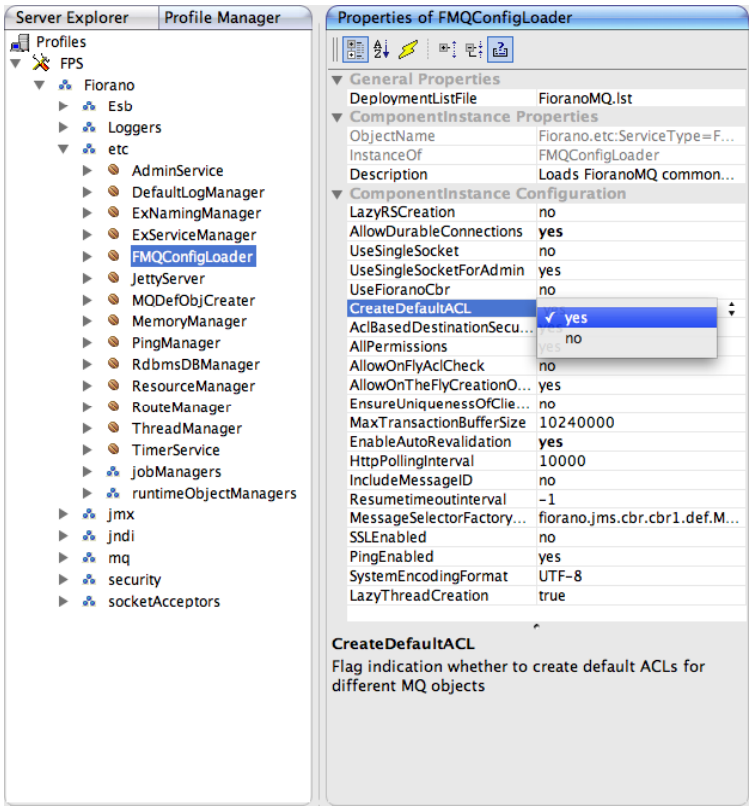
When an Event Process is launched, ACLs will be created for all the default (i.e. system-created) destinations such that the following users can publish/subscribe/send/receive the information on these destinations.

1. User launching the Application
2. The set of users allowed by the permission named PERMISSION TO REMOTELY ADMINISTRATE AN APPLICATION.

All components launched as a result of an Event Process launch/synchronize action will use the credentials of the user performing the launch/synchronize operation to create all connections with the Peer Server. Similarly, all connection(s) created by routes within the Event Process to remote Peer Server(s) will be created using the credentials of the user launching/synchronizing the Event Process.

Note: No specific permissions will be created for user-defined destinations. If a user-defined destination does not exist while launching the Event Process, it will be created by the Peer Server but the ACL settings on these destinations depend upon other Peer Server profile parameters. These parameters can be configured by opening the profile in Studio and navigating to Fiorano->etc->FMQConfigLoader as illustrated in the figure below. These parameters are as described below:

- **CreateDefaultACL** – This parameter decides whether a default ACL should be created for a newly created destination or not. The default ACL is to allow or disallow **Everyone** (based on the value of All Permissions parameter defined below) from accessing the destination.
- **AllPermissions** – This parameter determines whether the default permission created should be positive or negative.

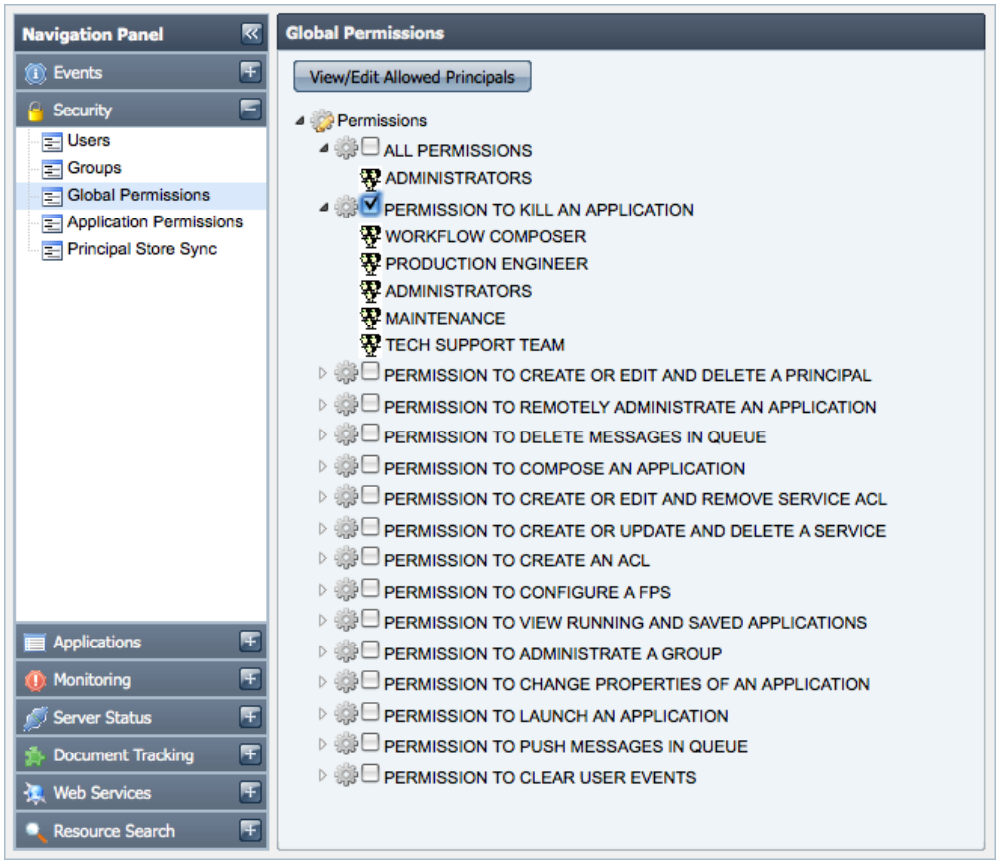


8.8.4 Changing Default Permissions

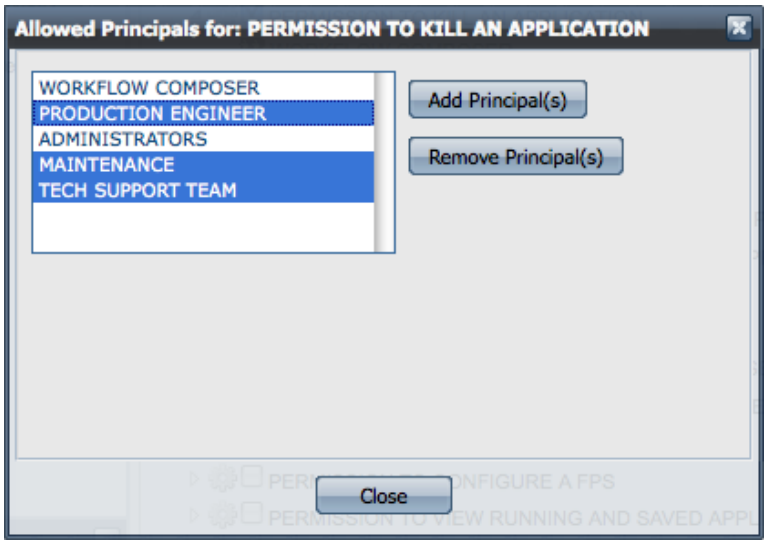
Default permissions at Application Level and System Level can be changed such that only a chosen set of users are allowed to perform the desired actions on desired set of Event Processes. These changes can be made via the Fiorano Web dashboard interface which provides an interactive UI to change these ACLs.

8.8.4.1 Changing Global Permissions

All available global permissions are listed under the **Security->Global Permissions** component of the Web interface. Each of these permissions when expanded displays a list of principals that are allowed to perform the action specified by that permission (as shown in figure below).

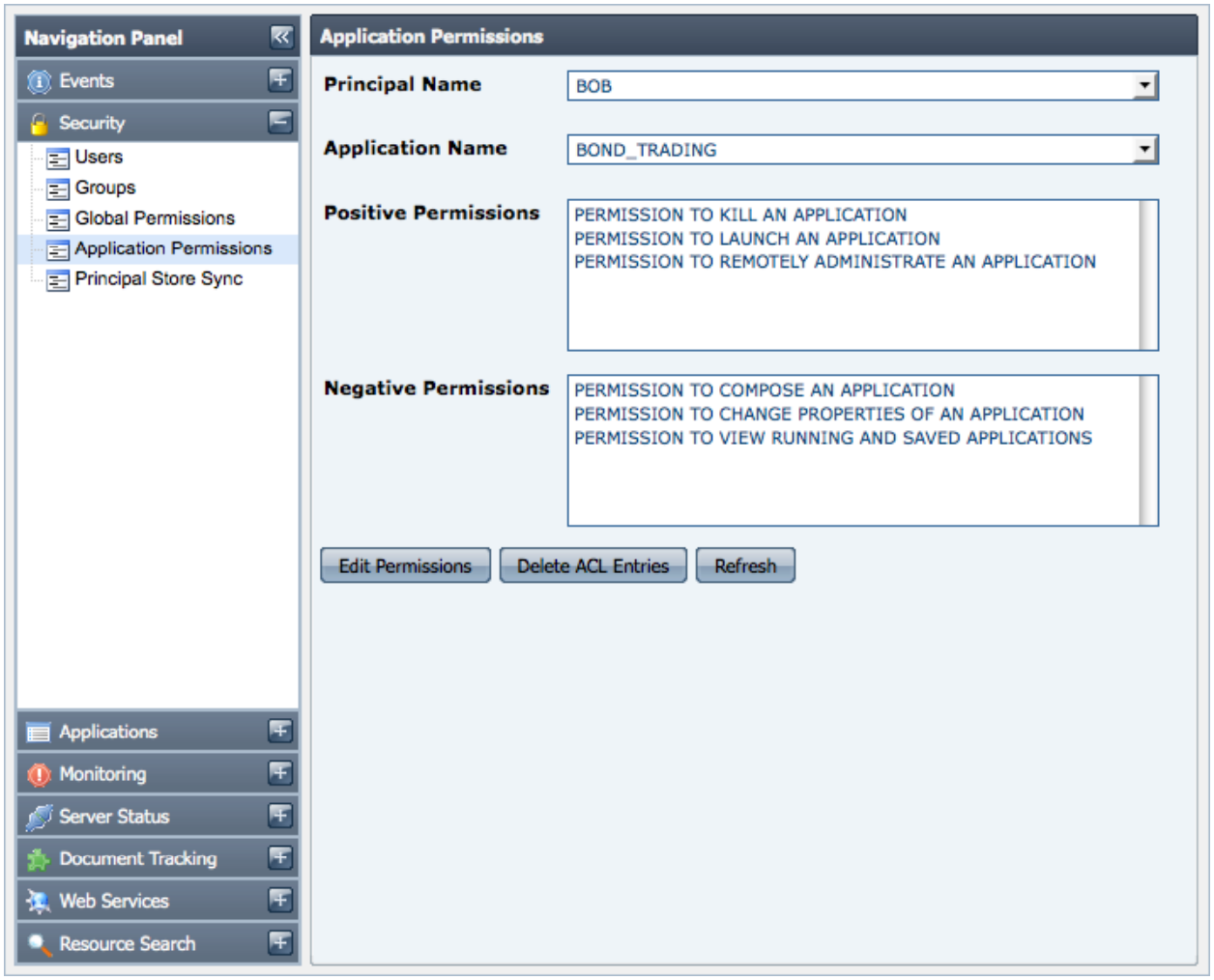


This list of allowed principals can be modified by selecting the appropriate checkbox and clicking the **View/Edit Allowed Principals** button. This pops-up another window showing the list of allowed users. You can also remove selected users from the permission by clicking the **Remove Principal(s)** button or add new users to the permission by clicking the **Add Principal(s)** button as shown in the figure below.

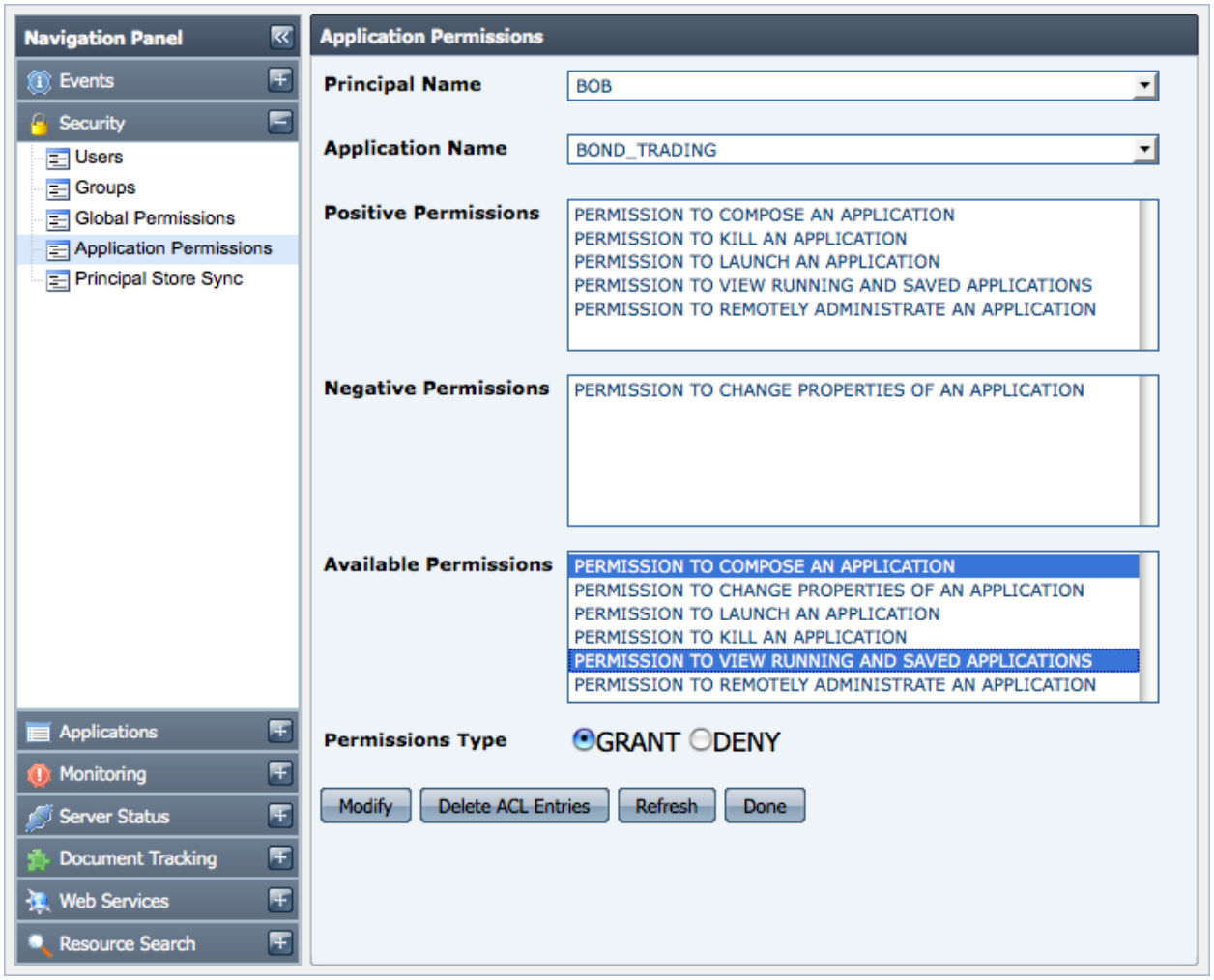


8.8.4.2 Changing Application Level Permissions

Application Level Permissions can be viewed and modified by navigating to the **Security->Application Permissions** panel of the Web interface. Selecting a principal and Event Process name in this view displays the set of positive and negative permissions assigned to that principal for the selected Event Process. The figure below shows that the user **BOB** has 3 positive and 3 negative permissions for the Event Process named **BOND_TRADING**.

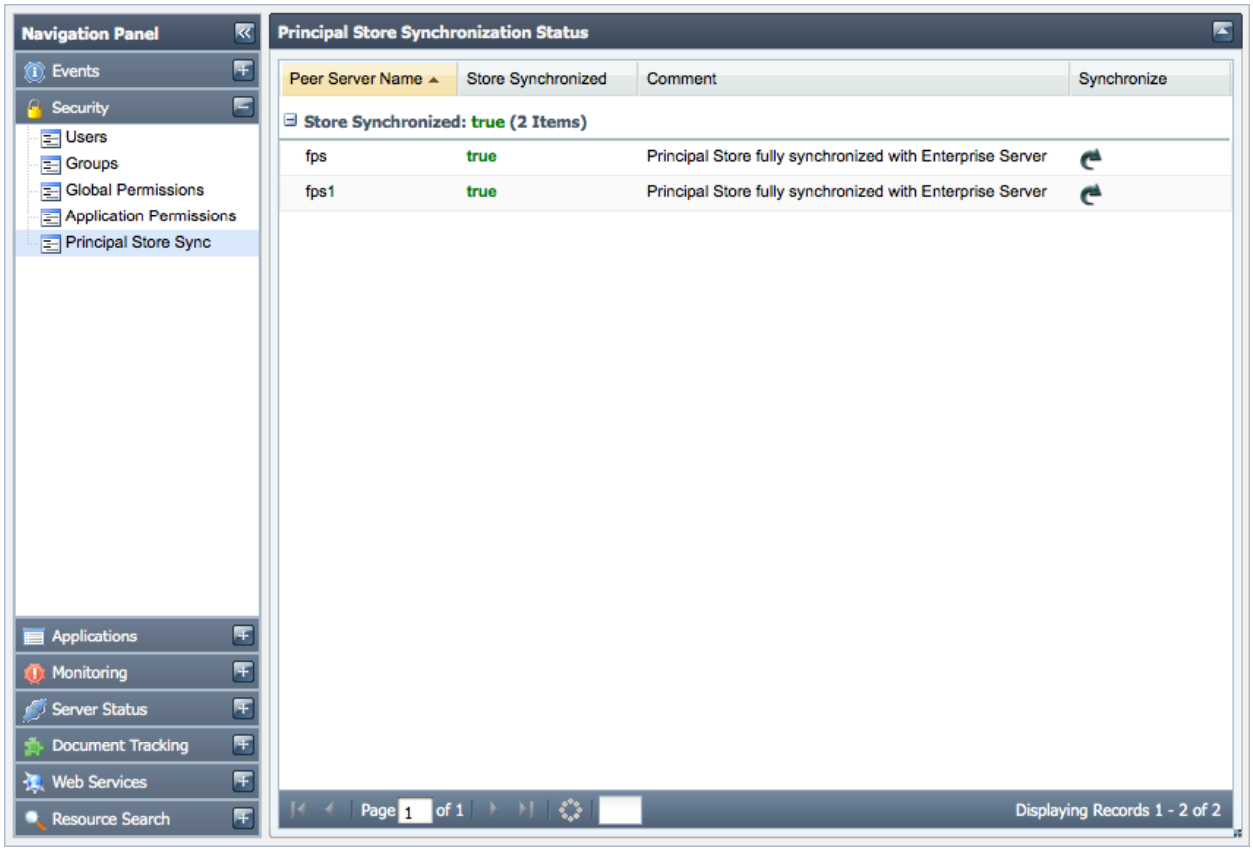


The set of positive and negative permissions can be modified by clicking on the **Edit Permissions** button. For example, to change the type of one or more negative permissions shown in figure above to positive permissions, first select the desired negative permissions from the set of Available Permissions; then choose **GRANT** option and click the **Modify** button. The selected principal would then be granted the permission to perform the actions represented by those permissions as shown in the figure below.



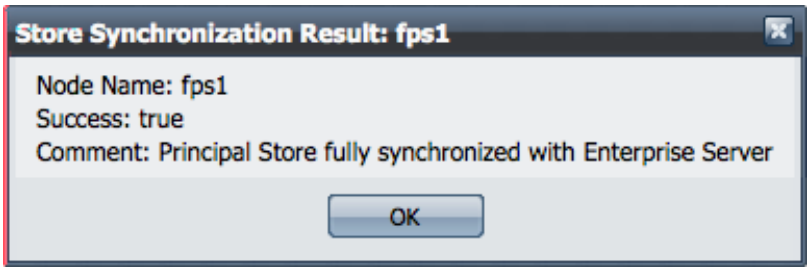
8.8.5 Principal Store Synchronization

Event Processes are entities that reside in the Enterprise Server repository while destinations used by the components are created on the Peer Server(s). To ensure that a user who has permission to launch an Event Process from the Enterprise Server also has the permission to publish/subscribe/send/receive messages onto destinations on the Peer Server, the username must be present in security data store of both the Enterprise Server and the Peer Server. This is achieved by ensuring that the **User & Group Store** (collectively called as **Principal Store**) of both the Enterprise Server and all connected Peer Servers are always in sync with each other. Any change in Principal Store of Enterprise Server is propagated to all connected Peer Servers . Whenever a new Peer Server becomes available on the Fiorano network, a complete one-way synchronization of the Principal Store of that Peer Server and the Enterprise Server will happen, where the Principal Store of Enterprise Server replaces the Principal Store of the Peer Server. The Enterprise Server is responsible for maintaining the status of synchronization with each connected Peer Server. The status of synchronization with each Peer Server can be viewed by navigating to the **Security->Principal Store Sync** component of Web interface as illustrated in the figure below.

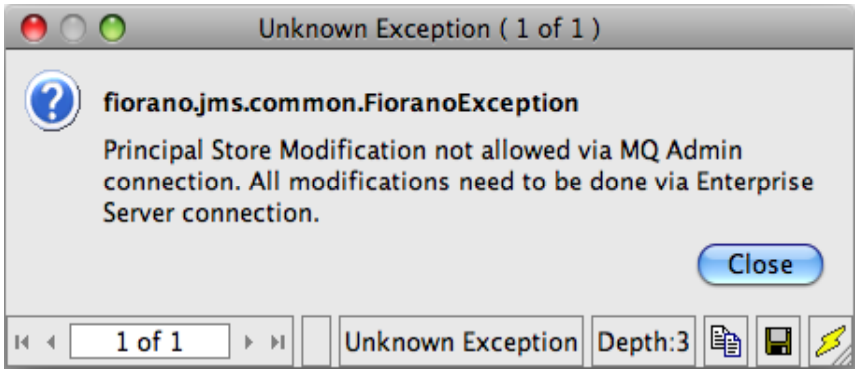


Important:

An Event Process launch/synchronize actions will be disallowed if the Principal Stores of Enterprise Server and all running Peer servers used within the Event Process are not in sync with each other. An exception will be thrown specifying the reason due to which the stores were found to be out-of-sync. In such cases, the user will have to take corrective actions in order to launch that Event Process on problematic Peer Server(s). The Web interface provides an option to force a re-sync of Principal Store of Enterprise Server with any particular Peer Server. This is achieved by clicking on the image under the Synchronize column as shown in the figure above. The result of this operation will be displayed to the user as shown in figure below.



Modifying the Principal Store of Peer Server by directly creating a connection with Peer Server is disabled. An exception will be thrown whenever a user tries to perform such an operation. Any modifications in the Principal Store of Peer Server can only be performed via the Enterprise Server connection. The figure below shows the result of an operation when a user tries to modify Principal Store of Peer Server by logging into Peer Server via Studio.



Chapter 9: Fiorano Mapper

The Fiorano Mapper is a high-end graphical tool that presents you with both source document structure and target document structure side-by-side and lets you define semantic transformation of data by simply drawing lines between nodes, elements, and functions.

The Fiorano Mapper uses standards based XSLT (Extensible Stylesheet Language for Transformations), which is a language for transforming documents from one XML structure to another.

Additionally, Fiorano Mapper ensures that the source and target document structures conform to the DTD (Document Type Definition) standards.

9.1 Key Features of Fiorano Mapper

The Fiorano Mapper performs a variety of operations including:

- Transforming one or more XML, XSD, DTD, or CSV files.
- Generating XML, XSD, DTD, or RDBMS queries for inserting, deleting, or updating records as output of the transformation.
- Using Funclets to define complex mapping expressions.
- Validate the transformation.
- Define the transformation (mapping) with simple drag-and-drop actions.

9.2 Fiorano Mapper Environment

The Fiorano Mapper tool has an easy to use graphical interface that consists of two views:

- **MapView:** Displays a graphical representation of the mappings between the Input and Output Structures and other details.
- **MetaData:** Displays the XSL Transformation that is created by defining the mappings between the loaded Input and Output Structures.

The interface of the Fiorano Mapper tool is displayed in Figure 9.2.1.

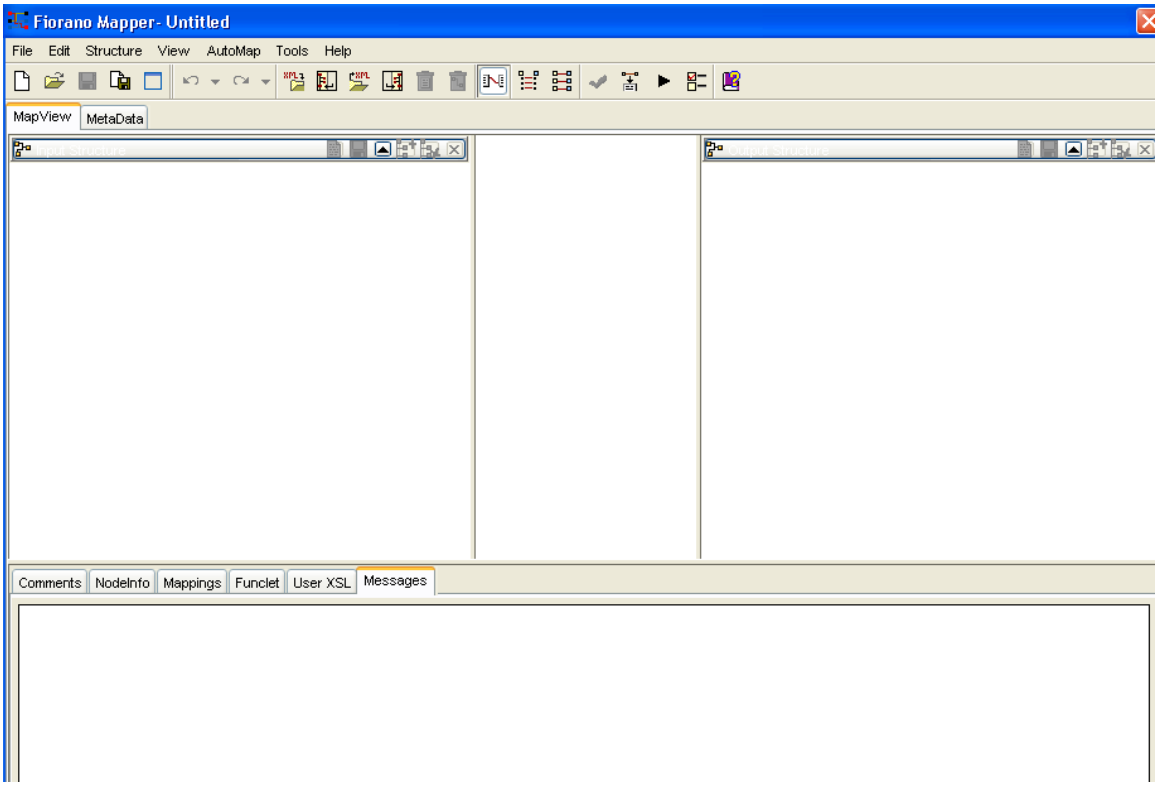


Figure 9.2.1: Fiorano Mapper's Main Window

The Fiorano Mapper tool consists of the following interface elements:

- Menu Bar
- Toolbar
- MapView Panel
  - Input Structure Panel
  - Lines Panel
  - Output Structure Panel
  - Details Panel
    a. Comments tab
    b. NodeInfo tab
    c. Mappings tab
    d. Funclet tab
    e. User XSL
    f. Messages tab
- Metadata Panel
  - Transformation Code

– Error Messages

9.2.1 Menu Bar

The Fiorano Mapper's menu bar organizes the commands that you can execute by using them. The Bar is divided into seven categories of menu, as shown in Figure 9.2.2.

Figure 9.2.2: Menu Bar of the Fiorano Mapper

The description of the options, in the menu bar, as shown in Figure 9.2.2, is as follow:

9.2.1.1 File

- **New:** Creates a new project
- **Open:** Opens an existing project
- **Save:** Saves a project
- **Save As:** Saves a project with a given name
- **Maximize:** Maximize the selected project page
- **Page Setup:** Sets the dimensions and settings for the selected page
- **Print:** Prints the selected page or project
- **Exit:** Exits Fiorano Mapper tool

9.2.1.2 Edit

- **Undo:** Undo the last action performed
- **Redo:** Redo the last action performed

9.2.1.3 Structure

- **Load Input Structure:** Loads the input structure
- **Import Input Structure:** Imports input structure from Fiorano Mapper project file
- **Load Output Structure:** Loads the output structure
- **Import Output Structure:** Imports output structure from Fiorano Mapper project file
- **Clear Input Structures:** Clears the existing input structures, if any
- **Clear Output Structures:** Clears the output structures, if any
- **Clear Structures:** Clears all input and output structures
- **Clear Mappings:** Clears the existing mappings

9.2.1.4 View

- **Show All Mappings:** Views all existing mappings

- **XSLT Properties:** Modifies the properties of the generated XSL

9.2.1.5 AutoMap

- **Child to Child:** Creates mappings from child to child automatically
- **Child to Child Recursively:** Creates mappings between descendants of the selected nodes
- **Default and Fixed Values:** Creates mappings for nodes with default and fixed values
- **Empty Values:** Creates mappings for empty elements (elements with no content)
- **Delete Unused Duplicate Node:** Deletes the unused duplicate nodes

9.2.1.6 Tools

- **Validate Mappings:** Validates existing mappings
- **Generate MetaData:** Generates metadata based on existing mappings
- **Test:** Tests the generated XSLT
- **Create/Edit User Defined Function(s):** Creates or modifies a user-defined functions
- **Import User Defined Function(s):** Imports user-defined functions from Fiorano Mapper project file
- **Schema Repository:** Adds and removes the Schema Repository
- **Options:** Configures the Fiorano Mapper options

9.2.1.7 Help

- **Help Topics:** Displays online help
- **About:** Displays product information

9.2.2 Toolbar

Most of the common tasks can be directly performed using the toolbar as shown in Figure 9.2.3.

Figure 9.2.3: Toolbar of the Fiorano Mapper

The descriptions of the buttons in the toolbar, shown in Figure 9.2.3, starting from left is given in table 9.1:

Icon	Description
	Creates a new project
	Opens an existing project
	Saves the changes made in the project
	Saves the project with the given name
	Maximize the selected project page
	Undoes the last action performed
	Redoes the last action performed
	Loads input structure
	Imports input structure
	Loads output structure
	Imports output structure
	Clears input and output structure
	Clears the existing mappings
	Shows all existing mappings
	Creates mappings from child to child automatically
	Creates mappings between descendants of the selected nodes
	Validates existing mappings

Icon	Description
	Generates metadata based on existing mappings
	Tests the generated XSLT
	Configures Fiorano Mapper Options
	Displays online help

Table 9.1 Toolbar buttons of Fiorano Mapper

9.2.3 MapView

The MapView shows the Input and Output Structures and the mappings defined in the pane. This view allows users to load the input and output structures.

This view consists of the following panels:

- Input Structure Panel
- Line Panel
- Output Structure Panel
- Details Panel

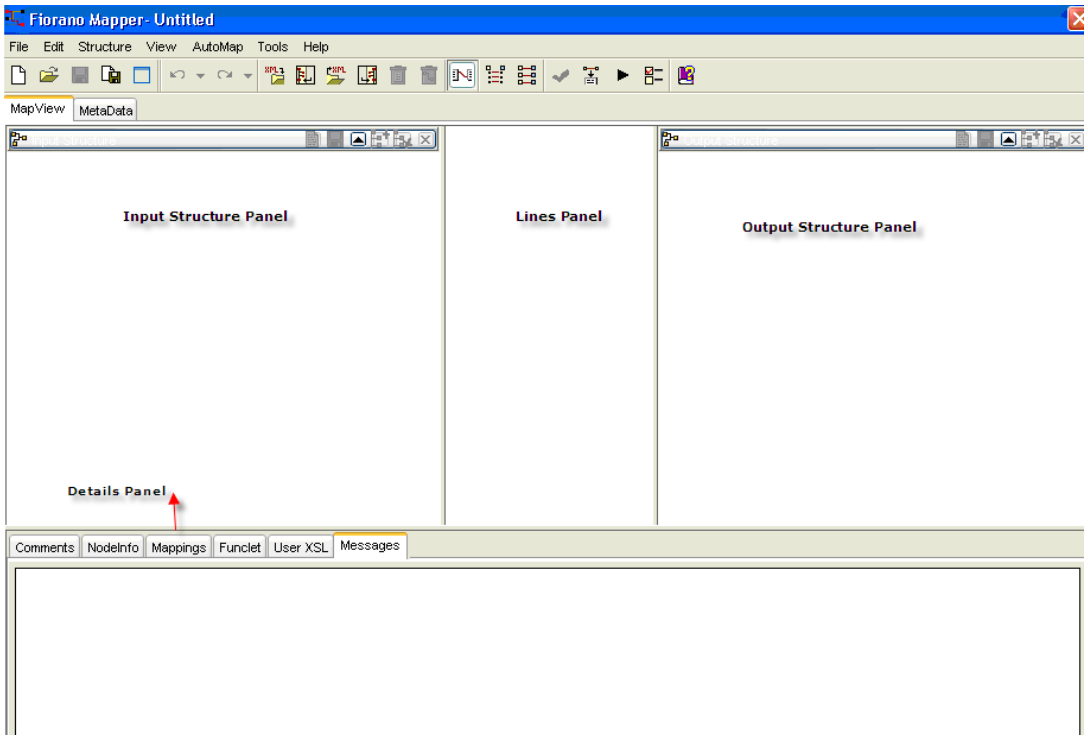


Figure 9.2.4: MapView

9.2.3.1 Input Structure Panel

This panel shows the input specification structure in a tree format.

9.2.3.2 Lines Panel

The middle panel in MapView is the line panel. It shows the mappings defined by lines (called Mapping lines). A Mapping can be selected by selecting one of the mapping lines in the line panel.

A Function icon at the end of a mapping line indicates that mapping uses function(s) as shown in Figure 9.2.5.

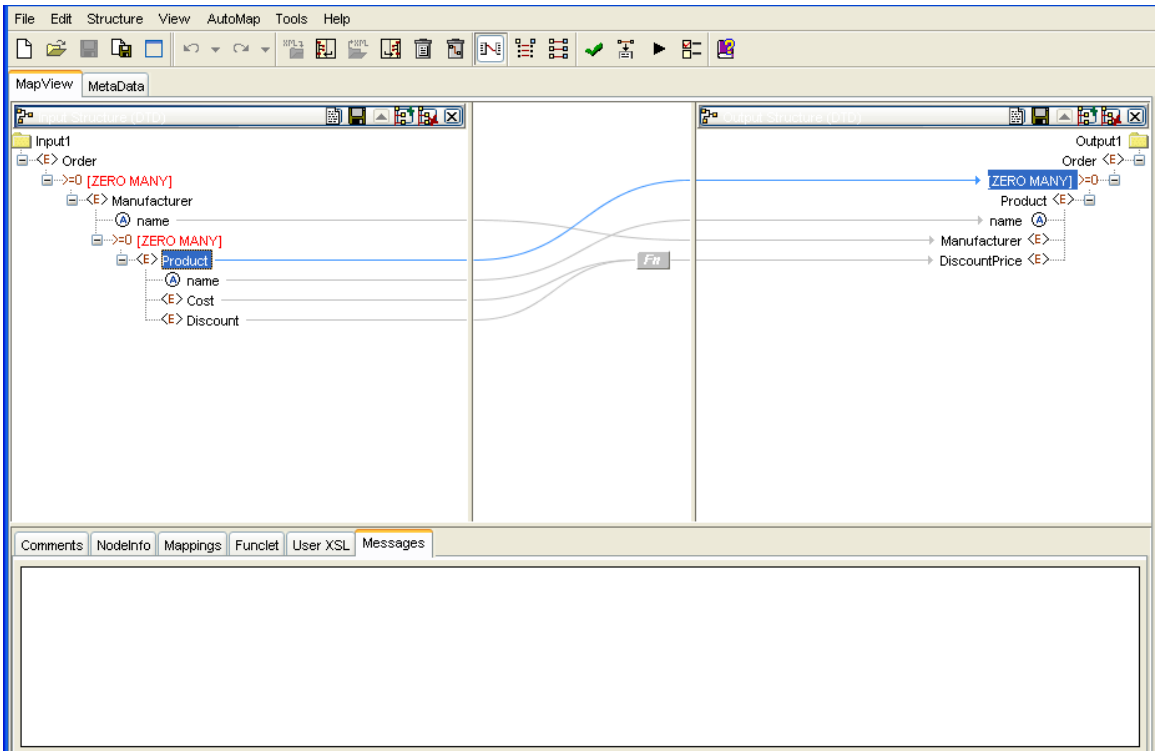


Figure 9.2.5: Mappings

9.2.3.3 Output Structure Panel

This panel shows the output document structure in a tree format.

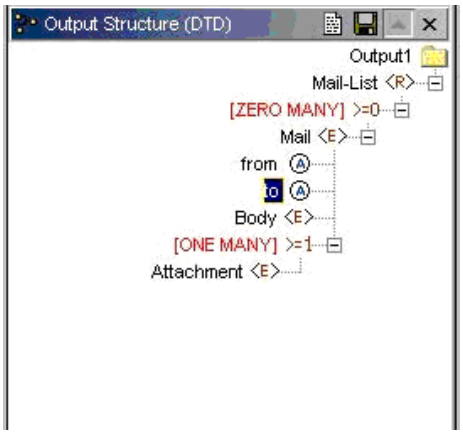


Figure 9.2.6: MapView: Target Structure Panel

9.2.3.4 Details Pane

The Details section of the Mapper tool has six tabs which are as follows:

1. Comments Tab
2. NodeInfo Tab
3. Mappings Tab
4. Funclet Tab
5. User XSL Tab
6. Messages Tab

Comments Tab

The Comments tab is used to add comments to the project. Figure 9.2.7 shows the Comments tab.

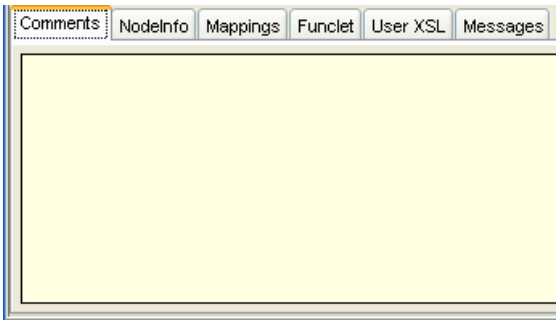


Figure 9.2.7: Comments Panel

NodeInfo Tab

The NodeInfo tab provides the data type and cardinality information about the selected input and output structure node/element.

For example, in case of a DTD element, it gives the element name, its datatype and cardinality as shown in Figure 9.2.8.

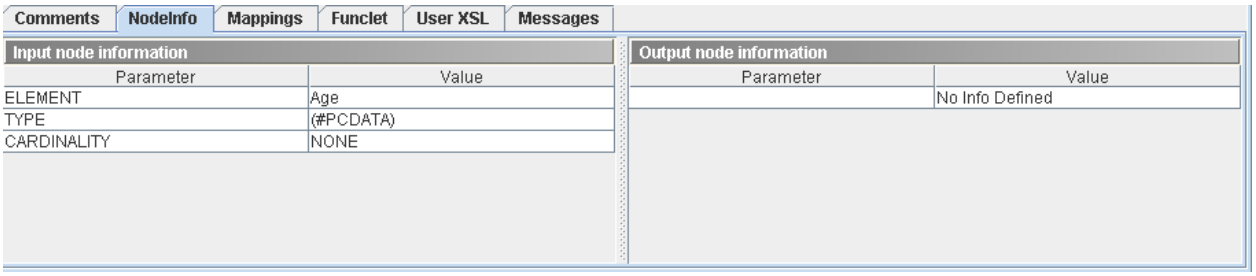


Figure 9.2.8: NodeInfo tab

Mappings Tab

The Mappings view shows the mappings between the input and output nodes/elements, as shown in Figure 9.2.9. The mappings are displayed based on the selected input node. When the user selects an input node, all the mappings originating from it are shown in this tab.

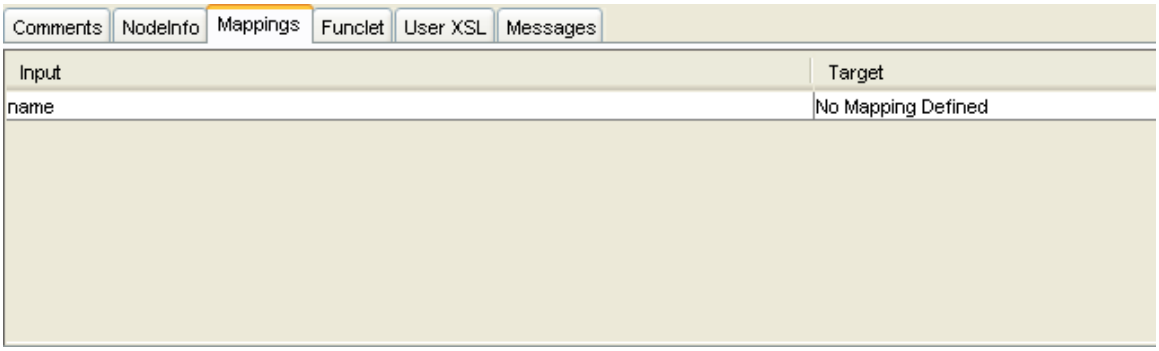


Figure 9.2.9: Mappings View

Funclet Tab

The Funclet tab contains the Visual Expression Builder that provides a graphical view for the mappings defined in the MapView, as shown in Figure 9.2.10. It also shows the functions and their linkage with the input and target nodes/elements.

Note: The Funclet tab is explained in detail in the Visual Expression Builder section later in this chapter.

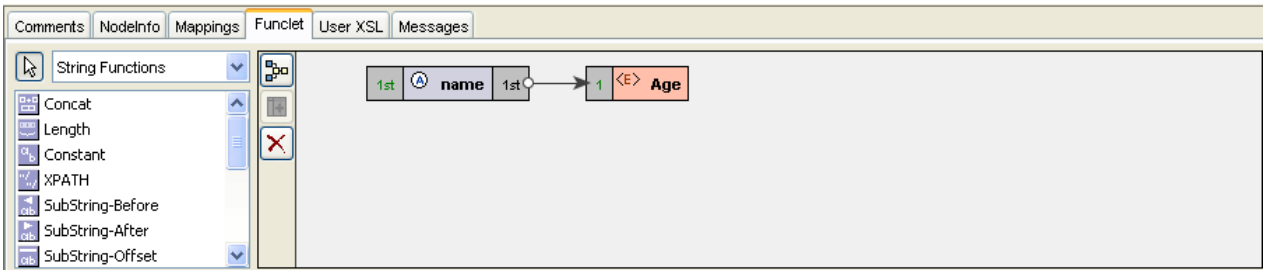


Figure 9.2.10: Funclet View

Messages Tab

The Messages view is used to display the various error and warning messages generated by the tool, as shown in Figure 9.2.11.

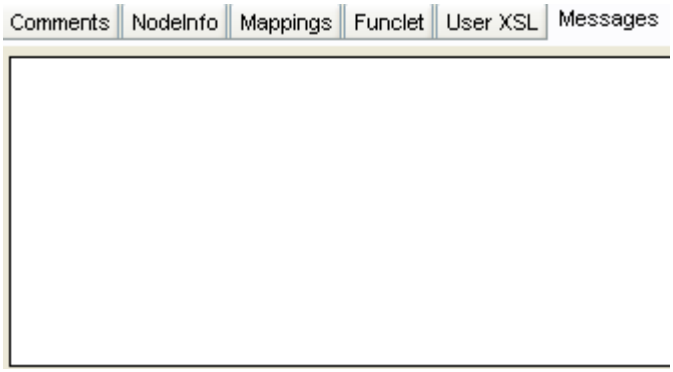


Figure 9.2.11: Messages View

9.2.4 MetaData View

The MetaData view pane shows the transformation XSL generated from the mappings defined in the MapView.

9.2.4.1 Error Messages Panel

Error Messages (if any) are displayed in the **Error Message** pane of the MetaData view. The MetaData view appears, as shown in Figure 9.2.12.

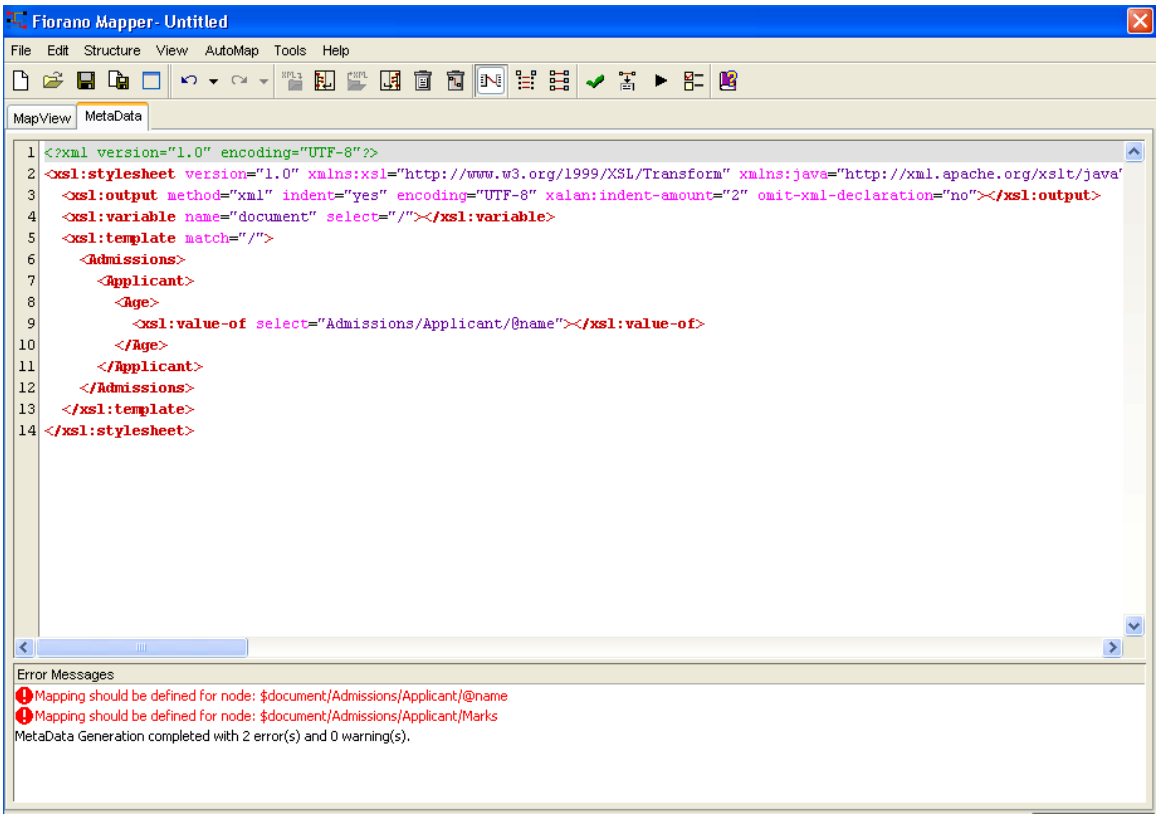


Figure 9.2.12: Viewing the transformation code in the MetaData view

9.3 Working with Input and Output Structures

9.3.1 Loading the Input Structure

Input structure can be loaded in one of the following ways:

1. Click the **Load Input Structure** icon in the Fiorano Mapper **Tools** toolbar.

   Or, click the **Load input structure** icon in the Input Structure panel title bar.
2. The **Select Input Structure Type** dialog box is displayed as shown in Figure 9.3.1.

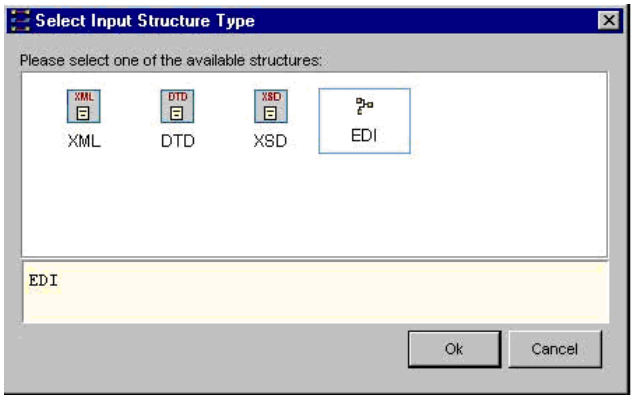


Figure 9.3.1: Selecting the Input Structure format

3. **Select Input Structure Type** dialog box has the following options:
   - **XML:** For loading an XML document as input
   - **DTD:** For loading an DTD document as input
   - **XSD:** For loading an XSD document as input
   - **EDI:** For loading an EDI format document as input

9.3.1.1 Loading an Existing XML Input Structure

1. Select **XML** in the **Select Input Structure Type** dialog box and click **OK.** The Load Input XML Structure dialog box would appear as shown in Figure 9.3.2.

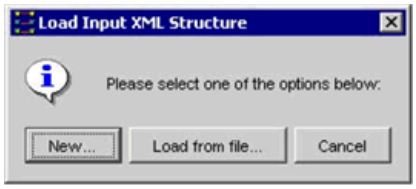


Figure 9.3.2: Loading an Input XML from File

2. Click **Load from file** to select an existing XML file. The **Select XML Structure File** dialog box is **displayed**, as shown in Figure 9.3.3.

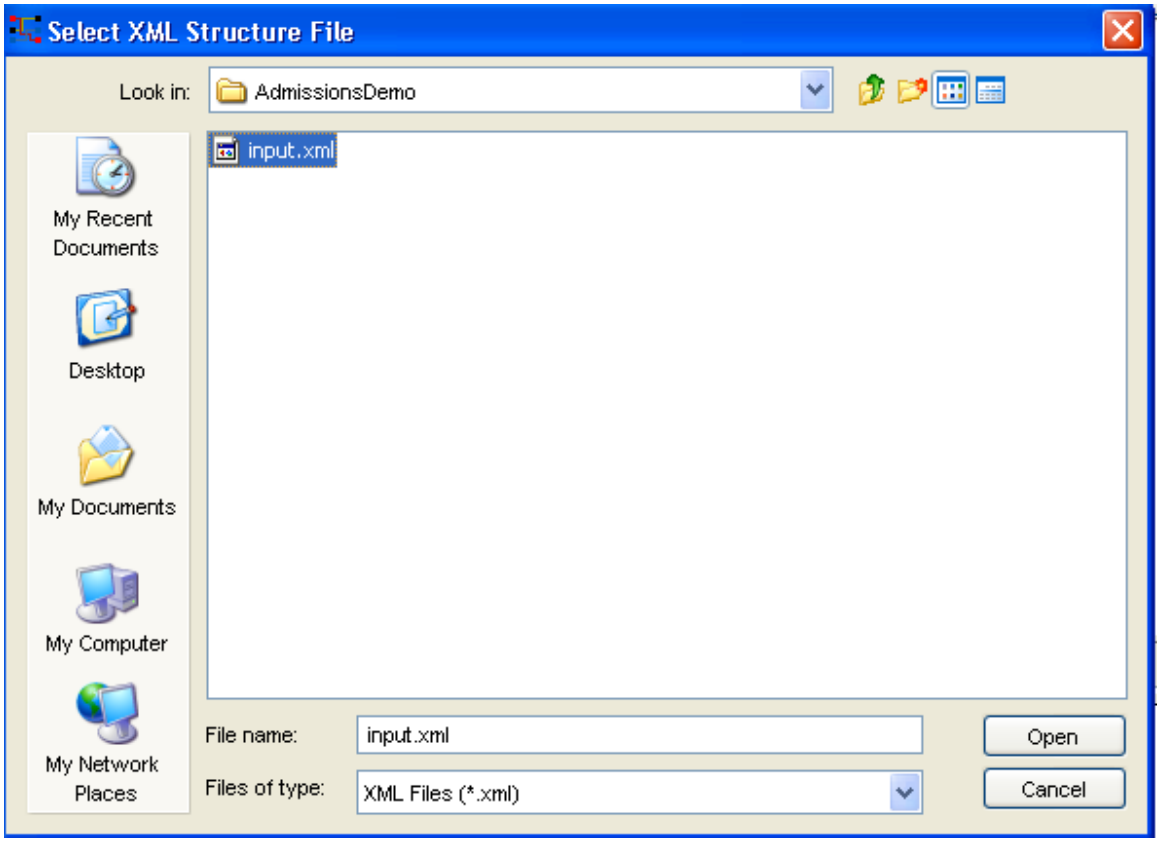


Figure 9.3.3: Selecting the Input XML File

3. Select the required file and click **Open**. The selected XML file is loaded in the Input Structure panel as shown in Figure 9.3.4.

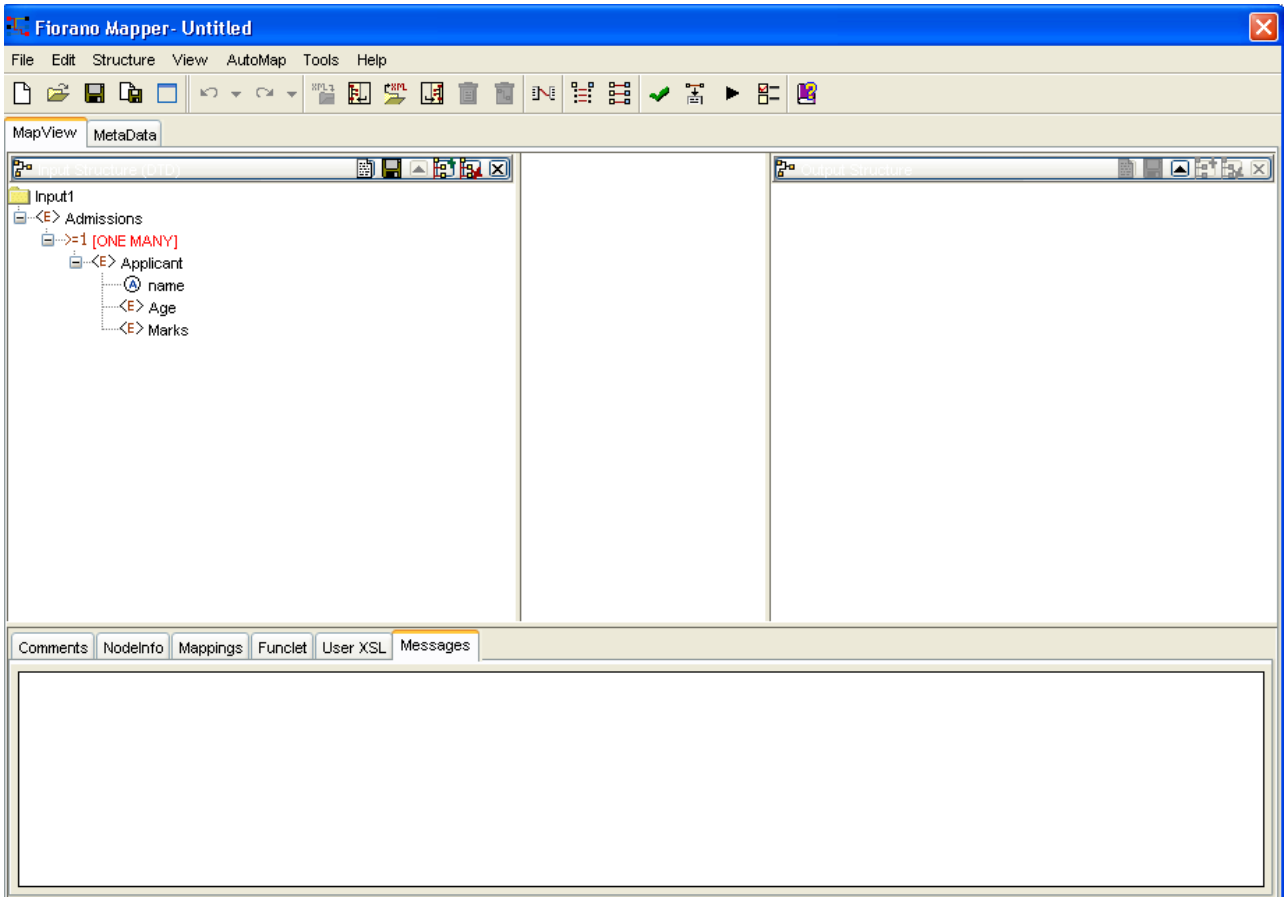


Figure 9.3.4: An Input Structure is loaded

4. To load a new XSD file, click the **New** button in the **Load Input XML Structure** dialog box as shown in Figure 9.3.5. The **Fiorano Mapper XSD Editor** is displayed as shown in Figure 9.3.5.

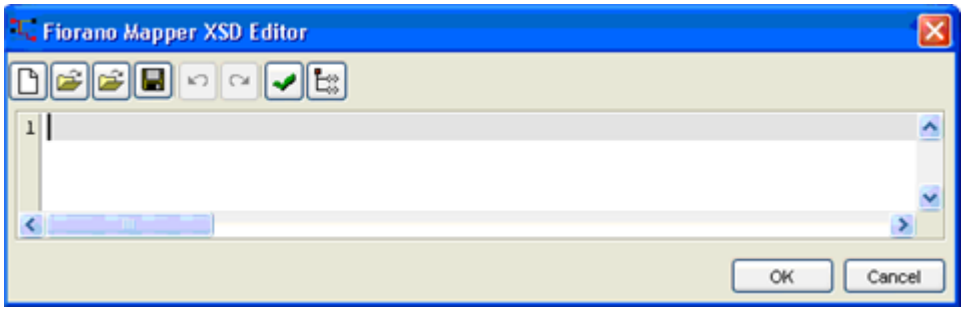


Figure 9.3.5: Loading New XSD File

5. Enter the XSD in the **Fiorano Mapper XSD Editor** dialog box.

   To load an imported XSD from this XSD file, you need to modify the XSD to include the absolute path of the imported and included XSDs. For example, if abc.xsd is stored at c:\hsbc\xsd then, modify the import statement in abc-include.xsd to specify schema location of abc.xsd as file:///C:/hsbc/xsd/abc.xsd, that is,

   <xs:import namespace="http://www.ftisoft.com/LookupAndResolveService.wsdl" schemaLocation="file:///C:/hsbc/xsd/abc.xsd"/>

6. Click the **OK** button. The XSD is loaded into Fiorano Mapper.

9.3.1.2 Loading a New Input XML Structure

While loading a single or multiple XML Input Structure, you can also create your own XML document and use that an Input Structure. This can be accomplished by performing the following steps:

1. Click **Load Input Structure** icon from the tool bar, and select XML from the **Select Input Structure** dialog box, click **OK**, the **Load Input XML Structure** dialog box appears
2. From the **Load Input Structure** dialog box, click **New.**
3. The **Fiorano Mapper XMLEditor** dialog box is displayed. Select the type of XML document that you are creating, by selecting the appropriate option from the **Files of type** drop-down list.
4. You can either enter the new XML structure or load or modify an existing XML structure. To load an existing XML structure, click **Load from file.** The **Select XML Structure File** dialog box is displayed.
5. Select the required file and click **Open**. The contents of the XML file you selected are displayed.
6. You can make changes to this structure as required or as stated earlier, create a completely new structure. After you have entered the new XML structure, you can validate its syntax. To validate the syntax, click **Validate** in the **Fiorano Mapper Editor**.
7. If the XML is not valid, an error message in red is displayed below the input area. Make the necessary corrections, and validate the structure again.
8. Click **OK** to complete the procedure. The defined XML structure is displayed in the Input Structure panel.

9.3.2 Viewing Source of Input Structure

You can replace the source of the Input Structure by performing the following steps:

1. In the Input Structure panel's title bar, click the **View Source** icon.
2. The **Source Viewer** is displayed with the source code of the XML structure, or structures, as shown in Figure 9.3.6.

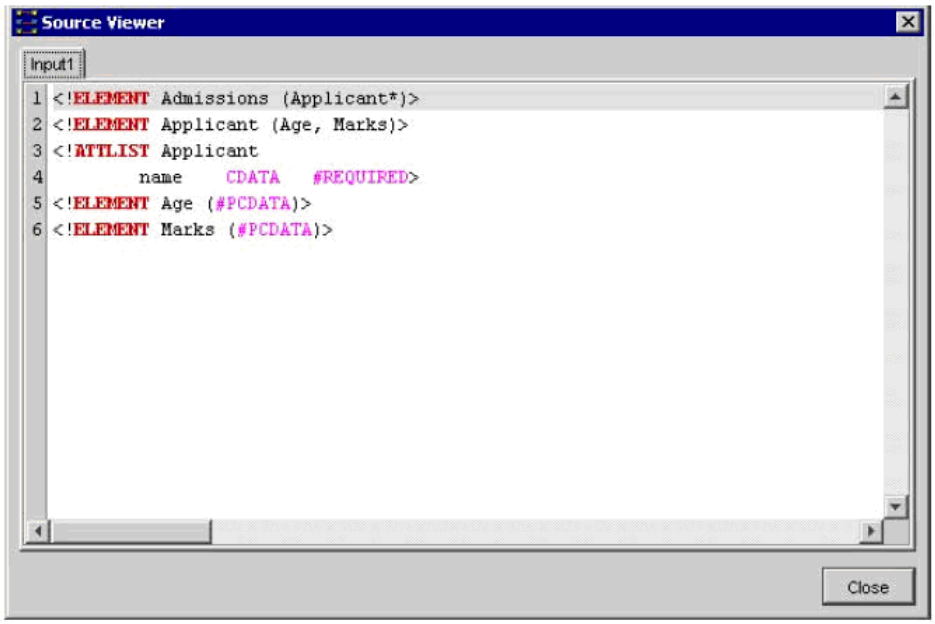


Figure 9.3.6: The Source Viewer window displays the Input/Output Structure source code

3. Click **Close** to close the **Source Viewer**.

9.3.3 Clearing the Input Structure

1. In the Input Structure panel's title bar, click the **Clear input structure** icon.

Figure 9.3.7: Confirming the Clear Structures command

2. A warning message asks you to confirm if you want to remove the Input Structure and clear all mappings, as shown in Figure 9.3.8. Click **Yes** to clear the mappings and remove the input structures.

9.3.4 Loading the Output Structure

Loading the output structure is similar to loading the input structure. However, the types of output structures that you can define are more in number and are more complex. Besides the XML, XSD, and DTD structures, you can also define RDBMS queries as an output structure.

The different types of Output Structures that can be defined are as follows:

- XML
- XSD
- DTD
- CSV
- EDI

These options are displayed when you start the procedure to load an Output Structure:

1. Click the **Load Output Structure** icon in the Fiorano Mapper **Tool** toolbar.

   Or, click the **Load output structure** icon in the Output Structure panel.

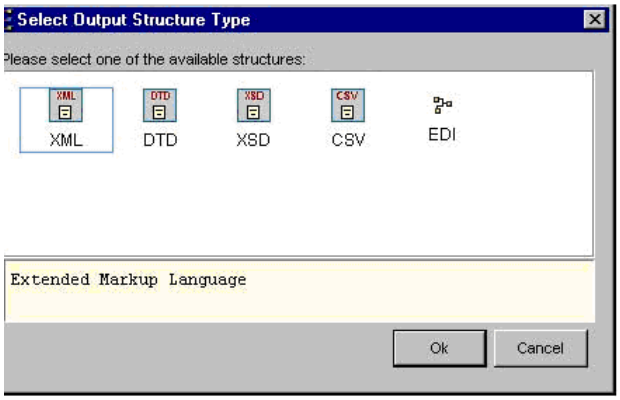


Figure 9.3.8: Selecting the Output Structure

3. The **Select Output Structure Type** dialog box is displayed, as shown in Figure 9.3.8.

The subsequent steps differ depending on which output structure you select in the **Select Output Structure Type** dialog box. The steps for loading and defining an Output structure can be separated into two categories:

- Loading and defining an XML Output Structure: XML, XSD, or DTD Output Structures
- **Loading a CSV Output Structure:** It is a commonly used Output Structure for many event processes.

These have been explained separately below:

9.3.4.1 Loading an XML Output Structure

You can load any of the three different types of XML formats: XML, XSD, and DTD by performing the following steps:

1. The **Select Output Structure Type** dialog box displays five types of XML formats. Select the appropriate type and click **OK**.

2. The **Load Output XML Structure** dialog box is displayed, as shown in Figure 9.3.9. Click **Load from file** to select an existing XML file

Figure 9.3.9: Loading an existing XML File

3. The **Select XML Structure File** dialog box is displayed, as shown in Figure 9.3.9. You can select the required file, and click **Open** to load it as an Output Structure

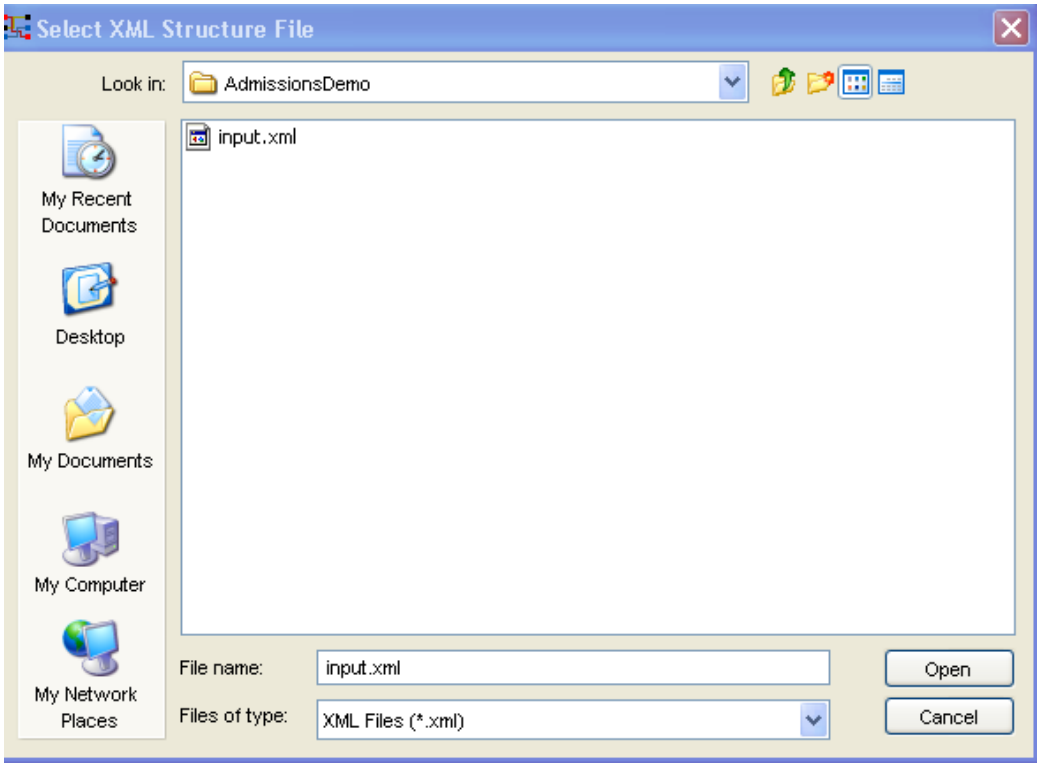


Figure 9.3.10: Selecting an Output XML File

4. The selected XML document is displayed in the Output Structure panel, as shown in Figure 9.3.11.

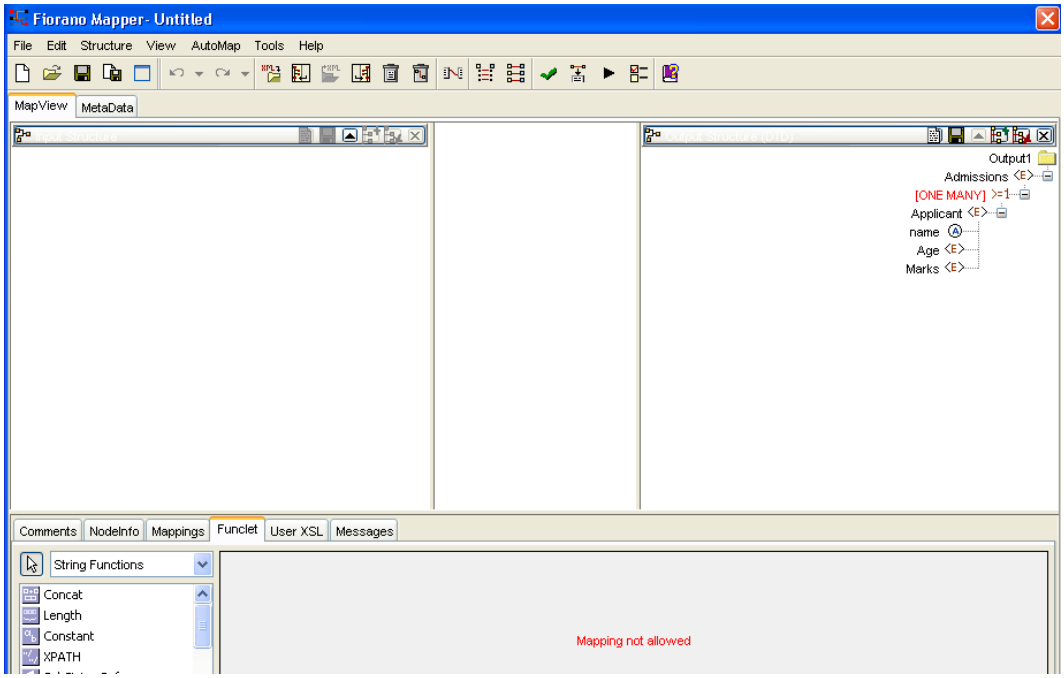


Figure 9.3.11: The specified XML File is displayed in the Output Structure panel

9.3.4.2 Loading a CSV Output Structure

A comma separated values or CSV file is a text file which is separated by delimiter, usually a comma. It is a commonly used Output Structure for many event processes.

To load a CSV Output Structure, perform the following steps:

1. Click the **Load Output Structure** icon from the tool bar, and select **CSV** from the **Select Output Structure Type** dialog box, click **OK.** A **Load Output CSV Structure** dialog box is displayed as shown in Figure 9.3.12.

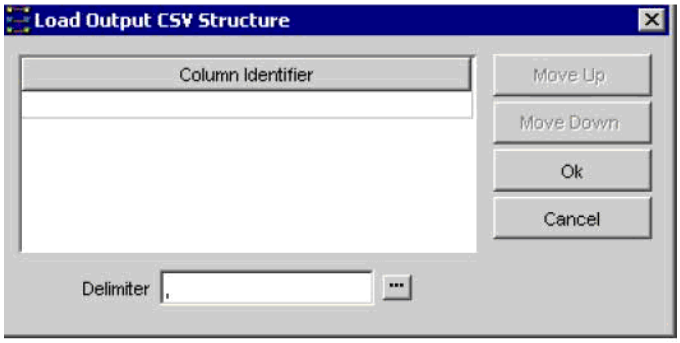


Figure 9.3.12: Entering the CSV fields

2. The **Load Target CSV Structure** dialog box is displayed, as shown in Figure 9.3.13. You can specify the delimiter to be used in the output structure in the **Delimiter** text box at the bottom of this dialog. The **Column Identifier** table lists the column names for the Output CSV Structure. Type the name of a column in this table and press **ENTER**.

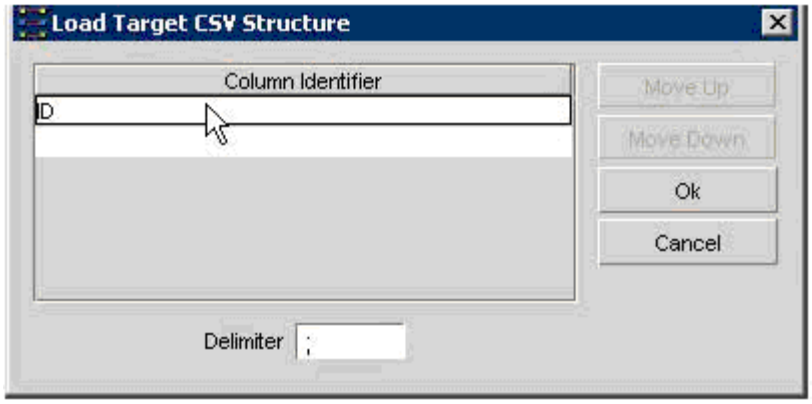


Figure 9.3.13: Entering the CSV Column Identifiers

3. A blank row is added to the **Column Identifier**, as shown in Figure 9.3.14. Enter the required Column Identifier names to define the CSV structure.

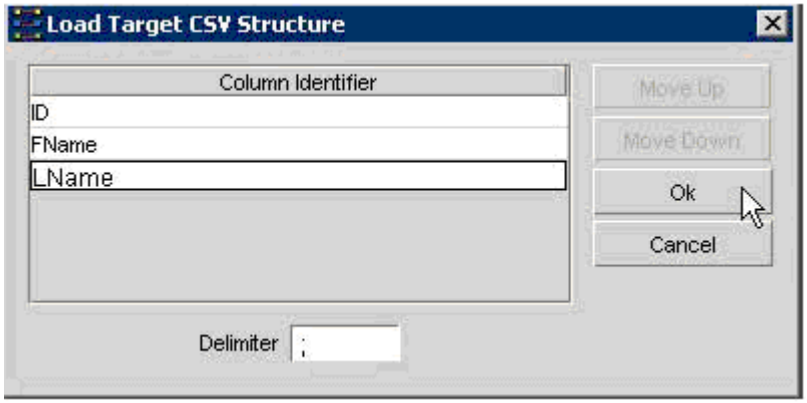


Figure 9.3.14: Defining the Output CSV Structure

4. Once you have defined the Output CSV Structure, click **OK.**

The CSV structure is loaded in the Output Structure Panel, as shown in Figure 9.3.15.

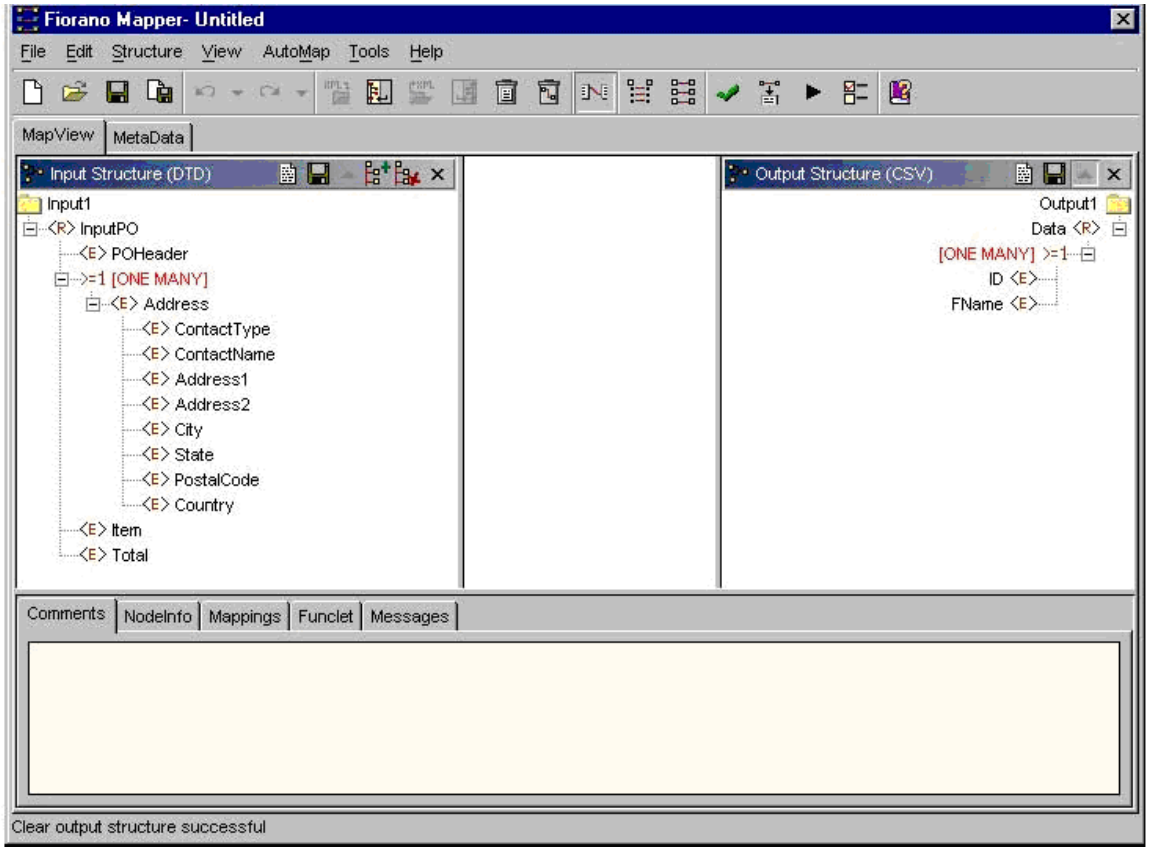


Figure 9.3.15: A CSV Output Structure is loaded

9.3.5 Viewing the Output Structure Source

You can view the source of Output Structure that you have defined by:

1. Selecting the **View Source** icon in the Output Structure panel.
2. This displays the **Source Viewer** window as shown in Figure 9.3.16. Click on **Close** to close the Source Viewer.

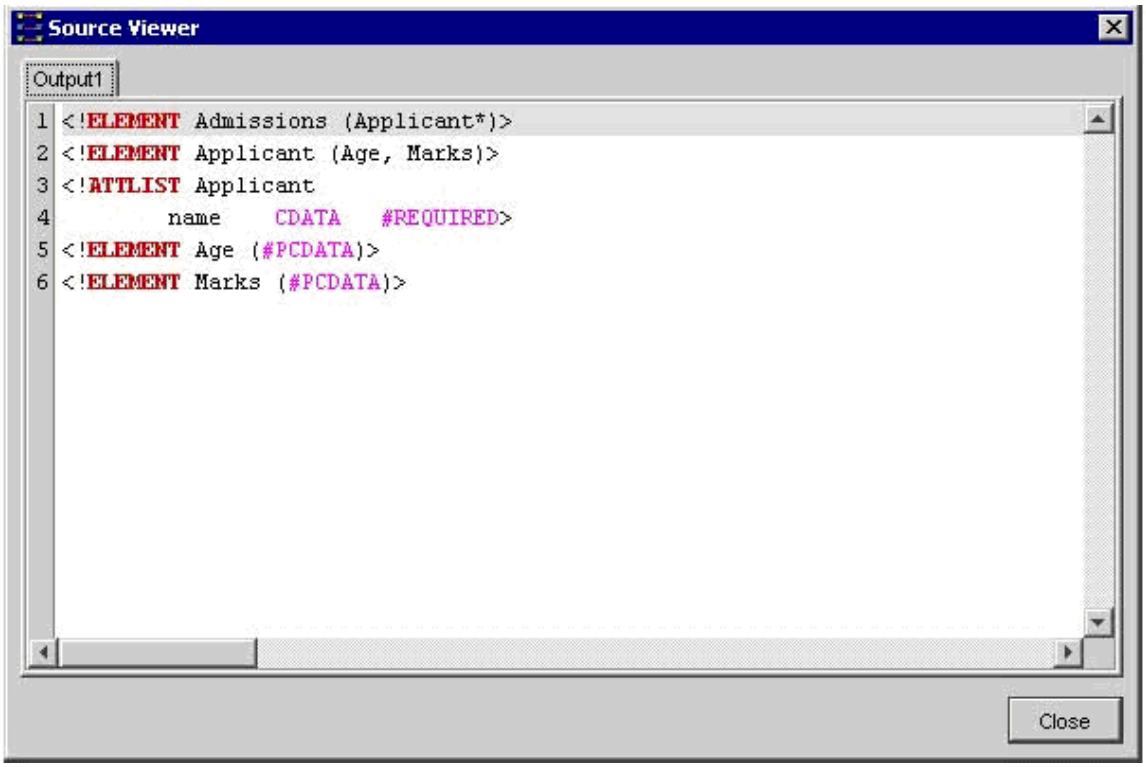


Figure 9.3.16: Viewing the source of the Output Structure

9.3.6 Clearing the Output Structure

To clear the Output Structure, select the Clear output structure icon in the Output Structure panel.

9.4 Working with the Visual Expression Builder

Fiorano Mapper provides an easy to use graphical user interface – the Visual Expression Builder, for building simple or complex expressions using several predefined functions. All this can be done by performing simple drag-n-drop of required functions, input nodes and connecting them visually.

To switch to the Visual Expression Builder:

1. Right-click any node of the output structure.
2. Select the **Funclet Wizard** option.

Alternatively,

1. Select any node of the output structure.
2. Click the **Funclet** tab.

The Visual Expression Builder consists of two areas:

- Function palette
- Funclet easel

9.4.1 Function Palette

The Function palette contains all the functions logically grouped into different categories:

- Arithmetic Functions
- String Functions
- Boolean Functions
- Control Functions
- Advanced Functions
- JMS Message Functions
- Date-Time Functions
- NodeSet Functions
- Math Functions
- Conversion Functions
- Look-up Functons
- User defined functions

9.4.1.1 Arithmetic Functions

Fiorano Mapper provides several Arithmetic functions to ++work with numbers and nodes. This section describes these functions.

Addition

Visual representation

Description: This function calculates and returns the sum of two nodes or numbers.

Input: Two number constants or input structure nodes.

Output: Number

Subtraction

Visual representation

Description: This function subtracts the values of two numbers or nodes.

Input: Two number constants or input structure nodes.

Output: Number

Division

Visual representation

Description: This function obtains and returns the quotient after dividing the values of two nodes or numbers.

Input: Two number constants or input structure nodes.

Output: Number

Modulo

Visual representation

Description: This function returns the remainder after dividing the values of the two nodes or numbers.

Input: Two number constants or input structure nodes.

Output: Number

Multiplication

Visual representation

Description: This function multiplies the values of two nodes or numbers.

Input: Two number constants or input structure nodes.

Output: Number

Floor

Visual representation

Description: This function rounds off the value of the node or number to the nearest lower integer.

Input: A number constant or an input structure node.

Output: Number

Example: The number 3.3 is floored to 3.

Ceiling

Visual representation

Description: This function rounds off the value of the node or number to the nearest higher integer.

Input: A number constant or an input structure node.

Output: Number

Example: The number 25.6 is ceilinged to 26.

Round

Visual representation

Description: This function rounds off the value of the preceding node or a number to the nearest integer.

Input: A number constant or an input structure node.

Output: Number

Example: The number 4.8 is rounded off to 5 and 4.2 is rounded off to 4.

Number Function

Visual representation

Description: This function converts the input to a number according to the XPath specifications.

Input: A number constant or an input structure node.

Output: Number based on the following rules:

- Boolean true is converted to 1, and false is converted to 0.
- A node-set is first converted to a string and then converted in the same way as a string argument.

- A string that consists of optional whitespace followed by an optional minus sign followed by a number followed by whitespace is converted to the IEEE 754 number that is nearest to the mathematical value represented by the string; any other string is converted to NaN.
- An object of a type other than the four basic types is converted to a number in a way that is dependent on that type.

9.4.1.2 Math Functions

Absolute

Visual representation

Description: This function returns the absolute (non-negative) value of a number.

Input: Number

Output: The absolute value of the input

Sin

Visual representation

Description: This function returns the Sine value of the input. The input is in radians.

Input: A number in radians.

Output: The Sine value of the input.

Cos

Visual representation

Description: This function returns the Cosine value of the input. The input is in radians.

Input: A number in radians

Output: The Cosine value of the input

Tan

Visual representation

Description: This function returns the Tan value of the input. The input is in radians.

Input: A number in radians.

Output: The Tan value of the input.

Arc sine

Visual representation

Description: This function returns the Arc Sine value or the Sine Inverse value of the input. The output is in radians.

Input: Number

Output: The Sine Inverse value of the input in radians.

Arc cos

Visual representation

Description: This function returns the Arc Cosine value or the Cosine Inverse value of the input. The output is in radians.

Input: Number

Output: The Cosine Inverse value of the input in radians.

Arc tan

Visual representation

Description: This function returns the Arc Tan value or the Tan Inverse value of the input. The output is in radians.

Input: Number

Output: The Tan Inverse value of the input in radians.

Exponential

Visual representation

Description: This function returns the exponential value of the input.

Input: Any number

Output: The exponential value the input.

Power

Visual representation

Description: This function returns the value of a first input raised to the power of a second number.

Input: Two numbers: the first number is the base, and the second number is the power.

Output: A number that is the result of the above described calculation or NaN in case the value could not be calculated.

Random

Visual representation

Description: This function returns a random number between 0 and 1.

Input: No input

Output: A number between 0 and 1.

Sqrt

Visual representation

Description: This function returns the square root of the input value

Input: A number

Output: A number that is the square root of the input value.

Log

Visual representation

Description: This function returns the natural logarithm (base e) of a numerical (double) value.

Input: A positive numerical value.

Output: The natural logarithm (base e) of the input - a numerical (double) value.

Special cases:

If the argument is NaN or less than zero, the result is NaN.

If the argument is positive infinity, the result is positive infinity.

If the argument is positive zero or negative zero, the result is negative infinity.

9.4.1.3 String Functions

Fiorano Mapper has several string functions. All the functions accept Unicode strings and are case-sensitive. This section covers the string functions.

XPath

Visual representation

Description: This function evaluates the specified XPath expression and returns the result.

Input: For elements within the first structure of the document, specify the XPath as:

/<root element>/<child element>

Example/school/student

For elements within the second structure onwards, specify the XPath as:

document('<structure name>')/<root element>/<child element>

Example:document('input2')/school/student

Output: Result of the XPath expression.

Concat

Visual representation

Description: This function accepts two or more string arguments and joins them in a specified sequence to a form a single concatenated string.

Input: Two or more string constants or input structure nodes.

Output: A concatenated string.

Example: Concat ("abc", "xyz") returns "abcxyz".

Constant

Visual representation

Description; This function creates a constant building block with a string literal.

Input: String

Output: String

Length

Visual representation

Description: This function returns the length of a string.

Input: A string constant or an input structure node.

Output: Number

Example: Length ("abcd") returns 4

Normalize_Space

Visual representation

Description: This function accepts a string as an argument and removes leading, trailing, and enclosed spaces in the specified string. The unnecessary white spaces within the string are replaced by a single white space character.

Input: A string or an input structure node.

Output: String with no whitespace before, after, or within it.

Example: Normalize_Space(" MapperTool ") returns "Mapper Tool".

White spaces before and after the string is removed and the white spaces between "Mapper" and "Tool" are replaced by a single blank space.

SubString-After

Visual representation

Description This function accepts two strings as arguments. The first string is the source and the second input string is the string pattern. It returns that part of the first input string that follows the string pattern.

Input: Two string constants or input structure nodes.

Output: String

Example: SubString-After('abcde','bc') returns "de"

SubString-Before

Visual representation

Description: This function accepts two strings as arguments. The first string is the source and the second is the string pattern. The function returns that part of the first input string that precedes the string pattern specified as the second argument to the function.

Input: Two string constants or input structure nodes.

Output: String

Example: SubString-Before('abcde', 'cd') returns 'ab'

SubString-Offset

Visual representation

Description: This function accepts two string constants as argument. The first string is the source and the second string is a numerical value that specifies the offset. The output is that part of the source string which starts from the offset specified as the second argument to the function.

Input: Two string constants or input structure nodes.

Output: String

Example: SubString-Offset(‘abcde’, 3) returns "cde"

SubString-Offset-Length

Visual representation

Description: This function accepts three arguments. The first argument is the source string, the second and third arguments are numerical that specify the offset and the size of the output substring respectively. The output is a substring which starts from the offset specified as the second argument to the function. The number of characters that need to be obtained is specified as the third argument.

Input: Two string constants or input structure nodes and a number.

Output: String

Example: SubString-Offset-Length(‘abcde’, 2, 3) returns "bcd"

9.4.1.4 Control Function

The following Control functions are available in Fiorano Mapper:

If-Then-Else

Visual representation

Description: This function accepts an input value. The first input is a Boolean value and the second and third are string constants. Based on the Boolean value, the function returns the output. If the Boolean value specified in the first input is TRUE, then the function returns the second input string else it returns the third input string.

Input: Boolean value and a string, an optional string in the same sequence.

Output: The second input string or third input string (if present) depending on the first input Boolean value.

Sort Function

Visual representation

Description: This function accepts two inputs. The first input is a set of nodes and the second input is the value of the nodes. The function sorts the nodes in its first input based on the second input.

Input: Sort (nodes, value)

Output: Sorted nodes as Loop Source

Filter Function

Visual representation

Description: This function accepts two arguments. The first argument is a set of node and the second argument is a Boolean value. It filters out and returns the nodes for which the second input value is TRUE.

Input: Filter (node set, bool)

Output: Nodes for which the second input value is true as Loop Source.

9.4.1.5 Conversion Functions

Fiorano Mapper consists of several Conversion functions to convert numerical from one format to the other. These functions are covered in this section.

Decimal

Visual representation

Description: Converts the first input value having a base that is specified by the second input value to a decimal number.

Input: Two numbers: The first input value is the number to be converted to decimal, and the second input value specifies the base of the first input value.

Output: Number in base 10.

Hex

Visual representation

Description: Converts a decimal number to a hexadecimal (base 16) number.

Input: Decimal number

Output: Hexadecimal (base 16) number

Octal

Visual representation

Description: Converts a decimal number to an octal (base 8) number.

Input: Decimal number

Output: Octal (base 8) number

Binary

Visual representation

Description: Converts a decimal number to a binary (base 2) number.

Input: Decimal number

Output: Binary (base 2) number

Radians

Visual representation

Description: Converts a value in Degrees to a value in Radians.

Input: Number

Output: Number

Degrees

Visual representation

Description: Converts a value in Radians to a value in Degrees.

Input: Number

Output: Number

ChangeBase

Visual representation

Description: The ChangeBase function is used to change a number from one base to another. This function accepts three arguments.

1. **num-** the number to be changed
2. **fromBase-** base of the given number
3. **toBase**- base to which number should be converted

Input: Number

Output: Number

9.4.1.6 Advanced Functions

Fiorano Mapper provides a number of advanced functions. This section explains all these functions.

CDATA Function

Visual representation

Description: This function accepts a string as an argument and specifies the character data within the string.

Input: String argument or input structure node.

Output: Input string or node text enclosed within the CDATA tag.

Example: CDATA ("string") returns <![CDATA[string]]>

Position

Visual representation

Description: This function is available for the RDBMS-Update or RDBMS-Delete Output structures only and returns the current looping position.

Input: None

Output: The position of the element in the parent tree.

Example: In an XML tree that has three elements, Position() returns

0 for the first element

1 for the second, and

2 for the third.

Format-Number

Visual representation

Description: This function converts the first argument to a string, in the format specified by the second argument. The first argument can be a real number or an integer, and can be positive or negative.

Input: Two values: The first input is a number, and the second, a string of special characters that specifies the format. These special characters are listed in the following table:

Representation	Signifies	Example
#	a digit [0-9]	###
.	the decimal point	###.##
,	digit separator	###, ###.##
0	leading and trailing zeros	000.0000
%	inserts a percentage sign at the end	###.00%
;	a pattern separator	##.00;##.00

The format string is created by using these characters in any order.

Output: String with the number in the specified format.

Node-Name

Visual representation

Description: This function accepts an element or attribute and returns the name of the particular element or attribute.

Input: A single element or attribute of any type

Output: A string

Count

Visual representation

Description: This function accepts an element or attribute and returns the number of instances of a particular element or attribute.

Input: A single element or attribute of any type

Output: A number

Deep-Copy

Visual representation

Description: Copies the current node completely including the attributes and sub-elements.

Input: An Input structure node

Output: All the contents of the Input structure node – including its attributes and sub-elements.

Param

Visual representation

Description: This function is used to access the runtime parameters by its name. Various properties of Tifosi Document (such as header, message, and attachments) are available as runtime parameters at runtime. The names of these parameters follow the convention given below:

Header Properties	_TIF_HEADER_<HEADERNAME>
Message (text)	_TIF_BODY_TEXT_
Message (byte)	_TIF_BODY_BYTE_
Attachment	_TIF_ATTACH_<NAME>

Input: Name of the parameter

Output: Value of the parameter specified

9.4.1.7 Date-Time Functions

Date-Time functions include:

Date

Visual representation

Description: The Date function returns the date part in the input date-time string or the current date if no input is given. The date returned format is: CCYY-MM-DD

If no argument is given or the argument date/time specifies a time zone, then the date string format must include a time zone, either a Z to indicate Coordinated Universal Time or a + or - followed by the difference between the difference from UTC represented as hh:mm. If an argument is specified and it does not specify a time zone, then the date string format must not include a time zone.

Input: Optionally, a string that can be converted to a date (the string should have the date specified in the following format: CCYY-MM-DD)

Output: A date in the format: CCYY-MM-DD

DateTime

Visual representation

Description: This function returns the current date and time as a date/time string in the following format:

CCYY-MM-DDThh:mm:ss

Where,

CC is the century

YY is the year of the century

MM is the month in two digits

DD is the day of the month in two digits

T is the separator between the Date and Time part of the string

hh is the hour of the day in 24-hour format

mm is the minutes of the hour

ss is the seconds of the minute

The output format includes a time zone, either a Z to indicate Coordinated Universal Time or a + or - followed by the difference between the localtime from UTC represented as hh:mm.

Input: This function has no input.

Output: The current date-time in the following format: CCYY-MM-DDThh:mm:ss as described above.

DayAbbreviation

Visual representation

Description: This function returns the abbreviated day of the week from the input date string. If no argument is given, then the current local date/time is used as the default argument.

Input: Optionally, a date-time string

Output: The English day of the week as a three-letter abbreviation: 'Sun', 'Mon', "Tue', 'Wed', 'Thu', 'Fri', or 'Sat'.

DayInMonth

Visual representation

Description: This function returns the day of a date as a number. If no argument is given, then the current local date/time is used as the default argument.

Input: A date-time string in any of the following formats:

CCYY-MM-DDThh:mm:ss

CCYY-MM-DD

--MM-DD

---DD

If no input is given, then the current local date/time is used.

Output: A number which is the day of the month in the input string.

DayInWeek

Visual representation

Description: This function returns the day of the week given in a date as a number. If no argument is given, then the current local date/time is used the default argument.

Input: A date string in any of the following formats:

CCYY-MM-DDThh:mm:ss

CCYY-MM-DD

Output: The day of the week as a number - starting with 1 for Sunday, 2 for Monday and so on up to 7 for Saturday. If the date/time input string is not in a valid format, then NaN is returned.

DayInYear

Visual representation

Description: This function returns the day of a date as a day number in a year starting from 1.

If no argument is given, then the current local date/time, as returned by date-time is used the default argument.

Input: Optionally, a date string in any of the following formats:

CCYY-MM-DDThh:mm:ss

CCYY-MM-DD

Output: A number representing the day in a year.

Example: The DayInYear for 2003-01-01 returns 1, where as for 2003-02-01 it returns 32.

DayName

Visual representation

Description: This function returns the full day of the week for a date. If no argument is given, then the current local date/time is used the default argument.

Input: Optionally, a date string in any of the following formats:

CCYY-MM-DDThh:mm:ss

CCYY-MM-DD

Output: An English day name: 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday' or 'Friday'.

DayOfWeekInMonth

Visual representation

Description: This function returns the occurrence of that day of the week in a month for a given date as a number. If no argument is given, then the current local date/time is used as the default argument.

Input: Optionally, a date string in any of the following formats:

CCYY-MM-DDThh:mm:ss

CCYY-MM-DD

Output: A number that represents the occurrence of that day-of-the-week in a month.

Example: DayOfWeekInMonth returns 3 for the 3rd Tuesday in May.

HourInDay

Visual representation

Description: This function returns the hour of the day as a number. If no argument is given, then the current local date/time is used as the default argument.

Input: A date string in any one of the following formats:

CCYY-MM-DDThh:mm:ss

hh:mm:ss

If the date/time string is not in one of these formats, then NaN is returned.

Output: The hour of the day or NaN if the argument is not valid.

LeapYear

Visual representation

Description: This function returns TRUE if the year given in a date is a leap year. If no argument is given, then the current local date/time is used as the default argument.

Input: Date string in any of the following formats:

CCYY-MM-DDThh:mm:ss

CCYY-MM-DD

CCYY-MM

CCYY

If the date/time string is not in one of these formats, then `NaN` is returned.

Output: Boolean value (TRUE/FALSE)

MinuteInHour

Visual representation

Description: This function returns the minute of the hour as a number. If no argument is given, then the current local date/time is used as the default argument.

Input: Optionally, a date string in any of the following formats:

CCYY-MM-DDThh:mm:ss

hh:mm:ss

Output: The minute of the hour or NaN if the argument is not valid.

MonthAbbreviation

Visual representation

Description: This function returns the abbreviation of the month of a date. If no argument is given, then the current local date/time is used as the default argument.

Input: Date string in any of the following formats:

CCYY-MM-DDThh:mm:ss

CCYY-MM-DD

CCYY-MM

--MM--

OutputThree-letter English month abbreviation: 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov' or 'Dec'.If the date/time string argument is not in valid, then an empty string ('') is returned.

MonthInYear

Visual representation

Description: This function returns the month of a date as a number. The counting of the month starts from 0. If no argument is given, the current local date/time is used as the default argument.

Input: Date string in any of the following formats:

CCYY-MM-DDThh:mm:ss

CCYY-MM-DD

CCYY-MM

--MM--

--MM-DD

If the date/time string is not valid, then NaN is returned.

Output: A number representing the month in a year.

Example: 0 for January, 1 for February, 2 for March and so on.

MonthName

Visual representation

Description: This function returns the full name of the month of a date. If no argument is given, then the current local date/time is used as the default argument.

Input: Optionally, a date string in any of the following formats:

CCYY-MM-DDThh:mm:ss

CCYY-MM-DD

CCYY-MM

--MM--

OutputThe English month name: 'January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November' or 'December'.If the date/time string is not valid, then an empty string ('') is returned.

SecondInMinute

Visual representation

Description: This function returns the second of the minute as a number. If no argument is given, then the current local date/time is used as the default argument.

Input: Optionally, a date string in any of the following formats:

CCYY-MM-DDThh:mm:ss

hh:mm:ss

Output: The second in a minute as a number. If the date/time string is not valid, then NaN is returned.

Time

Visual representation

Description: This function returns the time specified in the date/time string that is passed as an argument. If no argument is given, the current local date/time is used as the default argument. The date/time format is basically CCYY-MM-DDThh:mm:ss.

If no argument is given or the argument date/time specifies a time zone, then the time string format must include a time zone, either a Z to indicate Coordinated Universal Time or a + or - followed by the difference between the difference from UTC represented as hh:mm. If an argument is specified and it does not specify a time zone, then the time string format must not include a time zone.

Input: Optionally, a date/time string in the following format:

CCYY-MM-DDThh:mm:ss

Output: The time from the given date/time string in the following format:

hh:mm:ss

If the argument string is not in this format, this function returns an empty string ('').

WeekInYear

Visual representation

Description: This function returns the week of the year as a number. If no argument is given, then the current local date/time is used as the default argument. Counting follows ISO 8601 standards for numbering: week 1 in a year is the week containing the first Thursday of the year, with new weeks beginning on a Monday.

Input: Optionally, a date/time string in any of the following format:

CCYY-MM-DDThh:mm:ss

CCYY-MM-DD

Output: The week of the year as a number. If the date/time string is not in one of these formats, then NaN is returned.

Year

Visual representation

Description: This function returns the year of a date as a number. If no argument is given, then the current local date/time is used as a default argument.

Input: Optionally, a date/time string in any of the following format:

CCYY-MM-DDThh:mm:ss

CCYY-MM-DD

CCYY-MM

CCYY

Output If the date/time string is not in one of these formats, then NaN is returned.

9.4.1.8 SQL Functions

The following are the SQL functions available for defining mappings to RDBMS-Update or an RDBMS-Delete query:

Number-Constant

Visual representation

Description: This function inserts a number constant into a SQL query.

Input: None

Output: A numeric value

String-Constant

Visual representation

Description: This function inserts a string constant into a SQL query.

Input: None

Output: A string value

AND

Visual representation

Description: This function combines two conditions in the where clause of a SQL statement. It takes two conditions as input and performs a logical AND on them. The Boolean value returned is true only if both the conditions are true.

Input: Two nodes or sub-expressions created with the Visual Expression Builder

Output: A Boolean value (TRUE/FALSE)

OR

Visual representation

Description: This function is used for combining two conditions in the where clause of an RDBMS Update or RDBMS Delete output structure.

Input: Two nodes or sub-expressions created with the Visual Expression Builder.

Output: A Boolean value (TRUE/FALSE)

LIKE

Visual representation

Description: Compares the first expression to the pattern specified by the second expression.

Input: Two nodes or sub-expressions created with the Visual Expression Builder.

Output: A Boolean value (TRUE/FALSE)

PLUS

Visual representation

Description: Adds/ concatenates two expressions or nodes.

Input: Two nodes or sub-expressions created with the Visual Expression Builder.

Output: A string or number generated by concatenating/ adding the two input expressions.

MINUS

Visual representation

Description: Subtracts the second input from the first input.

Input: Two nodes or sub-expressions created with the Visual Expression Builder.

Output: A number generated by subtracting the second input from the first.

BETWEEN

Visual representation

Description: Checks whether the given column value lies between the specified two arguments.

Input: Three inputs:

1. A table column
2. A node or sub-expression
3. A node or sub-expression

Output: A Boolean value (TRUE/FALSE)

IS NULL

Visual representation

Description: Used in the Where clause to verify whether a column is null.

Input: A Table column

Output: A Boolean value (True/False)

IS NOT NULL

Visual representation

Description: Used in the Where clause to verify whether a column is not null.

Input: A Table column

Output: A Boolean value (TRUE/FALSE)

Null

Visual representation

Description: The NULL function represents the null values in SQL. This function can be used to set a column value to null.

Input: None

Output: Null value

=

Visual representation

Description: This function compares two input values in the where clause of a SQL statement. The Boolean value returned is true only if both the input values are equal.

Input: Two table columns

Output: A Boolean value (TRUE/FALSE)

NOT LIKE

Visual representation

Description: This function compares the first expression to the pattern specified by the second expression.

Input; Two nodes or sub-expressions created with the Visual Expression Builder.

Output: A Boolean value (TRUE/FALSE)

<>

Visual representation

Description: This function compares two input values in the where clause of a SQL statement. The Boolean value returned is TRUE only if both the input values are not equal.

Input: Two table columns

Output: A Boolean value (TRUE/FALSE)

>

Visual representation

Description This function compares two input values in the where clause of a SQL statement. The Boolean value returned is TRUE only if the first input value is greater than the second input value.

Input: Two table columns

Output: A Boolean value (TRUE/FALSE)

<

Visual representation

Description: This function compares two input values in the where clause of a SQL statement. The Boolean value returned is TRUE only if the first input value is lesser than the second input value.

Input: Two table columns

Output: A Boolean value (TRUE/FALSE)

>=

Visual representation

Description: This function compares two input values in the where clause of a SQL statement. The Boolean value returned is TRUE only if the first input value is greater than or equal to the second input value.

Input: Two table columns

Output: A Boolean value (TRUE/FALSE)

<=

Visual representation

Description: This function compares two input values in the where clause of a SQL statement. The Boolean value returned is TRUE only if the first input value is lesser than or equal to the second input value.

Input: Two table columns

Output: A Boolean value (TRUE/FALSE)

IN

Visual representation

Description: The In function is used in the where clause of a SQL statement. This function accepts two expressions as inputs and checks if the first expression is contained within the second expression.

Input: This function accepts two inputs

1. First input is table column name
2. Second input is a list of comma-separated values

Output: A Boolean value (TRUE/FALSE)

9.4.1.9 NodeSet Functions

SUM

Visual representation

Description: The Sum function sums all numbers in selected nodes.

Input: A nodes that has numerical values only.

Output: The sum of all the nodes. If any of the input nodes is not valid, a NaN value is returned.

DIFFERENCE

Visual representation

Description: The difference function returns the difference between the two node sets that are, in the node set passed as the first argument and the node that are not in the node set passed as the second argument.

Input: Two node sets

Output: Node set

DISTINCT

Visual representation

Description: The distinct function returns a subset of the nodes contained in the node-set passed as the first argument. Specifically, it selects a node N if there is no node in a given node-set that has the same string value as N, and that precedes N in the document order.

Input: A node set

Output: A node

HAS SAME NODE

Visual representation

Description: The has-same-node function returns TRUE if the node set passed as the first argument shares any nodes with the node set passed as the second argument. If there are no nodes that are in both node sets, then it returns FALSE.

Input: Two node sets

Output: Boolean value (TRUE or FALSE)

INTERSECTION

Visual representation

Description The intersection function returns a node set containing the nodes that are within both the node sets passed as arguments to it.

Input: Two node sets

Output: Node set

LEADING

Visual representation

Description: The leading function returns the nodes in the node set passed as the first argument that precede, in document order, the first node in the node set passed as the second argument. If the first node in the second node set is not contained in the first node set, then an empty node set is returned. If the second node set is empty, then the first node set is returned.

Input: Two node sets

Output: Node Set

TRAILING

Visual representation

Description: The trailing function returns the nodes in the node set passed as the first argument that follow, in document order, the first node in the node set passed as the second argument. If the first node in the second node set is not contained in the first node set, then an empty node set is returned. If the second node set is empty, then the first node set is returned.

Input: Two node sets

Output: Node set

HIGHEST

Visual representation

Description: The highest function returns the nodes in the node set whose value is the maximum (numerical) value for the node set.

- A node has this maximum value if the result of converting its string value to a number as if by the number function is equal to the maximum value, where the equality comparison is defined as a numerical comparison using the = operator.
- If any of the nodes in the node set has a non-numeric value, this function returns an empty node set.

Input: A node set

Output: A node set

LOWEST

Visual representation

Description: The lowest function returns the nodes in the node set whose value is the minimum (numerical) value for the node set.

- A node has this minimum value if the result of converting its string value to a number as if by the number function is equal to the minimum value, where the equality comparison is defined as a numerical comparison using the = operator.
- If any of the nodes in the node set has a non-numeric value, this function returns an empty node set.

Input: A node set

Output: A node set

MINIMUM

Visual representation

Description: The minimum function returns the node with the minimum numerical value within the given node-set. If the node set is empty, or if any of the nodes in the node set has non-numeric value, then NaN is returned.

Input: A node set

Output: A numerical value

MAXIMUM

Visual representation

Description: The maximum function returns the node with the maximum numerical value within the given node set. If the node set is empty, or if any of the nodes in the node set has non-numeric value, then NaN is returned.

Input: A node set

Output: A numerical value

9.4.1.10 Boolean functions

The following boolean (logical) functions are available in Mapper Tool:

Symbol	Function	Description
=	Equal	True if both inputs are equal.
!=	Not Equal	True if both inputs are not equal
>	Greater than	True if the first input is greater than the second input.
<	Less than	True if the first input is less than the second input.
>=	Greater than or Equal	True if the first input is greater than or equal to the second input.
<=	Less than or Equal	True if the first input is less than or equal to the second input.
AND	AND	Logical AND of the two inputs (the inputs must be outputs of logical building blocks only).
OR	OR	Logical OR of the two inputs (the inputs must be outputs of logical building blocks only).
NOT	NOT	Logical inverse of the input (the input must be the output of logical building block only).
BOOL	boolean(object)	Converts its argument to a boolean according to the XPath specifications, as follows: – a number is true if and only if it is neither positive or negative zero nor NaN. – a node-set is true if and only if it is non-empty – a string is true if and only if its length is non-zero an object of a type other than the four basic types is converted to a boolean in a way that is dependent on that type. – – IsNumber-IsNumber( )-Returns a boolean (true/ false) indicating if the input value is a number

AND function

Symbol: AND

Description: This function accepts two boolean expressions as arguments and performs a logical conjunction on them. If both expressions evaluate to TRUE, the function returns TRUE. If either or both expressions evaluate to FALSE, the function returns FALSE.

Input: AND (boolean AND boolean)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of Chat.dtd as Input Structure and Valid.dtd as Output Structure. Suppose we want to filter out mails that do not have message body and the email address is not equal to admin@nobody.com. That is, we want that the isValid node of the Output Structure takes the value true if the length of the Message node of the Input Structure is not equal to zero and the value of the Email node is equal to **admin@nobody.com**. Therefore,

1. Load Input Structure and Output Structure.
2. Map the Message node and Email node of Input Structure to the isValid node of the Output Structure.
3. Invoke the Function Wizard by right clicking on the isValid node.
4. The Function Easel opens with the existing mappings.
5. Place the BOOL node on the Function Easel.

   Link the output of the Message node to the input of the BOOL node, as shown in Figure 9.4.1.

Figure 9.4.1: Linking Message and BOOL nodes

6. Place a Constant node on the Function Easel, and set its value equal to admin@nobody.com.
7. Place a = node on the Function Easel.

   Link the outputs of the Email node and Constant node to the inputs of the = node, as shown in Figure 9.4.2.

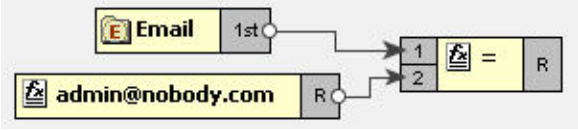


Figure 9.4.2: Linking the Email and Constant node outputs

8. Place an AND node on the Function Easel.
9. Link the outputs of the BOOL node and = node to the inputs of the AND node.

   Also, link the output of the AND node to the input of the isValid node, as shown in Figure 9.4.3.

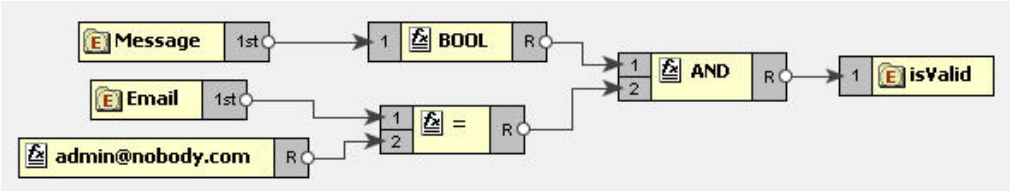


Figure 9.4.3: Linking the AND and = node outputs

10. This completes the desired mappings.

BOOL

Symbol: BOOL

Description:

This function converts its argument to a boolean according to the XPath specifications which are as follows:

- A number is TRUEif and only if it is neither positive or negative zero nor NaN.

- A node-set is TRUE if and only if it is non-empty.

- A string is TRUE if and only if its length is non-zero.

- An object of a type other than the four basic types is converted to a boolean in a way that is dependent on that type.

Input: BOOL (Object)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of Chat.dtd as Input Structure and Valid.dtd as Output Structure. Suppose we want to filter out mails that do not have message body. That is, we want that the isMessageExist node of the Output Structure takes the value true if the length of the Message node of the Input Structure is not equal to zero. The BOOL function returns true for a string of length non-zero. Therefore,

1. Load Input Structure and Output Structure.
2. Map the Message node of Input Structure to the isMessageExist node of the Output Structure.
3. Invoke the Function Wizard by right clicking on the isMessageExist node.
4. The Function Easel opens with the existing mappings.
5. Place the BOOL node on the Function Easel.

   Link the output of the Message node to the input of the BOOL node, as shown in Figure 9.4.4.

Figure 9.4.4: Linking Message and BOOl nodes

Link the output of the BOOL node to the input of the isMessageExist node, as shown in Figure 9.4.5.

Figure 9.4.5: Linking BOOL and IsMessageExist nodes

6. This completes the desired mappings.

Equal

Symbol: =

Description: This function returns TRUE if both the inputs are equal.

Input: = (Object = Object)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of Chat.dtd as Input Structure and Valid.dtd as Output Structure. Suppose we want to filter mails coming from a particular email address. That is, we want that the isFromAdmin node of the Output Structure takes the value true if the Email node of the Input Structure has the email address as admin@nobody.com. Then,

1. Load Input Structure and Output Structure.
2. Map the Email node of Input Structure to the isFromAdmin node of the Output Structure.
3. Invoke the Function Wizard by right clicking on the isFromAdmin node.
4. The Function Easel opens with the existing mappings.

   Now place a Constant building block on the Function Easel and set its value equal to **admin@nobody.com,** as shown in Figure 9.4.6.

Figure 9.4.6: Setting Constant building block value to admin@nobody.com

5. Now place a = node on the Function Easel.
6. Link the outputs of the Email node and Constant node to the inputs of the = node.

   Link the output of the = node to the input of the isFromAdmin node, as shown in Figure 9.4.7.

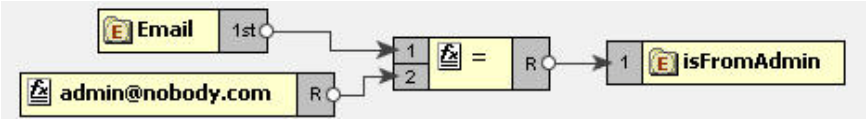


Figure 9.4.7: Linking = and isFromAdmin **node**

7. This completes the desired mappings.

Less Than

Symbol: <

Description: This function returns TRUE if the first input is less than the second input value.

Input: < (Number < Number)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of Numbers.dtd as Input Structure and Results.dtd as Output Structure. Suppose we want that Result node of Output Structure should have the value true if the value of Number1 node is less than the value of the Number2 node of the Input Structure. Then,

1. Load Input Structure and Output Structure.
2. Map the Number1 and Number2 nodes of Input Structure to the Result node of the Output Structure.
3. Invoke the Function Wizard by right clicking on the Result node.
4. The Function Easel shows the existing mappings.

   Place the < node on the Function Easel, as shown in Figure 9.4.8.

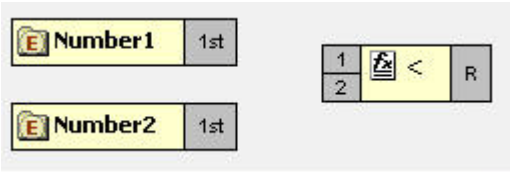


Figure 9.4.8: Placing a node on the Function Easel

5. Link the outputs of the Number1 node and Number2 node to the inputs of the < node.

   Also, link the output of the < node to the input of the Result node, as shown in Figure 9.4.9.

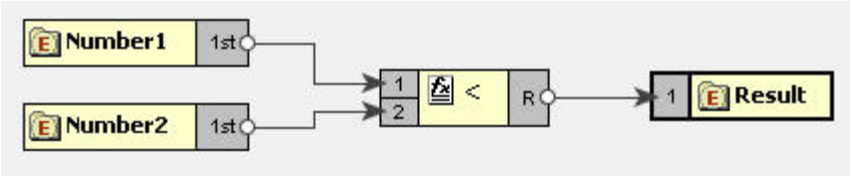


Figure 9.4.9: Linking < and Result node

6. This completes the desired mappings.

Greater than

Symbol: >

Description: This function returns TRUE if the first input is greater than the second input value.

Input: > (Number > Number)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of TotalMarks.dtd as Input Structure and Result.dtd as Output Structure. Suppose we want that the value of the PassStatus node is true if the value of the TotalMarks node of the Input Structure is greater than a constant value 150. Then,

1. Load Input Structure and Output Structure.
2. Map the TotalMarks node of Input Structure to the PassStatus node of the Output Structure.
3. Invoke the Function Wizard by right-clicking on the PassStatus node.
4. The Function Easel opens with the existing mappings.
5. Now place a Constant building block on the Function Easel and set its value equal to 150, as shown in Figure 9.4.10.

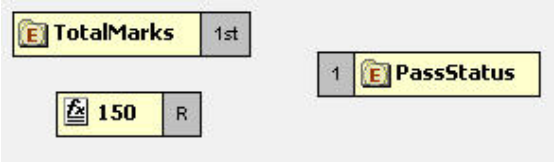


Figure 9.4.10: Setting the Constant building block to 150

6. Place a > node on the Function Easel.
7. Link the outputs of TotalMarks node and Constant node to the input of the > node.

   Also, link the output of the > node to the input of the PassStatus node, as shown in Figure 9.4.11.

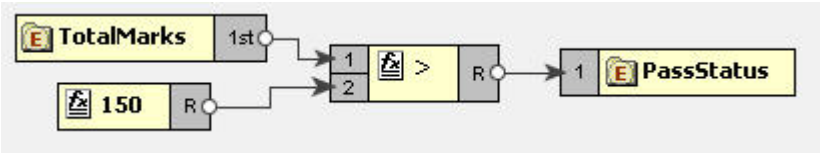


Figure 9.4.11: Linking the > and PassStatus node

8. This completes the desired mappings.

Greater than or Equal function

Function: >=

Input: >= (Number >= Number)

Description: True if the first input is greater than or equal to the second input.
Output: True/False

Example:

Consider example of TotalMarks.dtd as Input Structure and Result.dtd as Output Structure. Suppose we want that the value of the PassStatus node is true if the value of the TotalMarks node of the Input Structure is greater than or equal to a constant value 150.Then,

1. Load Input Structure and Output Structure.
2. Map the TotalMarks node of Input Structure to the PassStatus node of the Output Structure.
3. Invoke the Function Wizard by right-clicking on the PassStatus node.
4. The Function Easel opens with the existing mappings.

   Now place a Constant building block on the Function Easel and set its value equal to 150, as shown in Figure 9.4.12.

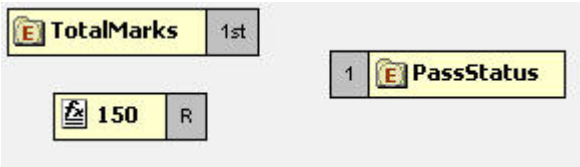


Figure 9.4.12: Setting the Constant building block to 150

5. Place a >= node on the Function Easel.
6. Link the outputs of TotalMarks node and Constant node to the input of the >= node.

   Also, link the output of the >= node to the input of the PassStatus node, as shown in Figure 9.4.13.

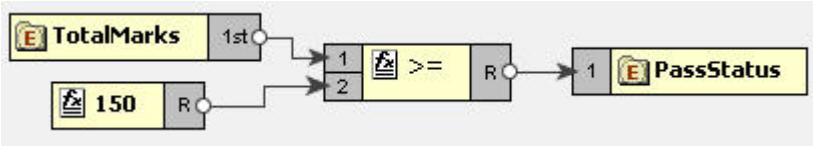


Figure 9.4.13: Linking the >= and PassStatus **nodes**

7. This completes the desired mappings.

OR

Symbol: OR

Description: This function accepts two boolean expressions as arguments and performs logical disjunction on them. If either expression evaluates to TRUE, the function returns TRUE. If neither expression evaluates to True, the function returns FALSE.

Input: OR (boolean OR boolean)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of Chat.dtd as Input Structure and Valid.dtdas Output Structure. Suppose we want to receive mails that are sent either from the address admin@nobody.com or aryton@nobody.com that is, we want that the isValid node of the Output Structure takes the value true if the Email node of the Input Structure has the value **admin@nobody.com** or **aryton@nobody.com**. Then,

1. Load Input Structure and Output Structure.
2. Map the Email node of Input Structure to the isValid node of the Output Structure.
3. Invoke the Function Wizard by right clicking on the isValid node.
4. The Function Easel opens with the existing mappings.
5. Place a Constant node on the Function Easel and set its value equal to **admin@nobody.com**.

   Place another Constant node and set its value equal to **aryton@nobody.com**, as shown in Figure 9.4.14.

Figure 9.4.14: Setting the Constant node value to aryton@nobody.com

Now place two = nodes on the Function Easel, and make links as shown in Figure 9.4.15.

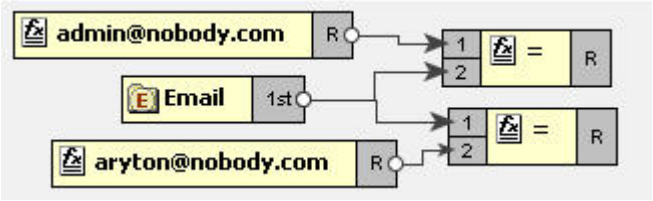


Figure 9.4.15: Placing two = nodes on the Function Easel

6. Place a OR node on the Function Easel.
7. Link the outputs of the two = nodes to the inputs of the OR node.

   Also, link the output of the OR node to the input of the isValid node, as shown in Figure 9.4.16.

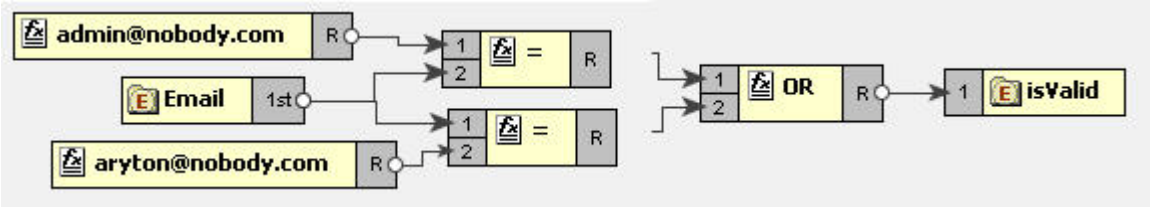


Figure 9.4.16: Linking the OR and isValid nodes

8. This completes the desired mappings.

Less Than or Equal

Symbol: <=

Description: This function returns TRUE if the first input is less than or equal to the second input.

Input: <= (Number <= Number)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of Numbers.dtd as Input Structure and Results.dtd as Output Structure. Suppose we want that Result node of Output Structure should have the value true if the value of Number1 node is less than or equal to the value of the Number2 node of the Input Structure. Then,

1. Load Input Structure and Output Structure.
2. Map the Number1 and Number2 nodes of Input Structure to the Result node of the Output Structure.
3. Invoke the Function Wizard by right-clicking on the Result node.
4. The Function Easel shows the existing mappings.

   Place the <= node on the Function Easel, as shown in Figure 9.4.17:

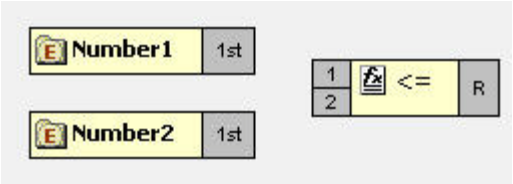


Figure 9.4.17: Placing <= node on the Function Easel

5. Link the outputs of the Number1 node and Number2 node to the inputs of the <= node.

   Also, link the output of the <= node to the input of the Result node, as shown in Figure 9.4.18:

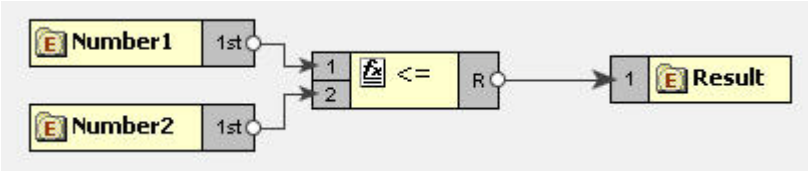


Figure 9.4.18: Linking outputs of the Number1 **and** Number2 **nodes**

6. This completes the desired mappings.

NOT

Symbol: NOT

Description: This function accepts a boolean expression as the argument and performs logical negation the expression. The result is a boolean value representing whether the expression is FALSE. That is, if the expression is FALSE, the result of this function is TRUE.

Input: NOT (boolean)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of Valid.dtd as Input and Output Structure. Suppose we want to make mails from email address admin@nobody.com as invalid. That is, we want that if the value of isFromAdmin node is true, then the value of isValid is set to false. Then,

1. Load Input Structure and Output Structure.
2. Map the isFromAdmin node of Input Structure to the isValid node of the Output Structure.
3. Invoke the Function Wizard by right-clicking on the isValid node.
4. The Function Easel shows the existing mappings.

   Now place a NOT node on the Function Easel, as shown in Figure 9.4.19.

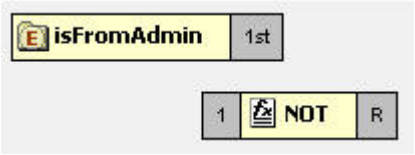


Figure 9.4.19: placing a NOT **node on the Function Easel**

5. Link the output of the isFromAdmin node to the input of the NOT node.

   Also link the output of the NOT node to the input of the isValid node, as shown in Figure 9.4.20.

Figure 9.4.20: Linking theNOT and isValid nodes

6. This completes the desired mappings.

Not Equal

Symbol ! =

Description: This function returns TRUE if both the inputs are not equal.

Input != (Object = Object)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of Chat.dtd as Input Structure and Valid.dtd as Output Structure. Suppose we want to filter out mails that do not have message body. That is, we want that the isMessageExist node of the Output Structure takes the value true if the length of the Message node of the Input Structure is not equal to zero. Then,

1. Load Input Structure and Output Structure.
2. Map the Message node of Input Structure to the isMessageExist node of the Output Structure.
3. Invoke the Function Wizard by right clicking on the isMessageExist node.
4. The Function Easel opens with the existing mappings.
5. Now place a **Constant** building block on the Function Easel and set its value equal to 0.
6. Place a **Length** node on the Function Easel.

   Link the output of the Message node to the input of the Length node, as shown in Figure 9.4.21.

Figure 9.4.21: Linking the Message and Length nodes

7. Place a != node on the Function Easel.
8. Link the outputs of the **Length** node and Constant node to the inputs of the != node.

   Also, link the output of the != node to the input of the isMessageExist node, as shown in Figure 9.4.22.

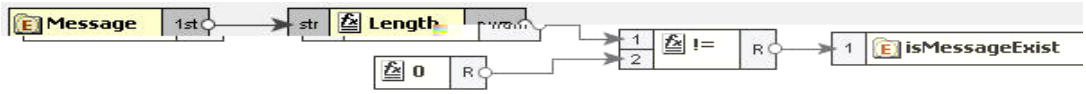


   Figure 9.4.22: Linking the !=and isMessageExist nodes

9. This completes the desired mappings.

IsNumber

Symbol: IsNumber

Description: This function returns TRUE if the input value is a number.

Input: Any value

Output: Boolean value (TRUE/FALSE)

9.4.1.11 Lookup functions

The functions in this category are used to perform the lookup of keyvalue pairs in a database and return the result in sorted fashion.

9.4.1.11.1 Lookup with Default Connection Details

DB

Description: This function accepts a table name, keyvalue pairs and column names as arguments and does the lookup in the database and returns the result in sorted form.

Input: Table name, Key value pairs, Columns names.

Output: String containing the lookup result in sorted order.

Points to note

1. DBLookup fetches the entries from a table and stores them in local cache the first time and it uses this cache to perform lookups for subsequent requests. This cache will not be updated at runtime again. Therefore it should preferably be used only for tables which are not updated once populated, that is for static data.
2. Lookup functions take key columns name value pairs as <column1>=<value1>,<column2>=<value2> etc.

   For example: dvSendDept=100, dvSendCode=BLK

3. Lookup functions can return value of multiple columns. To get multiple columns, use the format <column3>,<column4>.
   For example: dvValueDA, dvDescription
4. Dates are expected in MM/dd/yyyy HH:mm:ss format
5. Make sure the input value match the column length defined in the database. For example, if the dvSendCode is defined as char(10) in the database, the input value should be BLK followed by seven spaces.

   Note: Spaces are not required if you are using MSSQL 2005.

Prerequisites

1. Add required database drivers in the Mapper classpath.

   For example, if the lookup tables are in HSQL, include the path of hsqldb.jar in <java.classpath> of mapper.conf present at {FIORANOHOME}/esb/tools/mapper/bin.
2. To use this function in Mapper tool, a system property mapper.lookup.dbconfig has to be defined in mapper.conf and it should point to the path of db.properties file which contains the url, driverName, user and password.
   Sample db properties file is shown below which contains the data for oracle data base.

```
#DB PORPS
#Sat Jun 03 19:26:53 IST 2006
url=jdbc:oracle:thin:@192.168.2.92:1521:fiorano
driverName=oracle.jdbc.driver.OracleDriver
user=scott
password=tiger
```

   Figure 9.4.23: Sample db properties file

3. For use in Route transformations, **mapper.lookup.dbconfig** property has to be set in {FIORANOHOME}/fps/bin/fps.conf.
4. For use in XSLT component, **mapper.lookup.dbconfig** property has to be included in JVM_PARAMS
   For example: -Dmapper.lookup.dbconfig=<path of db.properties>

9.4.1.11.2 Lookup with Connection Details

DB

Description: This function accepts a table name, keyvalue pairs, column names, url, driver name, user name and password as arguments and does the lookup in the database and returns the result in sorted form.

Input: Table name, Key value pairs, Columns names, url, driver name, user name and password.

Output: String containing the lookup result in sorted order.

Points to note

1. DBLookup fetches the entries from a table and stores them in local cache the first time and it uses this cache to perform lookups for subsequent requests. This cache will not be updated at runtime again. Therefore it should preferably be used only for tables which are not updated once populated, that is for static data
2. Lookup functions take key columns name value pairs as <column1>=<value1>,<column2>=<value2> etc.

   For example, dvSendDept=100, dvSendCode=BLK
3. Lookup functions can return value of multiple columns. To get multiple columns, use the format <column3>,<column4>.

   For example, dvValueDA, dvDescription
4. Dates are expected in MM/dd/yyyy HH:mm:ss format
5. Make sure the input value match the column length defined in the database. For example, if the dvSendCode is defined as char(10) in the database, the input value should be BLK followed by 7 spaces.

Prerequisites

1. Add required database drivers in the Mapper classpath.

   For example, if the lookup tables are in HSQL, include the path of hsqldb.jar in <java.classpath> of mapper.conf present at {FIORANOHOME}/esb/tools/mapper/bin.

9.4.1.12 JMS Message Functions

The various functions in this category extract specific information from a JMS Message and output to the same. The input for these functions is a JMS Message. The following are the available JMS Message Functions:

- Byte Content
- Text Content
- Header
- Attachment

9.4.1.12.1 Byte Content

Function: Byte Content

Description: The Byte Content function returns the byte content of a Fiorano document.

Output: Base64 encoded string value

9.4.1.12.2 Text Content

Function: Text Content

Description: The Text Content function returns the content which is in text format from a Fiorano document.

Output: String value

9.4.1.12.3 Header

Function: Header

Description: The Header function returns the value of the name that is passed as a property to the function.

Output: String value

9.4.1.12.4 Attachment

Function: Attachment

Description: The Attachment function returns any attachments attached to a Fiorano document. The name of the attachment needs to be passed as a property to the function.

Output: Base64 encoded string value

9.4.1.13 User Defined functions

The various functions in this category are user defined and perform various functionalities. The following User Defined functions are available:

- dateConversion
- compute
- nextMillenium
- replace

9.4.1.13.1 myExt:dateConversion

Description: Converts the date from one format to the other. For example, date can be converted from MM-dd-yyyy to dd-MM–yy function convertDate (dateString, inFormat, outFormat)

Field	Full Form	Short Form
Year	yyyy (4 digits)	yy (2 digits), y (2 or 4 digits)
Month	MMM (name or abbr.)	MM (2 digits), M (1 or 2 digits)
	NNN (abbr.)	
Day of Month	dd (2 digits)	d (1 or 2 digits)
Day of Week	EE (name)	E (abbr)
Hour (1-12)	hh (2 digits)	h (1 or 2 digits)
Hour (0-23)	HH (2 digits)	H (1 or 2 digits)

Hour (0-11)	KK (2 digits)	K (1 or 2 digits)
Hour (1-24)	kk (2 digits)	k (1 or 2 digits)
Minute	mm (2 digits)	m (1 or 2 digits)
Second	ss (2 digits)	s (1 or 2 digits)
AM/PM	a	

Input: Accepts three arguments. The first argument is the date passed as a string to the **function.** The second argument is the input format and the third argument is the required output format for the date.

Output: The date string

Examples:

MMM d, y matches: January 01, 2000, Dec 1, 1900, Nov 20, 00

M/d/yy matches: 01/20/00, 9/2/00

MMM dd, yyyy hh:mm:ssa matches: January 01, 2000 12:30:45AM

9.4.1.13.2 myExt: replace

Description: This user-defined function replaces parts of a string that match a regular expression with another string.

string regexp:replace(string, string, string, string)

Input: The function accepts four arguments. The first argument is the string to be matched and replaced. The second argument is a regular expression that follows the Javascript regular expression syntax. The fourth argument is the string to replace the matched parts of the string.

The third argument is a string consisting of character flags to be used by the match. If a character is present then that flag is true. The flags are:

g: global replace - all occurrences of the regular expression in the string are replaced. If this character is not present, then only the first occurrence of the regular expression is replaced.
i: case insensitive - the regular expression is treated as case insensitive. If this character is not present, then the regular expression is case sensitive.

Output: String

9.4.1.13.3 myExt:compute

Description: This user-defined function can be used to compute all mathematical operations such as Addition, Subtraction, Multiplication and division of number. The function does not compute mathematical operations such as cos, sin etc.

Input: A valid javascript expression

Output: A number

9.4.1.13.4 myExt: nextMillenium

Description: This user-defined function returns the number of days in the next millenium.

Input: There is no input for this function

Output: Number

9.4.2 Funclet Easel

This panel is the basic work area for creating expression based mappings. The user can place the Function nodes as well as the Source or Destination nodes on this area and make the required mappings.

The Funclet easel appears as shown in Figure 9.4.24.

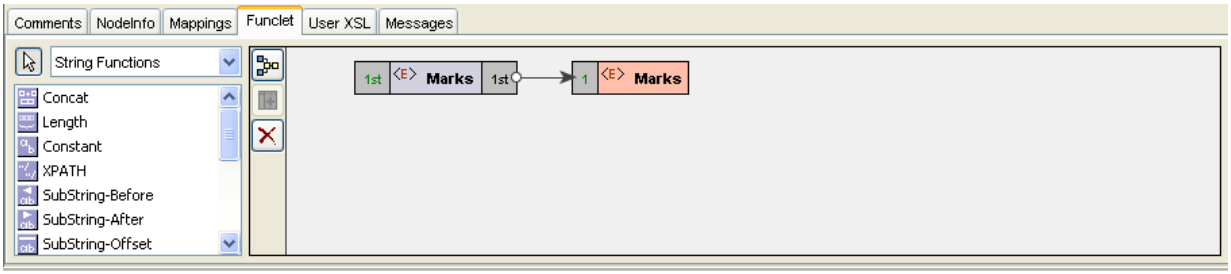


Figure 9.4.24: Funclet easel

9.4.2.1 Source Node

The Source node corresponds to a node in the Input Structure Panel. A Source node appears, as shown in Figure 9.4.25.

Figure 9.4.25: Source Node

9.4.2.2 Destination Node

The Destination node corresponds to a node in the Output Structure Panel. A Destination node appears, as shown in Figure 9.4.26.

Figure 9.4.26: Destination Node

Add Link between two Nodes

To make a link between two nodes placed on the Funclet easel, follow the steps below:

1. Click on the gray box on the source building block. A small circle appears, as shown in Figure 9.4.27. This represents the starting point of the link and the output box of the building block.

Figure 9.4.27: Source node

2. Now drag-and-drop the mouse to the Destination node's input point, which is again represented by a gray box. A big circle appears on the destination node, as shown in Figure 9.4.28.

Figure 9.4.28: Linking the Source and the Destination node

3. Release the mouse. A link between the two nodes is created.

Add Source node to Funclet easel

Drag-and-drop the source node from the **Input Structure Panel** to the Funclet easel, as shown in Figure 9.4.29.

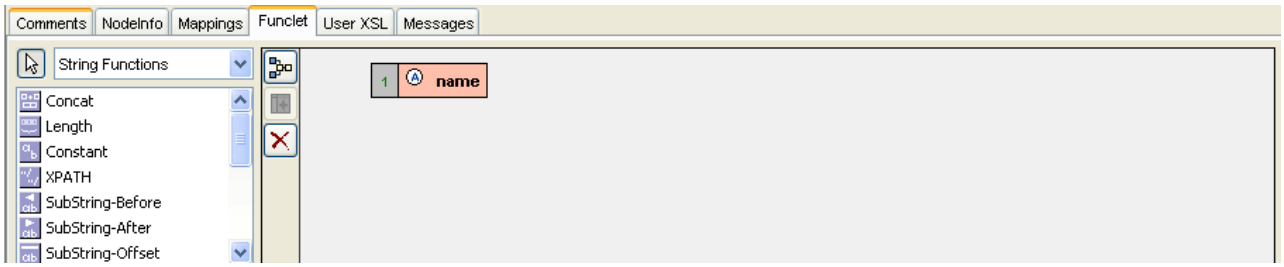


Figure 9.4.29: Adding Source node to Funclet easel

Add Function node to Funclet easel

1. Click the Function node on the Function palette that is to be placed on the Funclet easel, as shown in Figure 9.4.30.

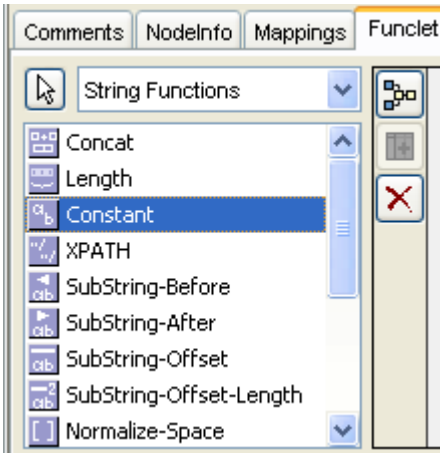


Figure 9.4.30: Selecting the Function node

2. Now move mouse into the Funclet easel. This changes the mouse to a?+? '+' sign, representing that the corresponding function node is selected.
3. Now click on the Funclet easel.
4. This places the corresponding function node building block on the Funclet easel.

Alternatively,

1. Drag-and-Drop the function node from Function palette to the Funclet easel, as shown in Figure 9.4.31.

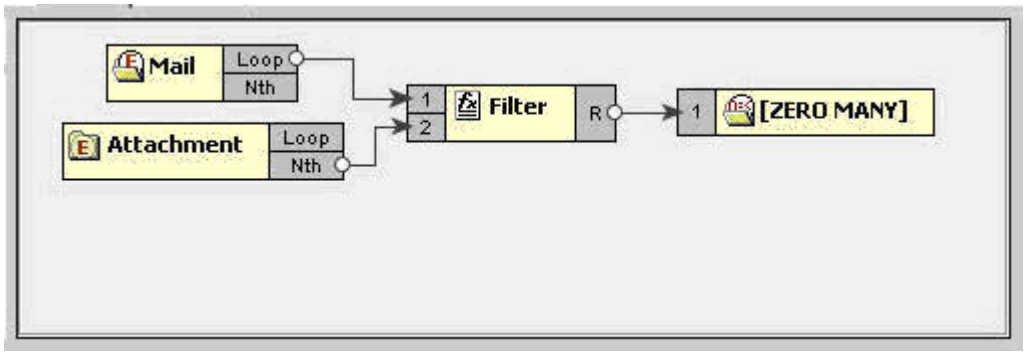


Figure 9.4.31: Funclet easel

Add Link between two nodes

To make a link between two nodes placed on the Funclet easel, follow the steps below:

1. Click on the gray box on the source building block. A small circle appears, as shown in Figure 9.4.32. This represents the starting point of the link and the output box of the building block.

Figure 9.4.32: Source node

2. Now drag-and-drop the mouse to the destination node's input point, which is again represented by a gray box. A big circle appears on the destination node, as shown in Figure 9.4.33.

Figure 9.4.33: Linking the Source and the destination node

3. Release the mouse. A link between the two nodes is created, as shown in Figure 9.4.34.

Figure 9.4.34: Linking Source and Destination nodes

Delete link between two nodes

To delete a link between two building blocks,

1. Click on the pointing arrow (ending point) of the link and drag it to an empty area in the Funclet easel, as shown in Figure 9.4.35.

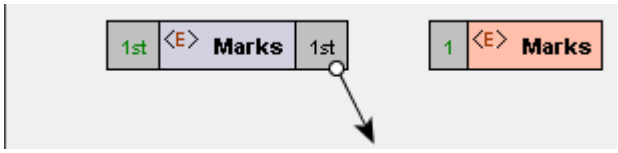


Figure 9.4.35: Deleting link

2. Now, release the mouse. This removes the link between the corresponding nodes.

Delete node from Funclet easel

1. Select the corresponding building block and right click on it. The shortcut menu appears as shown in Figure 9.4.36.

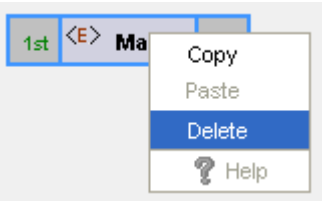


Figure 9.4.36: Pop-up menu

2. Click **Delete** to delete the selected building block.

Auto layout Funclet easel

1. Click on the Autolayout icon in the Funclet easel as shown in Figure 9.4.36.

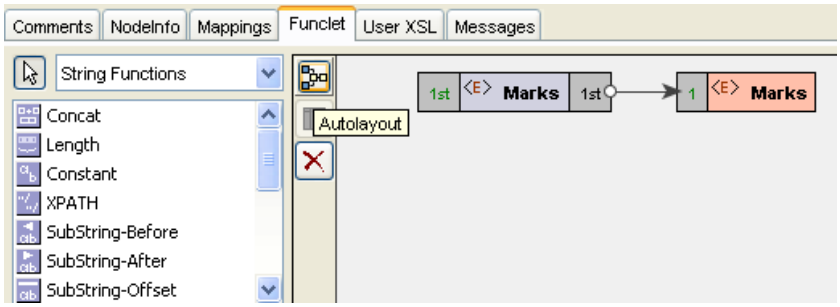


Figure 9.4.37: Shortcut menu

The Funclet easel would properly be laid out, as shown in Figure 9.4.38.

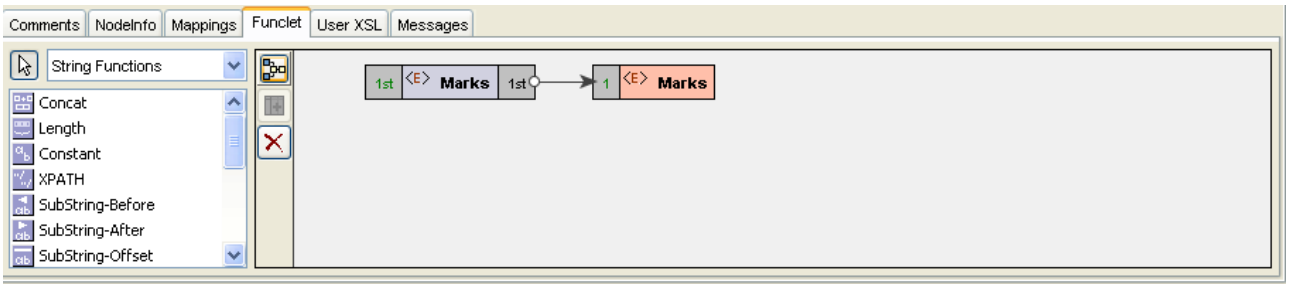


Figure 9.4.38: Funclet easel layout

9.5 Creating Mappings

Mappings are defined between nodes of the Input and Output structures. The Structure is displayed in a tree form.

9.5.1 Understanding Types of Nodes

Mappings are defined between nodes of the Input and Output structures. These nodes can be divided into four types:

1. **Element Node:** This type of node contains an XML element.
2. **Text Node:** This type of node contains an XML element only.
3. **Attribute Node:** This type of node contains an attribute of the XML element that contains it.
4. **Control Node:** The control node is a pseudo node that depicts the cardinality of the elements in an XML structure. The Control node is displayed in red color, and is surrounded by square brackets.

The control node serves as a useful indicator while creating mappings between the Input and Output Structures. For example, an Output structure node that has a cardinality of one or more requires that at least one element should be added to that XML structure.

5. **Control Node[ZERO-MANY] :** This Control node specifies that zero to many occurrences of a node can exist in its parent node. For example, in Figure 9.5.1 the Mail-List element can contain zero or many Mail nodes.

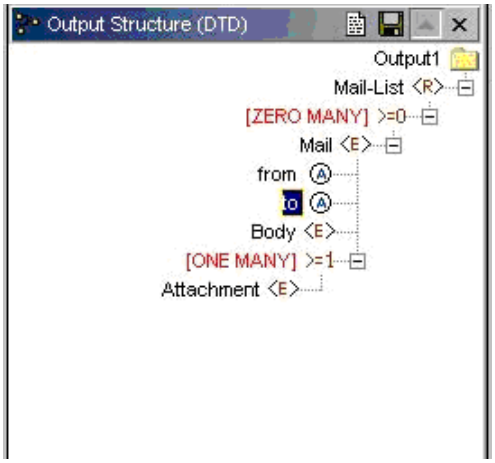


Figure 9.5.1: Example of Zero to Many control node

6. **Control Node [ONE-MANY]:** This Control node specifies that one to many occurrences of a node can exist in its parent node. For example, in Figure 9.5.2 the Mail node can contain one or many occurrences of the Attachment node.

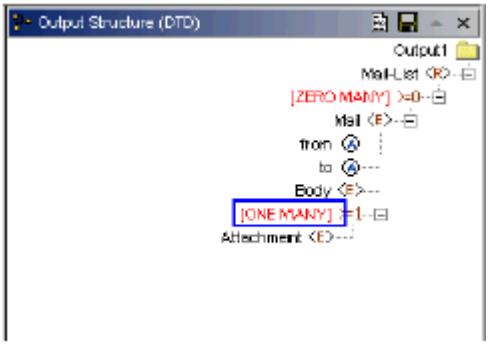


Figure 9.5.2: Example of One to Many control node

7. **Control Node [OPTIONAL]:** This Control node specifies that zero to one occurrences of a node that can exist in its parent node. This Control node specifies that either zero or one occurrence of a node can exist in its parent node. For example, in Figure 9.5.3 Student node can have zero or one occurrence of the Nick-Name node.

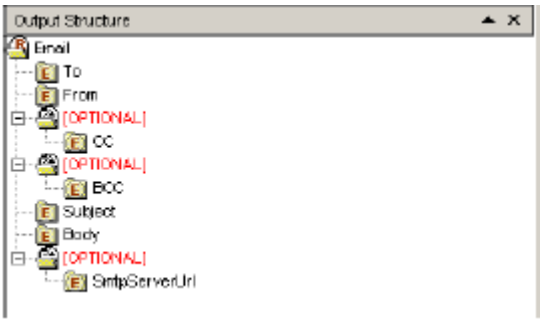


Figure 9.5.3: Example of Optional control node

8. **Control Node [OR]:** This Control node specifies that only one of the descendant nodes can exist in the parent node. For example in Figure 9.5.4 TifosiService node can have either Java node or Win32 node, but not both.

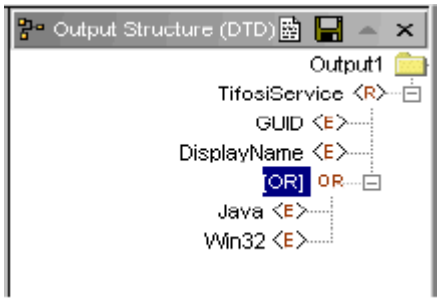


Figure 9.5.4: Example of OR Control Node

9. **Control Node [SEQUENCE]:** This Control node specifies that all the descendant nodes should exist in the specified sequence in the parent node. For example, in Figure 9.5.5, TifosiService element should have either Java element or Win32 OS elements.

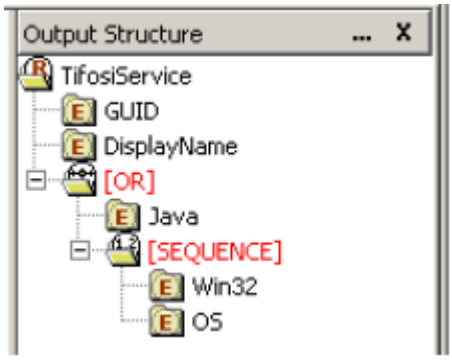


Figure 9.5.5: Example of SEQUENCE control node

Limitation

In Fiorano Mapper, the orders of attributes of an element are loaded alphabetically in descending order. As a result, when you open Fiorano Mapper, the order of attributes might change, thus leading to generation of incorrect mapping. In such case, Mapper flashes a message which is as follows:

Note: Order of attributes has been changed for element: Response

9.5.2 Types of Mappings

Mappings from an Input Structure node to an Output Structure node can be singular or iterative. Singular mappings, known as Name-to-Name mappings in Fiorano SOA Platform, create only one output element from the first instance of the mapped element in the Input Structure.

On the other hand, iterative mappings, known as For-Each mappings in Fiorano SOA PLATFORM, iterate through all instances of the mapped Input Structure element and create corresponding Output Structure elements.

For Input Structure nodes that contain only single instances of child elements, only Name-to-Name mappings can be defined.

9.5.2.1 Name-to-Name Mapping

Now create mapping from Name-to-Name, as shown in Figure 9.5.6.

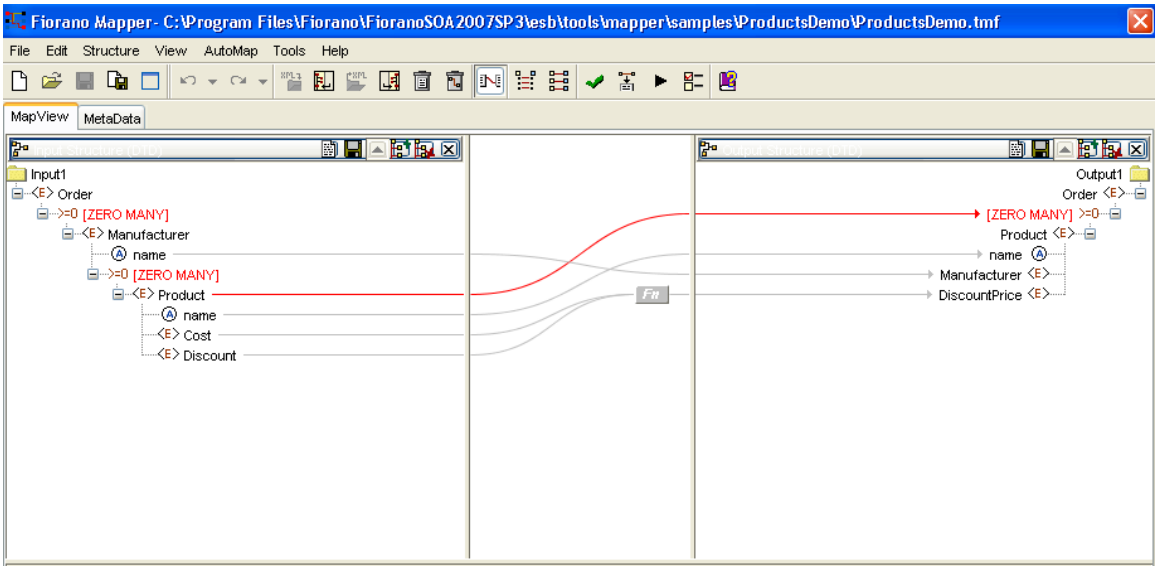


Figure 9.5.6: Name-to-name Mapping

The Funclet Wizard shows a link starting from Nth output of the name input node to name output node. The Name-to-Name mapping defines how elements and attributes in the Input Structure map on to elements and attributes in the Output Structure. A Name-to-Name mapping on its own (without a For-Each mapping context) creates a single instance of the mapped Input Structure node to the Output Structure.

If the Name-to-Name mapping exists within a For-Each mapping context and there are multiple elements and attributes in the Input Structure then each of those elements and attributes is mapped on to an Output Structure node.

9.5.2.2 For-Each Mapping

When an Input Structure node can have multiple instances and you want to define a mapping for each one of them, then For-Each mapping should be used. A necessary condition for this type of mapping is that the Output Structure node to which For Each Mapping is being defined should be of[ZERO-MANY] or [ONE-MANY] cardinality. Figure 9.5.7, shows an instance of a For-Each mapping.

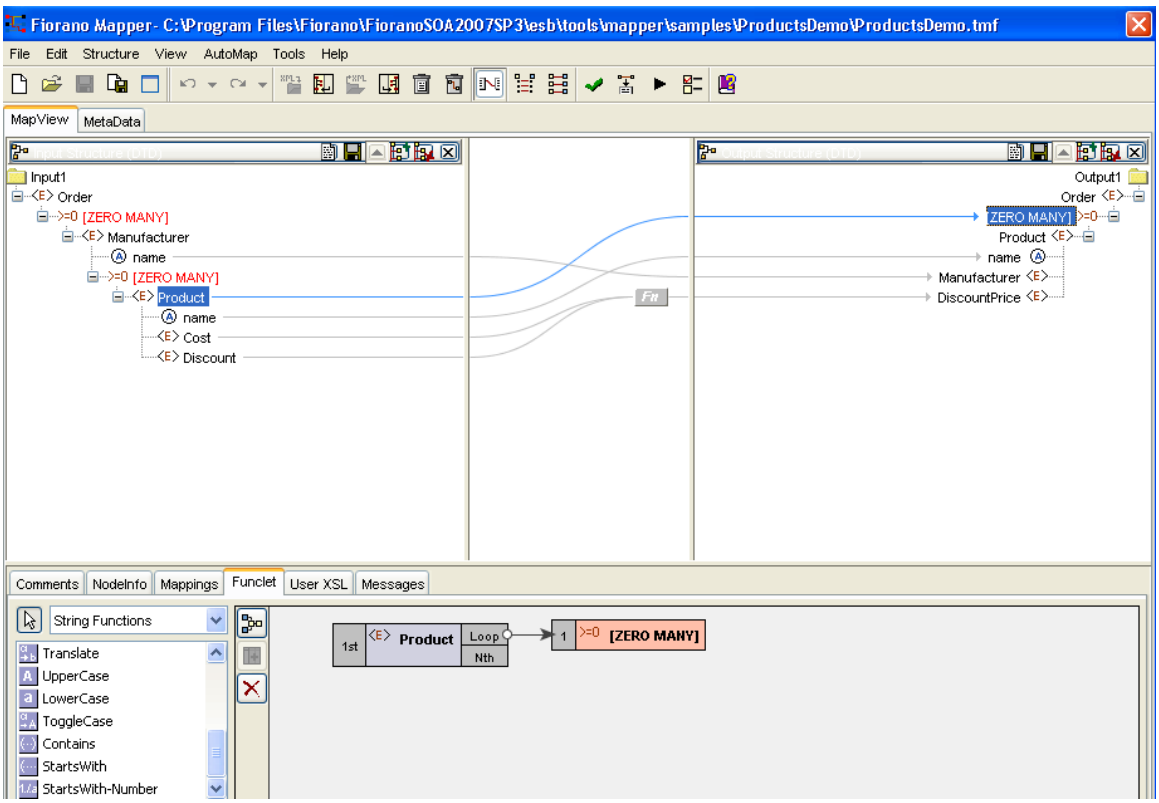


Figure 9.5.7: For-Each Mapping

This mapping specifies that for each Product element in the input XML, the output XML contains a Product element. For-each mapping can be applied only to [ZERO-MANY] or [ONE-MANY] control nodes in the Output Structure.

To create a For-each mapping in the Funclet Wizard, you need to link the Loop output label of the Input Structure node to a [ZERO MANY] or [ONE-MANY] control node in the Output Structure. These control nodes signify the cardinality of contained elements and attributes.

All value mappings for the attributes and child elements of a [ZERO MANY] or [ONE-MANY] node with For-Each mapping, are carried out within this For-Each context.

So, in Figure 9.5.7 the mapping defined creates multiple instances of the Product element from the Product elements in the Input Structure. The Output element, Product, is created as per the mappings defined for its attributes and child elements by the respective Name-to-Name mappings.

9.5.3 Duplicating a For-Each Mapping

There may be situations in which you may want to specify different input values for different iterations of a For-Each loop. This can be accomplished by duplicating a [Zero Many] or [One Many] control node in the output structure.

The following example illustrates this situation. A Student DTD has two types of child elements: male and female. These need to be mapped to student element in the output structure DTD, as shown in Figure 9.5.8.

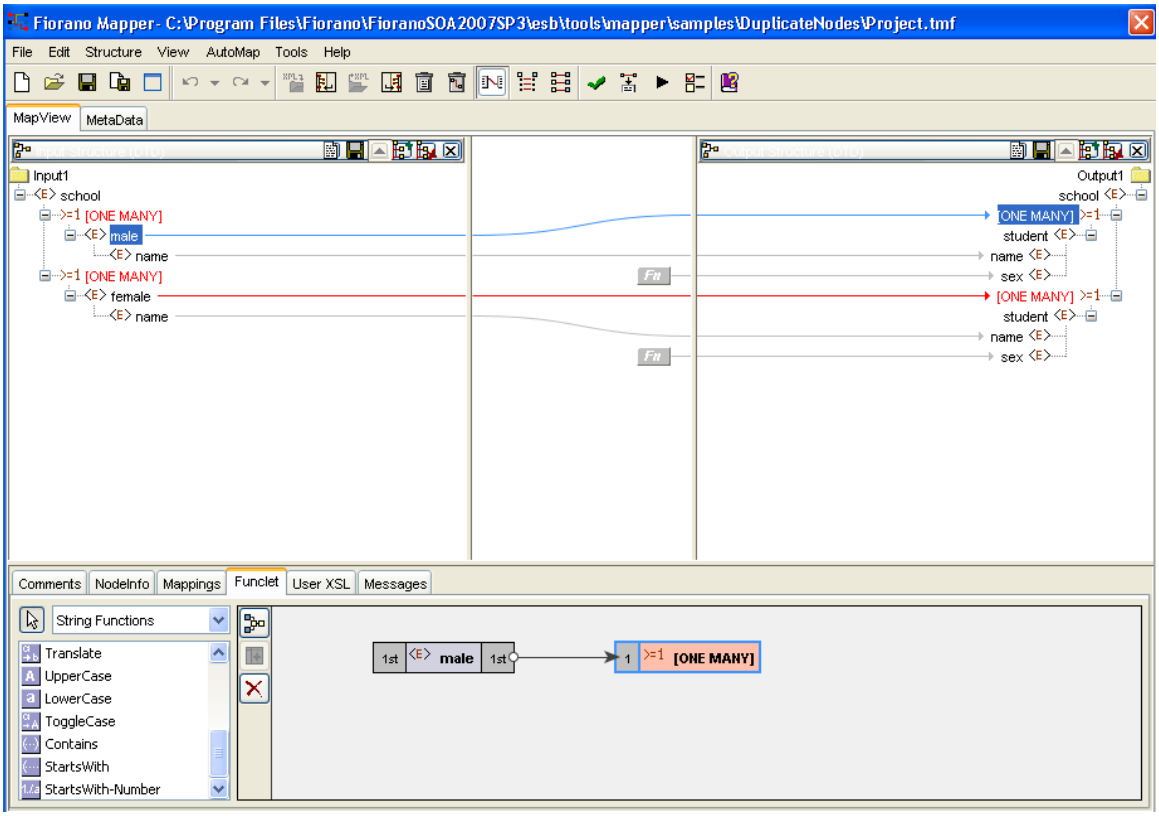


Figure 9.5.8: Mapping a node to One Many control node

The same mapping has to be defined for the female elements. To do this, drag the female node from the input structure to the output structure. A shortcut menu is displayed as shown in Figure 9.5.9.

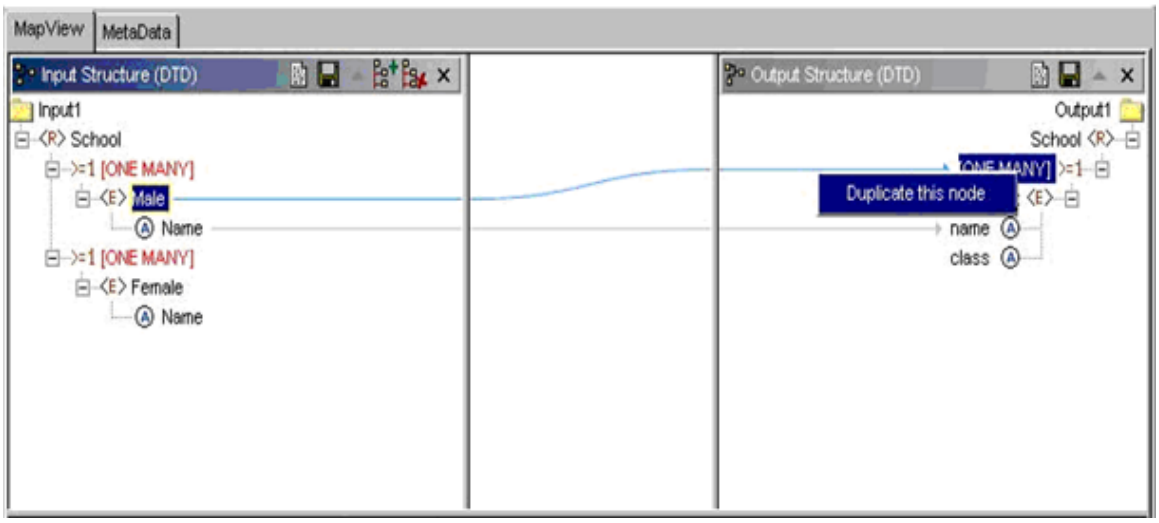


Figure 9.5.9: A shortcut menu prompts you to duplicate the node

Select the **Duplicate this node** option in the shortcut menu. A mapping is created as shown in Figure 9.5.10.

Figure 9.5.10: The One Many Node is Duplicated

9.5.4 Linking Nodes to Define Mappings

A Mapping is defined in the Fiorano Mapper tool by visually linking the Input Structure nodes to the Output Structure nodes. This linking can be defined using any of the following techniques:

1. Drag and drop the node from the Input Structure Panel to the Output Structure Panel
2. Or, create an automatic mapping between child nodes of the selected Input Structure node and child nodes of the selected Output Structure node
3. Or, by using the Visual Expression Builder

9.5.4.1 Using the Automatic Mapping option to Define Mappings

To create automatic mappings between the selected Input and Output Structure nodes:

Select the nodes in the Input and Output Structure whose child nodes are mapped. Click the **AutoMap** > **Child to Child** option in the menu bar, as shown in Figure 9.5.11.

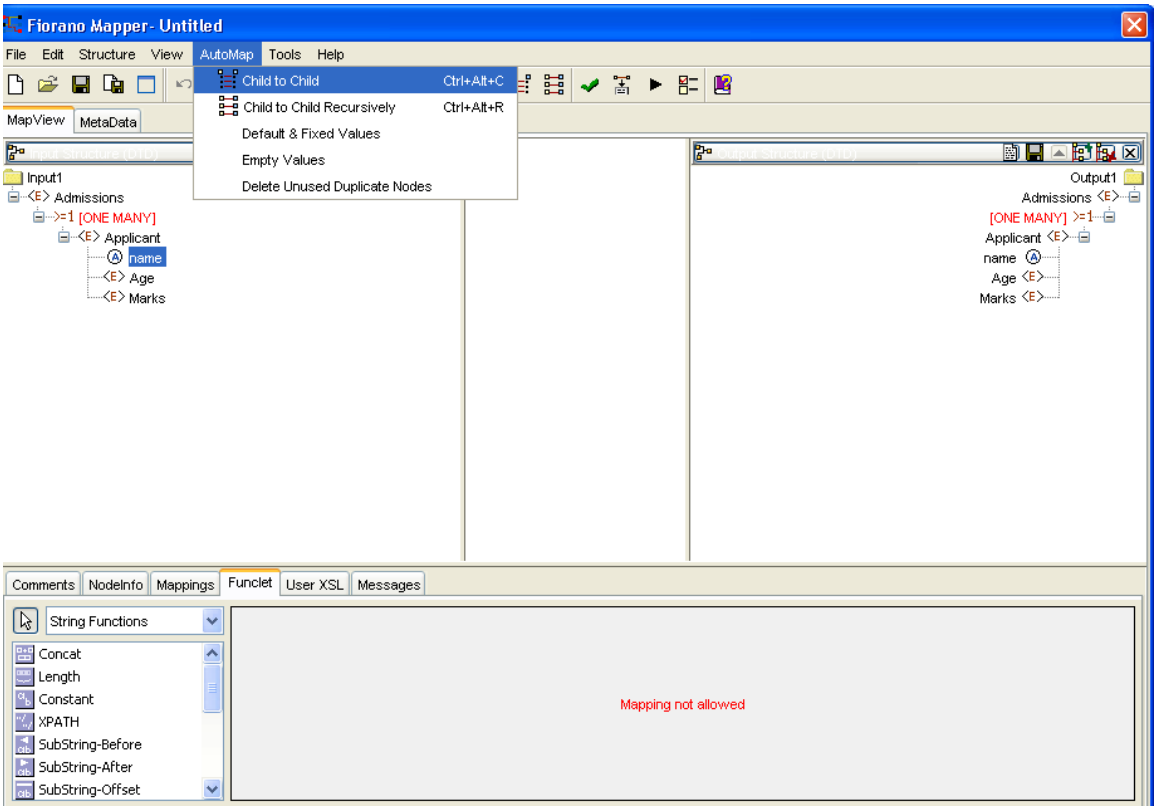


Figure 9.5.11: Creating Automatic Mapping between child nodes

9.5.4.2 Using the Visual Expression Builder to Define Mappings

The Visual Expression Builder (VEB) is a useful feature of the Mapper tool. It allows you to visually link nodes and insert functions to define complex mapping expressions. As an example, we define a mapping for the DiscountPrice output node. This node should have a value that is generated by subtracting the value of the Discount input node from the Cost input node. To use the VEB to define the mapping perform the following steps:

1. Click the **Funclet** tab in the bottom panel of the MapView. Select the DiscountPrice output node, as shown in Figure 9.5.12.

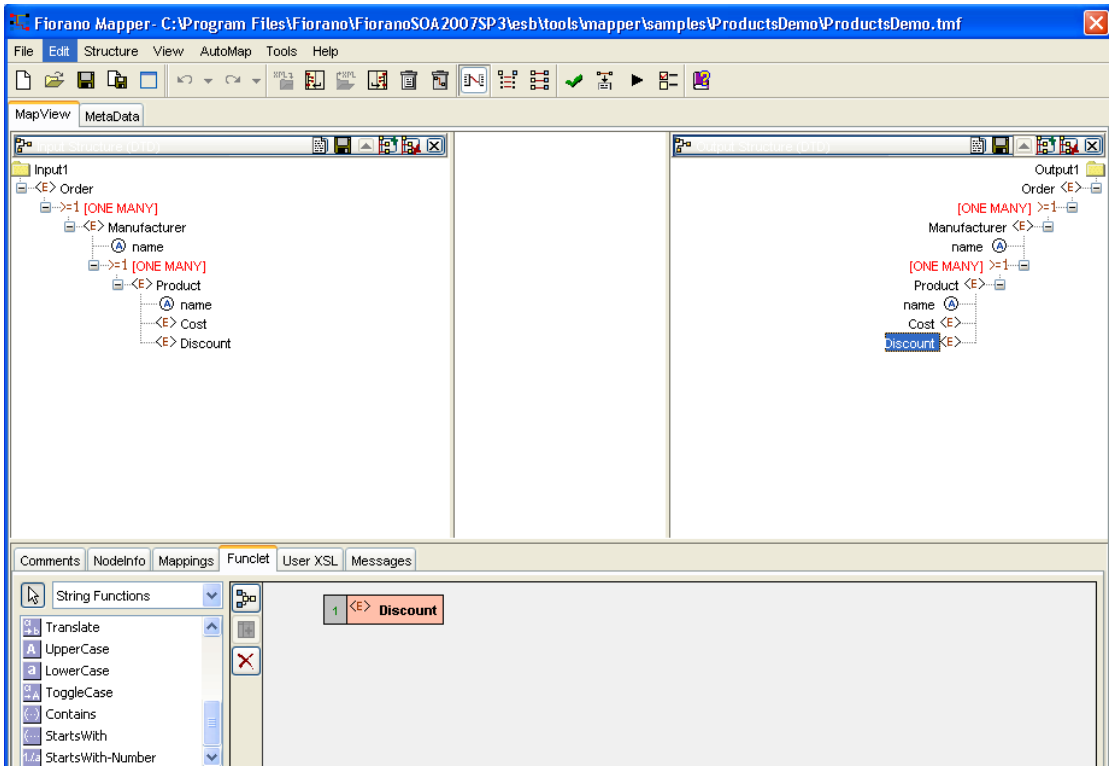


Figure 9.5.12: Selecting the Output Node for Mapping

2. The selected Output node is automatically displayed in the Function easel, as shown in the Figure 9.5.12. To add the input structure nodes to the mapping you need to drag them to the Funclet easel of the Visual Expression Builder. First, drag the Cost input node from the Input Structure Panel to the Funclet easel. The Cost input node is added to the Funclet easel as shown in Figure 9.5.13.

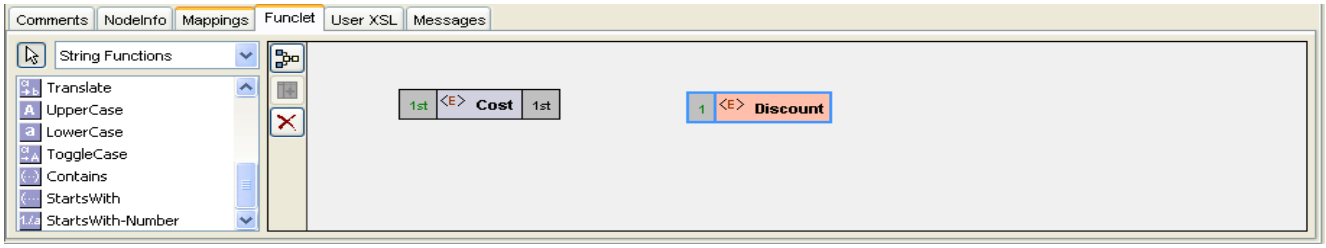


Figure 9.5.13: Dragging an Input node

3. Now, you need to subtract the value of Discount input node. For this, you need to add the subtract function to the Funclet easel. The subtract function is available in the Arithmetic functions. To add the subtract function; first select the Arithmetic function category from the Function palette. Click on the drop-down list in the Funclet palette. The drop-down list is displayed in the Funclet palette, as shown in Figure 9.5.14.

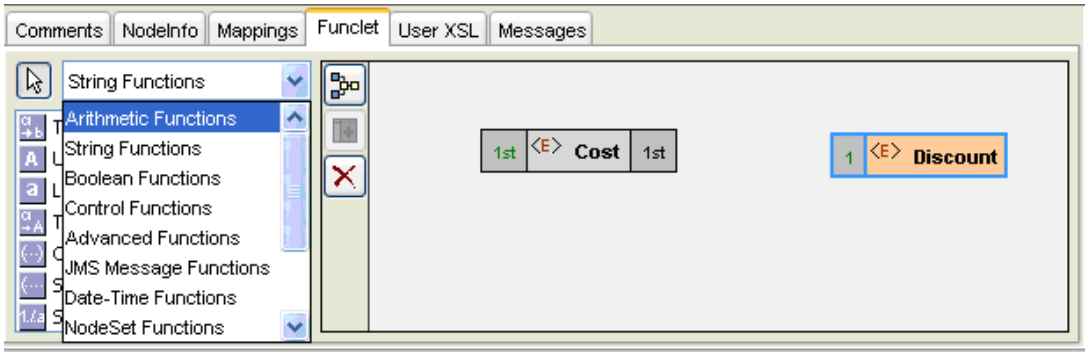


Figure 9.5.14: Selecting the Arithmetic Function Category in the Funclet palette

4. Select **Arithmetic Functions** from the list. The Arithmetic functions are displayed in the **Function palette**. Drag the subtract function from the Function palette to the Funclet easel. The subtract function is added to the Funclet easel as shown in Figure 9.5.15.

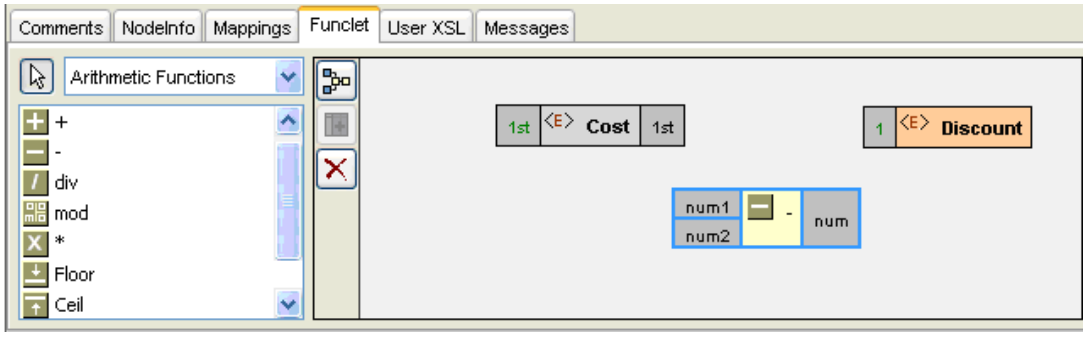


Figure 9.5.15: Adding the Subtract function

5. Next, add the Discount input node to the Funclet easel as you added the Cost node as shown in Figure 9.5.16.

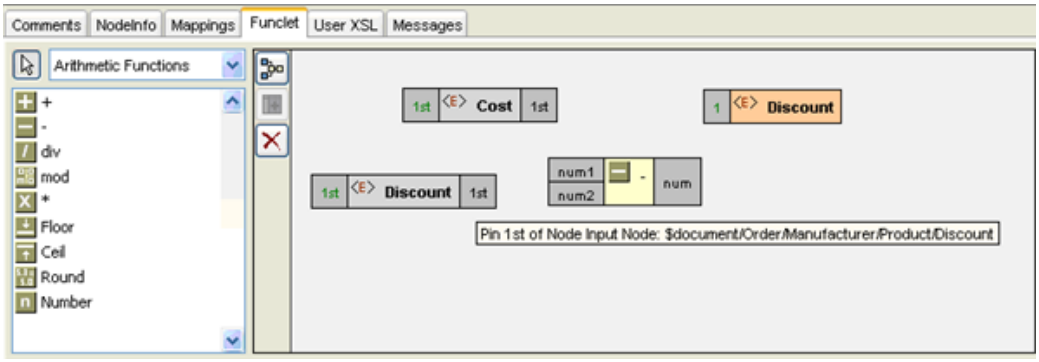


Figure 9.5.16: Adding another input node

6. Now, you need to link the input nodes to the subtract function. First link the Cost node to num1 Pin of the subtract function.

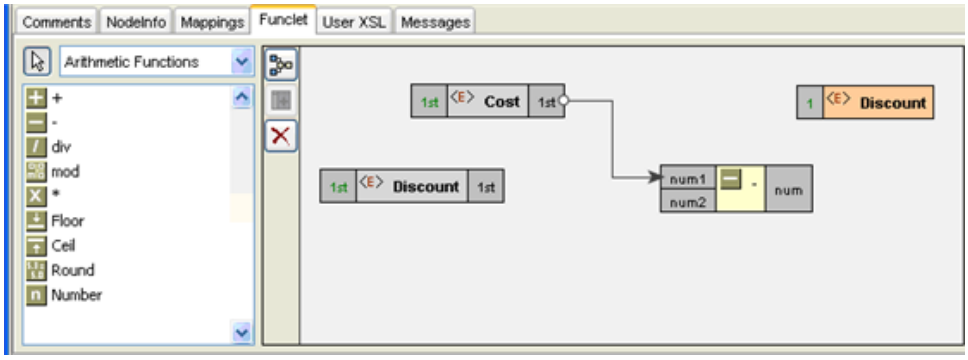


Figure 9.5.17: Linking the input nodes to the subtract function

7. Next, link the Discount node to num2 pin of the subtract function.

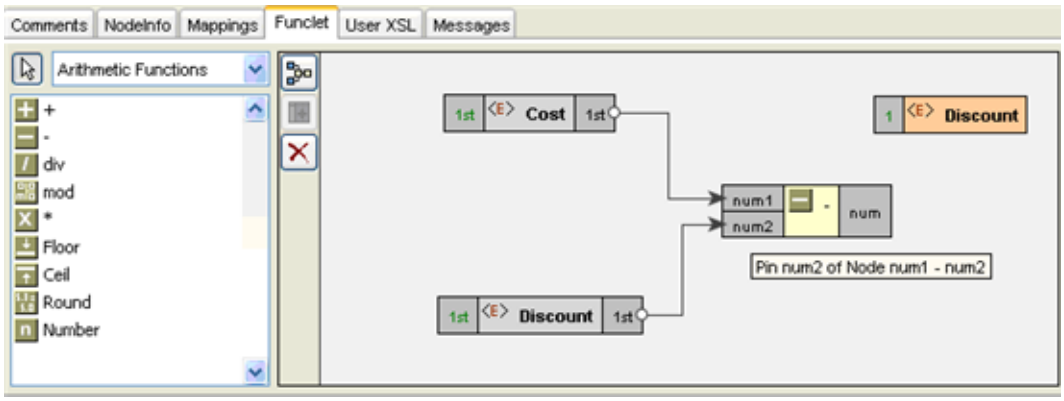


Figure 9.5.18: Linking the input nodes to the subtract function

8. Finally, link the subtract function to the DiscountPrice to create the mapping.

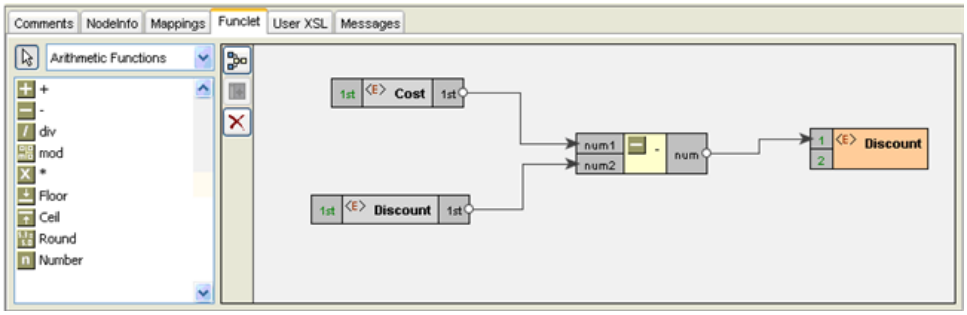


Figure 9.5.19: The final mapping is defined

9. The required mapping is defined as shown in Figure 9.5.19.

9.5.5 Mapping XML Formats

Mapping one XML format to another is a common requirement. The steps for mapping XML, formats to each other are as follows:

1. Load the XML, DTD, or XSD input structure or structures.
2. Load an XML, DTD, or XSD output structure.
3. Link the Input XML Structure node or nodes to the Output XML Structure node.

The following restrictions and conditions apply when mapping one XML format to another:

1. Nodes that do not have any content cannot be mapped. You can, however, map the child nodes of these nodes, if they can contain content.
2. The SQL and advanced function categories are not available for XML to XML mapping

9.5.6 Mapping XML Formats to CSV Files

The procedure for mapping XML formats to CSV formats is the same as that of mapping XML formats.

1. Load the XML, DTD, or XSD input structure or structures.
2. Load the CSV output structure.
3. Link the Input XML Structure node or nodes to the Output CSV Structure node.

The following restrictions and conditions apply when mapping an XML format to a CSV format:

1. Nodes that do not have any content cannot be mapped. You can, however, map the child nodes of these nodes, if they contain contents.
2. The SQL and advanced function categories are not available for XML to XML mapping.

9.5.7 Mapping XML Formats to RDBMS Queries

The procedure for defining mappings to RDBMS queries is a two-step process that involves configuring the RDBMS Output Structure, and then defining the mapping.

The steps for configuring the RDBMS Output structures have been defined earlier in this chapter.

9.5.7.1 Mapping XML Formats to RDBMS-Insert Queries

Mapping an XML Input Structure to an RDBMS-Insert output structure creates a SQL query that inserts the data received from the input XML structure to the specified RDBMS database.

1. **Load an RDBMS-Insert output structure:** If you invoke the Mapper tool from the STUDIO tool, the RDBMS-Insert output structure is automatically loaded. However, if you start it independently, you need to load the output structure.
2. **Specify the Set clause:** Define the mappings between the input XML structures to the Set clause of the RDBMS-Insert output structure.

All nodes in the RDBMS-Insert output structure that are displayed in black are database columns that cannot be null. Therefore, you need to specify a mapping for them that insert the appropriate values into them.

9.5.7.2 Mapping XML Formats to RDBMS-Update Queries

Mapping an XML input structure to an RDBMS-Update output structure is a simple two step procedure:

1. **Load an RDBMS-Update output structure:** If you invoke the Mapper tool from the STUDIO tool, the RDBMS-Update output structure is automatically loaded. However, if you start it independently, you first need to load the output structure.
2. **Define the mappings:** For an RDBMS-Update output structure, you need to define mappings for both the Set clause and the Where clause:
    a. **Set clause of RDBMS-Update:** All nodes that are displayed in black color are database columns that cannot be null. Therefore, you need to define a mapping and set an input value for them.
    b. **Where clause of RDBMS-Update:** You must specify a Where condition for the RDBMS-Update output. This can be achieved easily using the Visual Expression Builder. The following section describes the procedure for specifying the Where clause using the Visual Expression Builder.

Specifying the Where clause using the Visual Expression Builder

1. The Where clause appears as a separate field in the Output Structure Panel for RDBMS-Update and RDBMS-Delete. To specify the Where clause, first select the Where node in the output structure and then click the **Funclet** tab. The Where node appears in the Funclet easel as shown in Figure 9.5.20.

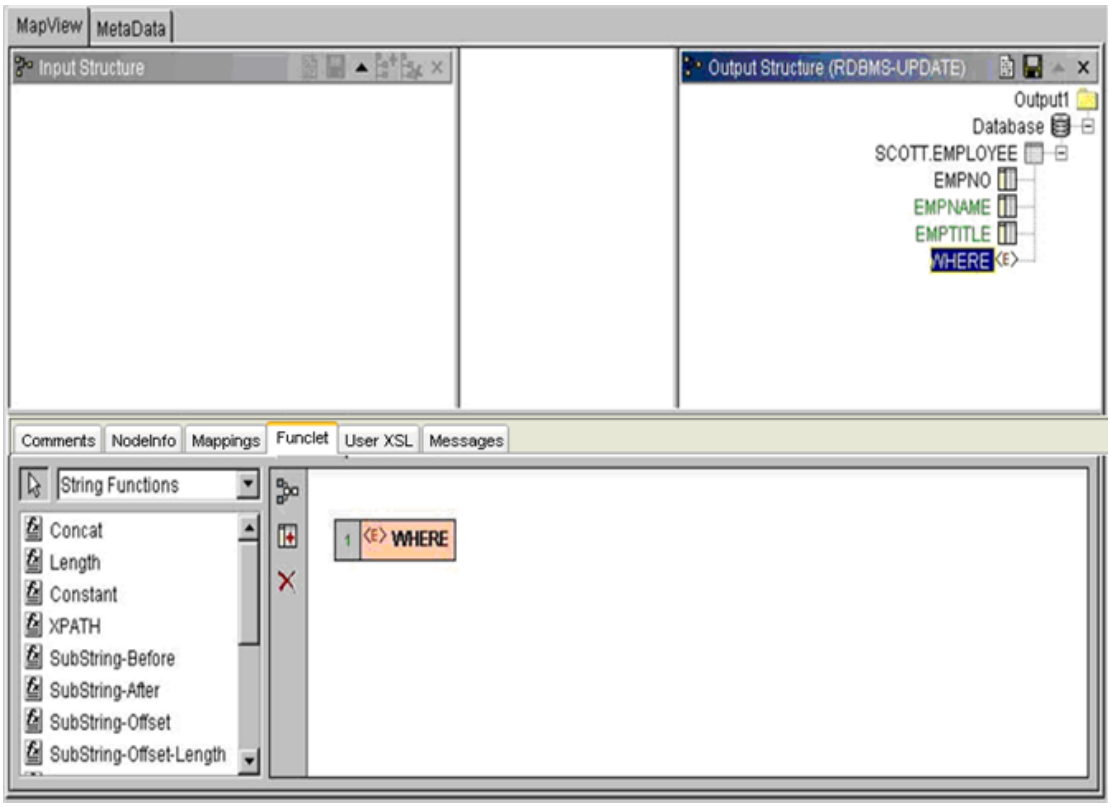


Figure 9.5.20: Selecting the Where node in the Output Structure Panel

2. Next, click on the **Insert Column(s)** icon in the Visual Expression Builder.

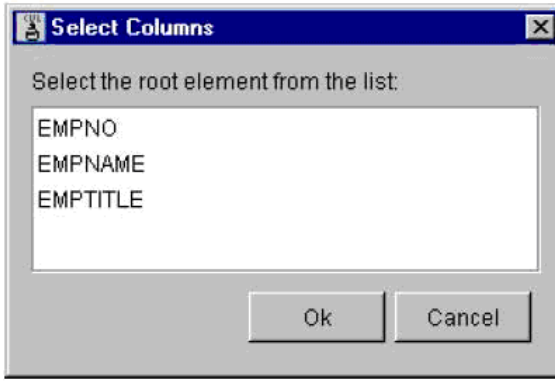


Figure 9.5.21: Selecting New Columns

3. The **Select Columns dialog box** is displayed, as shown in Figure 9.5.21. Select the columns you wish to add in the **Select Column dialog box** and click **Ok**. The selected columns are added to the Funclet easel as shown in Figure 9.5.22.

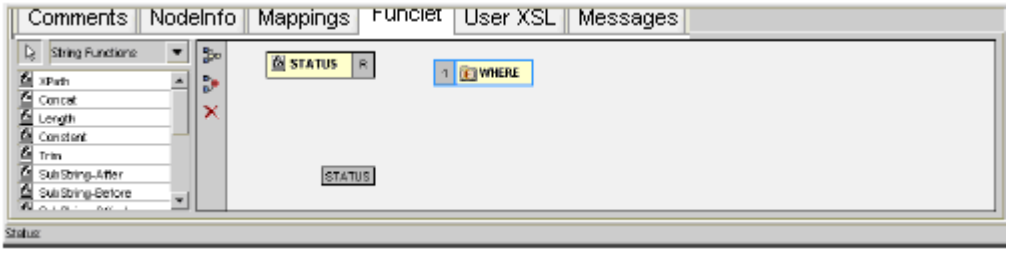


Figure 9.5.22: Selected Columns are added

4. You can build condition expressions using the selected columns and SQL Functions such as AND, OR, Like, and Between available in the Function palette. For this example, we have built an expression that evaluates if the Status column has the value 'Active' using the = and Constant functions, as shown in Figure 9.5.23.

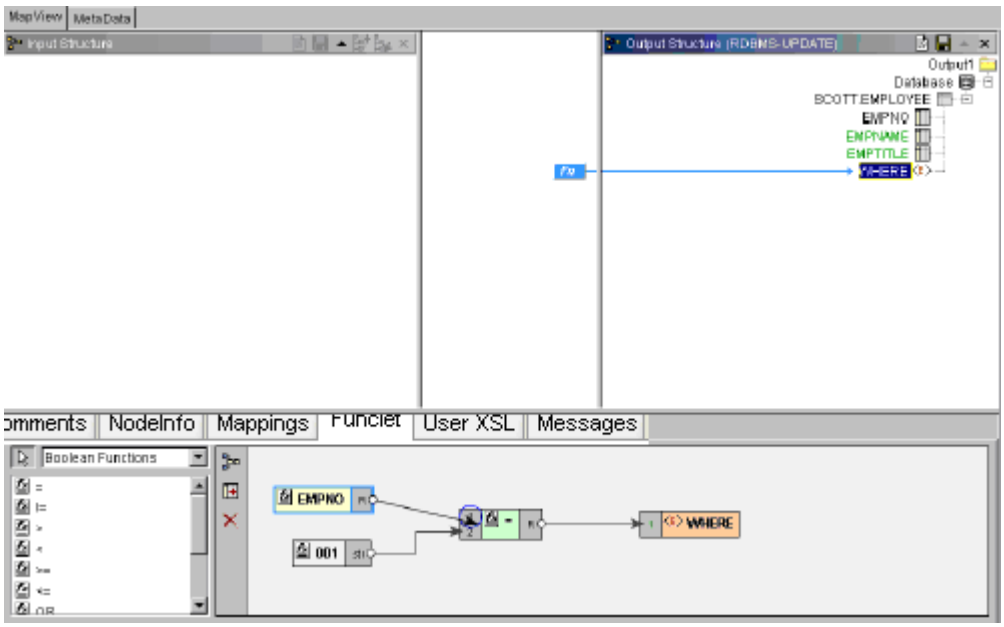


Figure 9.5.23: A Defined Where clause

5. You can validate, test, and save this transformation.

9.5.7.3 Mapping XML Formats to RDBMS-Delete Queries

Mapping an XML input structure to the RDBMS-Delete output structure is a simple one-step procedure, as the RDBMS-Delete query does not require a Set clause:

Specify the Where clause: You must specify a Where condition for the RDBMS-Delete output. This can be achieved easily using the Visual Expression Builder

9.6 Adding User XSLT

Mapper also allows you to customize the output of the transformation by adding your own xslt code to the generated XSLT. You can add XSLT code snippets before and after the beginning tag<> of an element and before and after the end tag </> of an element in the XSLT. By enabling this, Mapper allows you to further refine on the auto-generated output.

Let's take an example where the Mapper generates an output that contains elements not required by the user. In this example, the Mapper generates an output which contains elements that is not mapped. The mapping has an output structure in which the parent element is not mapped but the child elements are mapped, Fiorano Mapper does not generate the if conditions around this unmapped parent element as a result of which this element is generated in the output.

To avoid the generation of unmapped elements in the output, there should be an if condition around <unmapped> element in XSLT whose condition is OR of both the child nodes' if conditions.

Under such conditions, you can use the **User XSL** feature to customize the output and avoid the generation of unmapped tags.

1. Right-click the <unmapped element> in the output structure and select the **User XSL** option from the shortcut menu as shown in Figure 9.6.1.

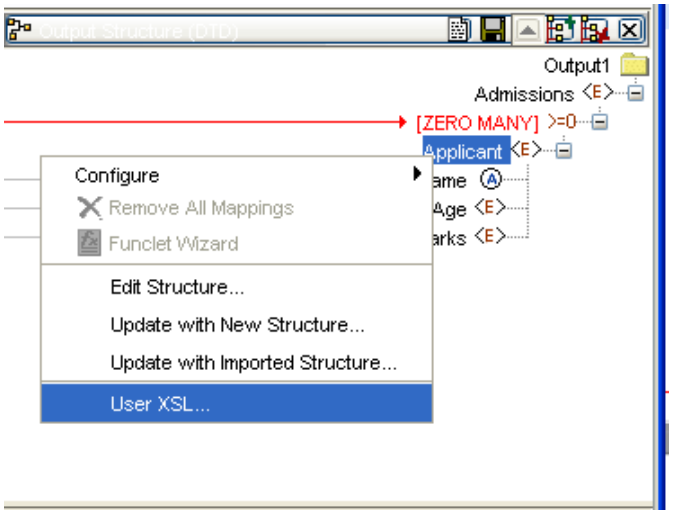


Figure 9.6.1: Selecting the User XSL option from shortcut menu

2. A dialog box appears which contains the xslt script. The xslt script displayed in this dialog box is partially editable. The editable regions are shown in Figure 9.6.2.

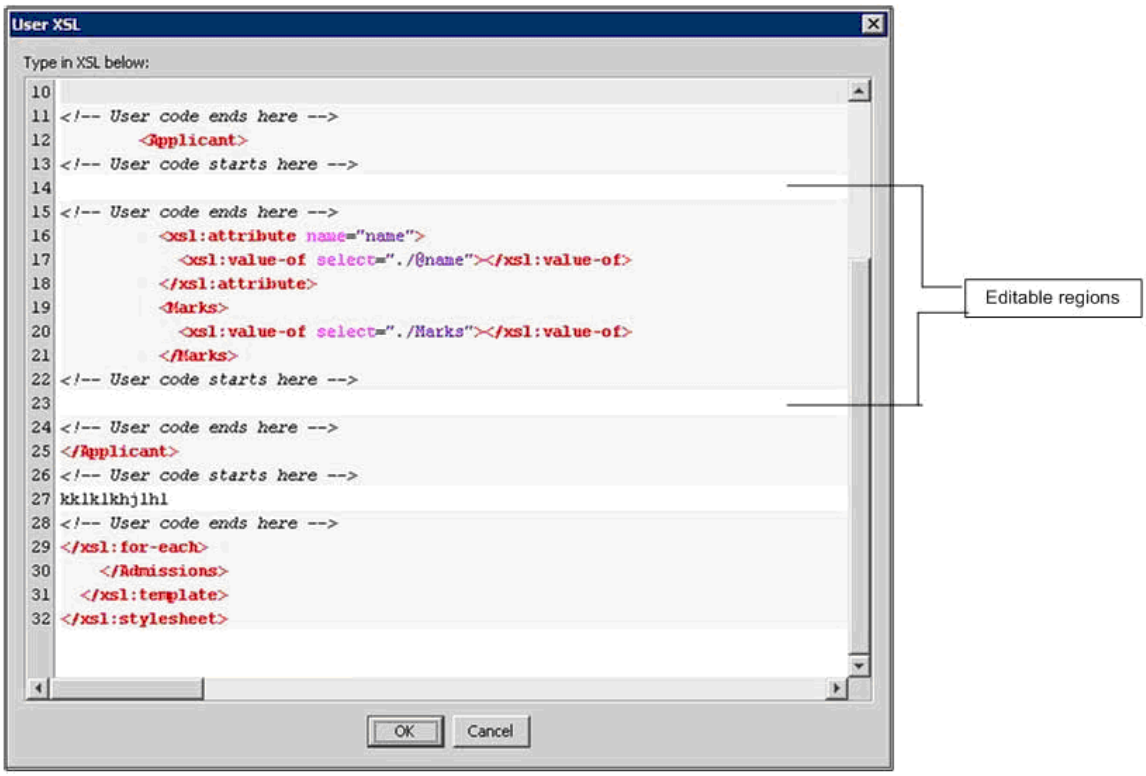


Figure 9.6.2: Editing the user xsl

As shown in Figure 9.6.2, XSL snippets can be added in the following four places:

- just above <element>
- just below <element>
- just above </element>
- just below </element>

3. Add the required if code snippet in these regions.
4. Click the **OK** button and then test it using **Test** option as described in the section 9.7 Testing the Transformation**.**

9.7 Testing the Transformation

To test the transformation that you have created, perform the following steps:

1. Click **Tools** > **Test** in the Fiorano Mapper's menu bar, as shown in Figure 9.7.1.

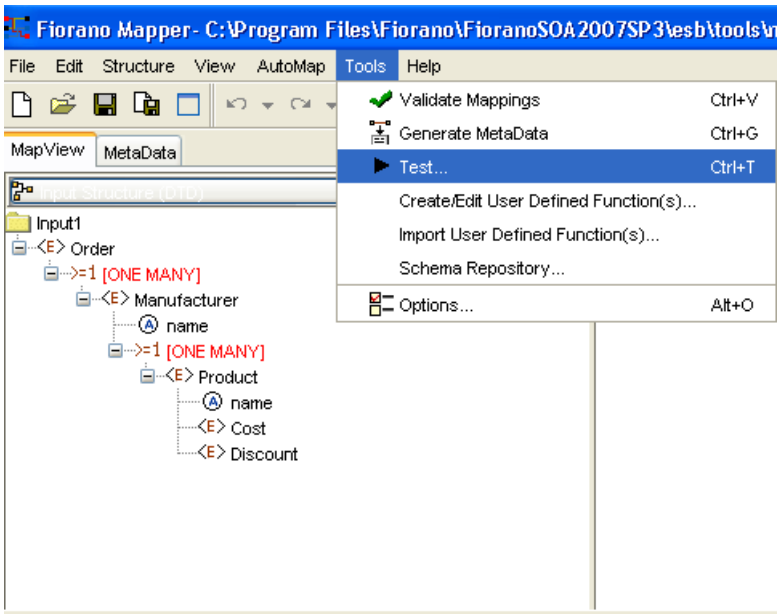


Figure 9.7.1: Invoking the Test option

2. The **Test XSL** dialog box is displayed, as shown in Figure 9.7.2. Click the **Input XML** tab.

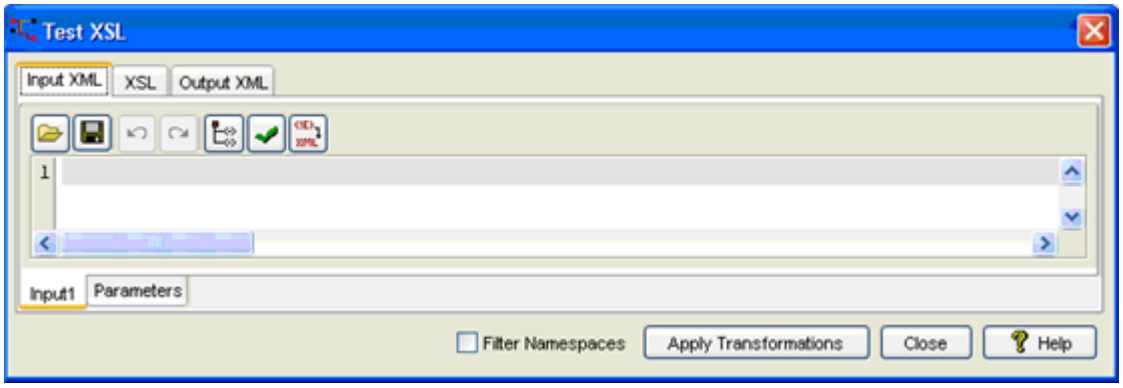


Figure 9.7.2: Generating the Input XML

3. Click **Generate Sample XML** to create a sample input XML, the **Generate Sample XML** dialog box is displayed, as shown in Figure 9.7.3. The default values are appropriate in most situations.

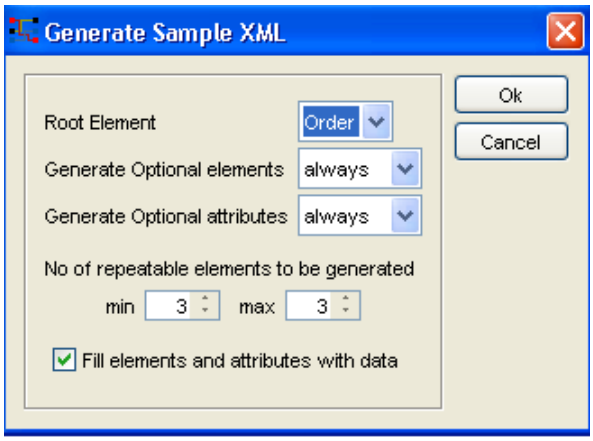


Figure 9.7.3: Selecting the sample Input XML generation options

4. Click **Ok** to generate a sample XML. The sample XML is generated in the Input XML tab as shown in Figure 9.7.4.

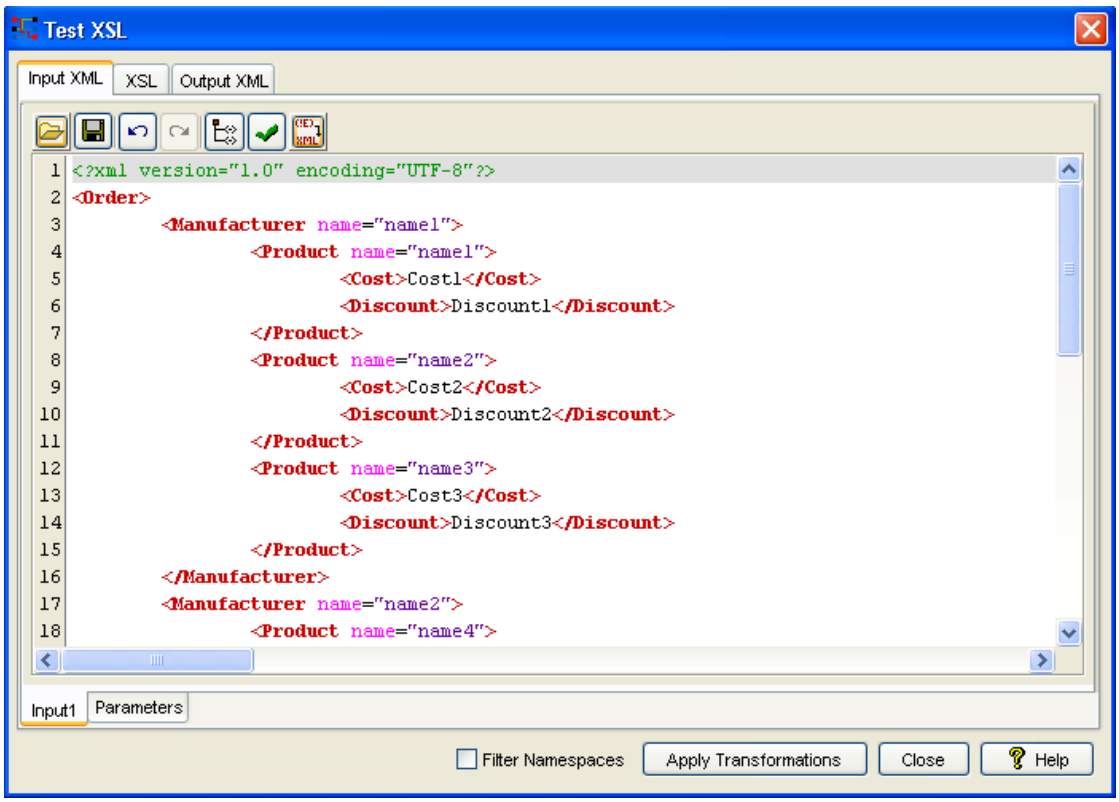


Figure 9.7.4: A Sample Input XML

5. Click on **XSL** to switch to the XSL tab. The XSL tab is displayed, as shown in Figure 9.7.5.

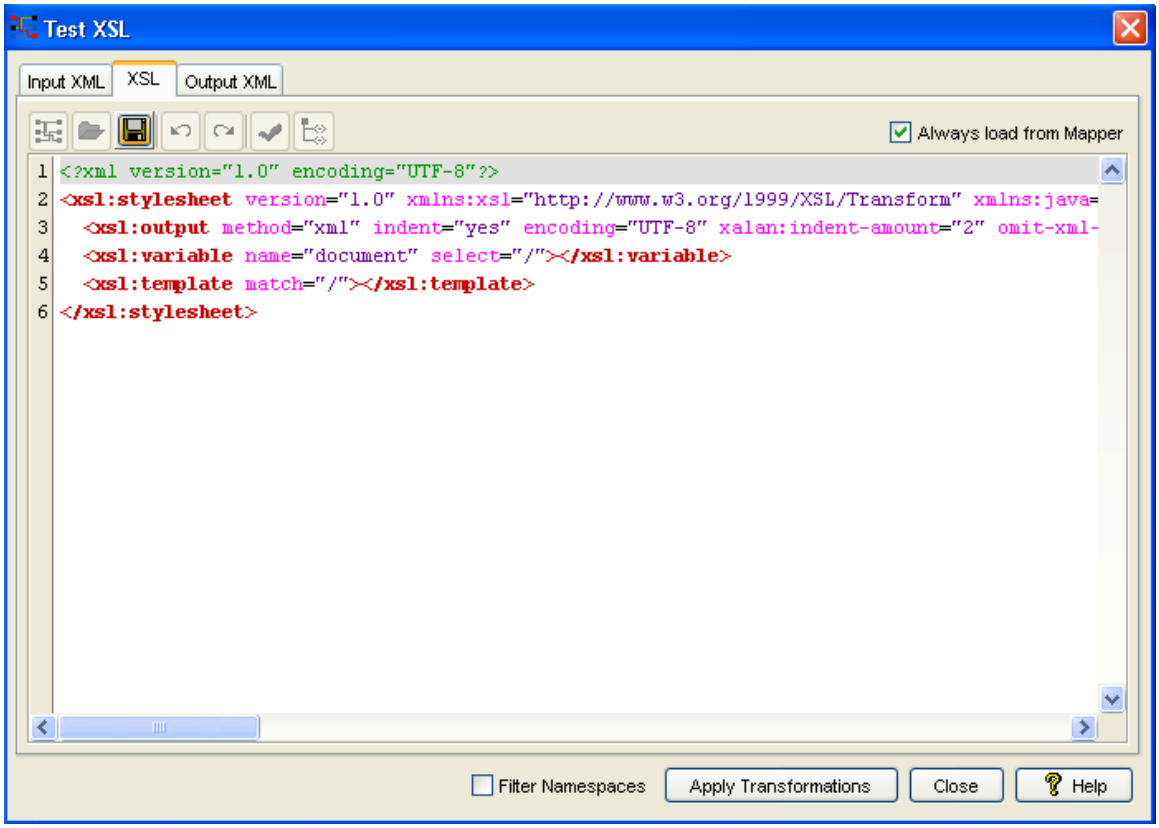


Figure 9.7.5: Loading another Transformation Script

6. You can also modify the transformation generated automatically from the mappings to modify or use some other transformations such as:
   - Deselect the **Always load from Mapper** option at the top-right corner of the XSL tab.
   - Click **Load from file** to load an XSL file, or make changes to the displayed transformation script.
7. Click the **Apply Transformations** button to test the defined transformation.

8. The output XML is displayed in the Output XML tab, as shown in Figure 9.7.6.

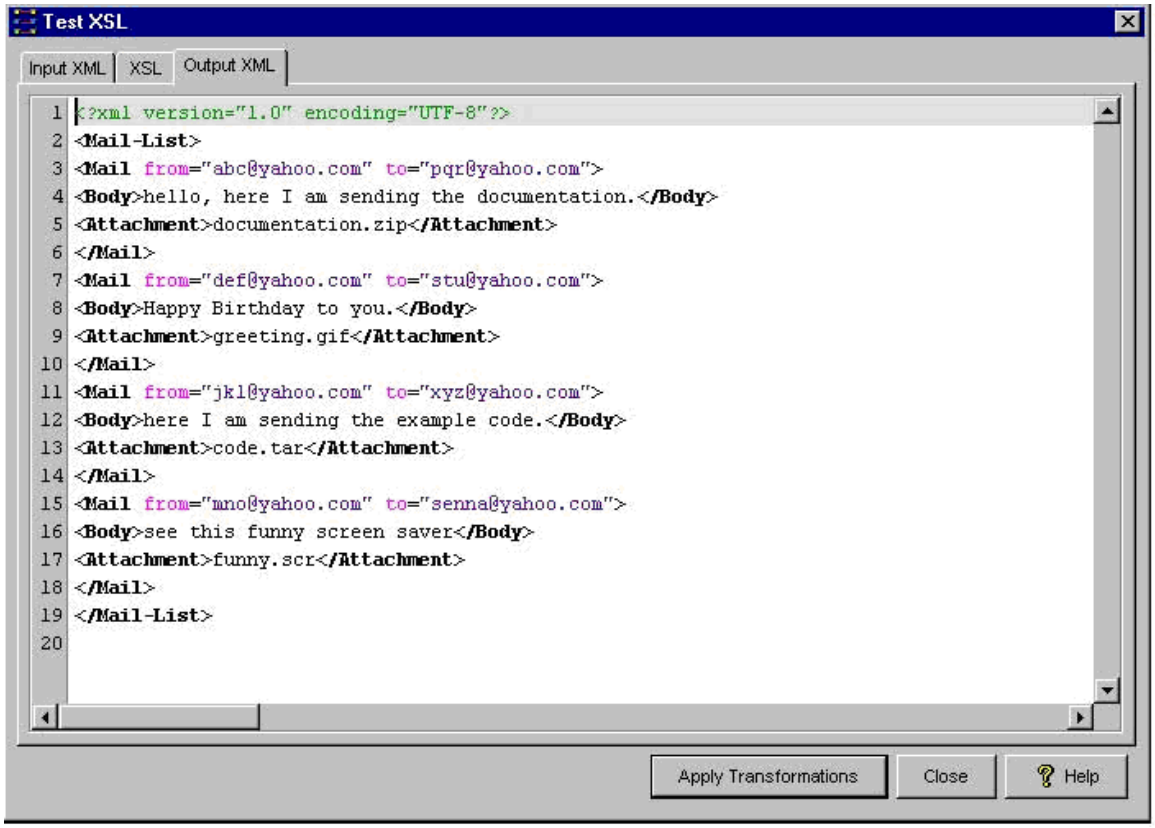


Figure 9.7.6: The Output XML resulting from the Transformation

9.8 Managing Mappings

Creating mappings is as simple as dragging an input node and dropping it on a target structure node.

9.8.1 Exporting Mappings to a File

To export the Mappings to a File:

Click File > Save As

Enter the file name in which you want to export the project and click on **OK** button. The project gets saved with default file extension (.tmf).

9.8.2 Importing Project from the File

To import the project from a file:

1. Click File >Open
2. Select the project file you want open and click on OK.

9.8.3 Validating All Mappings

To validate the existing mappings between the Input and Output Structures,

Click **Tools** from the menu bar and select **Validate Mappings**.

Or, click the icon on the toolbar.

Or, right-click on the line panel and select **Validate Mappings**.

If the validation is successful, a dialog box is displayed as **Validation Successful** message.

If the mappings are invalid, a dialog box is displayed as **Validation Failed** message.

The **See MetaData View for errors** message is displayed. The message window also displays the invalid mapping details. You can see the validation errors in the **Error Messages** panel of **Metadata** view.

9.8.4 Displaying All Mappings

To view all the existing mappings between the input and output structures, perform the following steps:

3. Click View > Show All Mappings

   Or, click the icon on the toolbar

   Or right click in the line panel and select **Show All Mappings** from the shortcut menu

9.8.5 Removing Mappings for a Node

In order to remove all mappings for a particular node:

1. Right-click on the corresponding target node in the Output Structure Panel.
2. Click **Remove All Mappings**.

Alternatively

1. Select one of the map lines to the target node in the line panel.
2. Right-click on one of the highlighted map line of the selected map line and click **Delete**

9.8.6 Copying functions in a Mapping

You can copy functions within a mapping project and across mapping projects. To copy a function

1. Select the function in the funclet panel and press <Ctrl+C> as shown below in Figure 9.8.1.

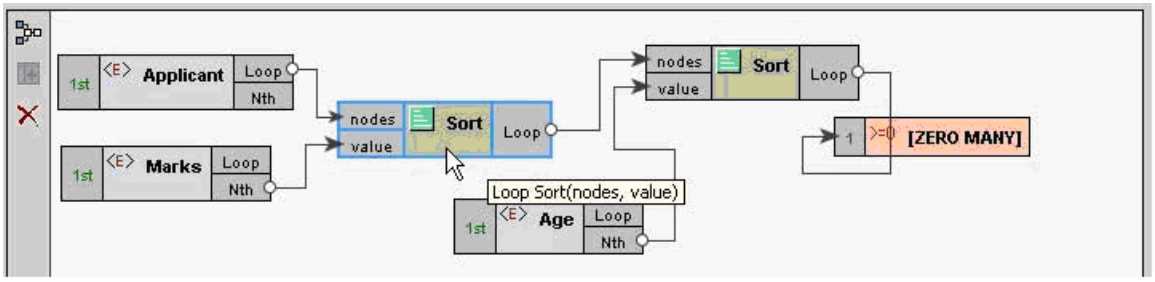


Figure 9.8.1: Copying a function

2. Press <Ctrl+V>. The function is pasted in the funclet tab and can be reused within the mapping. The copied nodes are pasted exactly above over the original nodes. You need to move these nodes to view the original and copied node

9.8.7 Clearing All Mappings

To clear all the mappings between the Input and the Output Structure,

1. Click **File** from menu bar and select **Clear Mappings**.

   Or click the icon from the tool bar.

   Or right-click on the line panel and select **Clear Mappings**.
2. A warning dialog box is displayed showing a confirmation message. Click **Yes** to remove all the existing mappings between the input and output structures.

9.8.8 Clearing Data

Clearing Data is meant to unload both the input and output structures along with the mappings defined between them. To clear data:

1. Click **File** from the menu bar and click **Clear Data**

   Alternatively, you can click the icon from the toolbar.

Click **Yes** in the warning dialog box that is displayed.

9.8.9 Modifying the RDBMS Output Structure Settings

For RDBMS output structures, you can modify the tables selected and their dependencies at any time, using a shortcut menu.

This shortcut menu is displayed when you right-click the Database node in any of the RDBMS output structures, as shown in Figure 9.8.2.

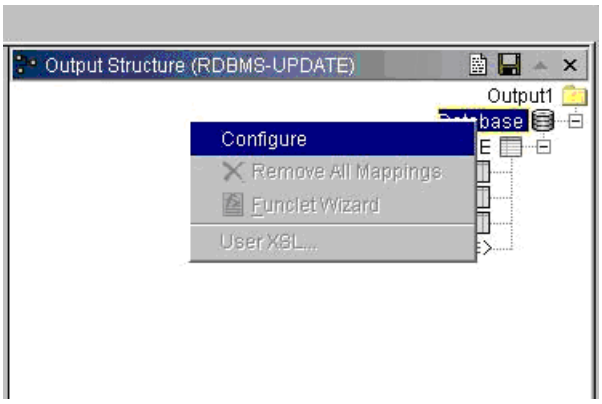


Figure 9.8.2: Modifying the settings of an RDBMS Output Structure

9.8.10 Configuring Mapper Settings

Fiorano Mapper allows user to configure the following settings through **Options** dialog box.

To view the **Options** dialog box:

1. Click **Tools** from the menu bar and select **Options**.

   Alternatively, click on icon on the toolbar.

General Options

- **Look and Feel:** Select the UI theme from the dropdown list.
- **Default directory:** Select the default location for the **File** and **Open** dialog box.
- **Datatype Conversion:** Select this option to convert the datatypes while linking nodes in the funclet. This needs to be done to control automatic datatype conversion in Mapper and avoid datatype mismatch.
- **Enable Automatic For Mappings:** Select this option to automatically map the parent nodes once the child nodes are mapped in a transformation. If user defines a mapping for a child element which can occur multiple times in its parent, then Mapper generates a FOR mapping automatically for the parent's control nodes. XML specific options.

XML

- **XML Schema Parser:** Select the schema parser from the dropdown list. This parser validates the xml file and its corresponding schema. Selecting this option affects only the new projects created after this. When an existing project is opened Mapper decides the parser that needs to be used from the project file. Fiorano Mapper provides two parsers - Xerces and Castor to validate xml schema.
- **Format XML:** Formats the XML.
- **Canonical support for XSL and XML:** Makes a Canonical form of the XSL and XML.
- **Use Normalize Function in XPath:** It enforces the use of Normalize-String function whenever using the XPath in logical functions.

9.8.11 Managing XSLT Properties

You can also manage the XSLT properties of the output XSLT. To do the same:

Click **View > XSLT Properties**. The XSLT Properties dialog box is displayed as shown in Figure 9.8.3.

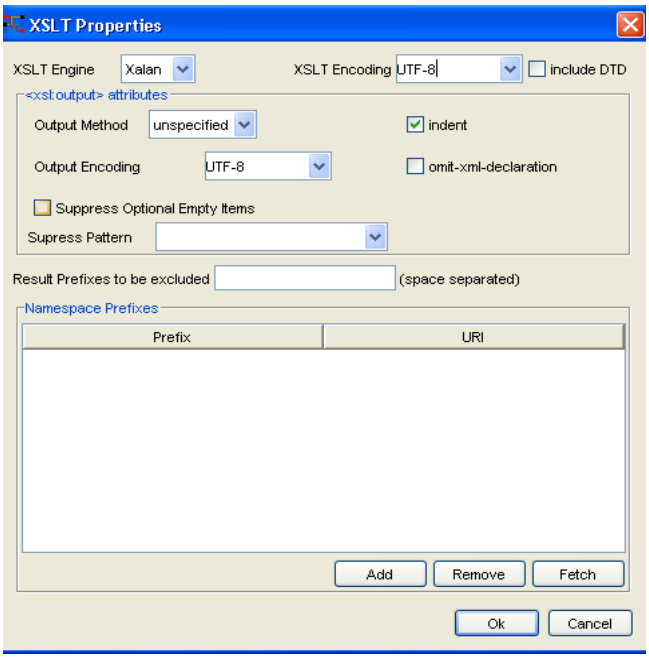


Figure 9.8.3: Viewing XSLT Properties

This dialog box contains the following components:

- **XSLT Encoding;** Specifies the encoding of the generated XSL.
- **Include DTD:** Select this option to include the internal specified DTD in the transformation output. By default, this option is disabled.
- **<xsl output: attributes>**
    a. **Output Method:** Select the method of output after transformation from the dropdown list. The method of output can be HTML, XML, or text.

b. **Indent:** Select this option to indent the output XSLT.

c. **Output Encoding:** Specifies the encoding of the generated output XSL

- **Omit-xml-declaration:** Specifies whether the output XML generated should contain XML declaration or not.
- **Suppress optional empty items:** Select this option for defining a mapping to an output node, always generate the output node in output xml, event input xml has no matching node. It is some times desirable not to generate optional output nodes if no input matching node is found in input xml. This requires using a conditional mapping. You can specify such conditional mapping by using "User XSL" feature. Mapper can generate such conditions automatically for optional elements if this option is selected.

9.9 Customizing the Mapper User Interface

The Fiorano Mapper is equipped with preset themes to improve the look and feel of the user interface. You can alter the appearance of the tool by selecting one of the following themes:

- Windows
- Metal
- Motif

To set the theme, perform the following steps:

1. Click **Tools > Options**
2. Select an option from the **LookAndFeel** drop-down list, as shown in Figure 9.9.1.

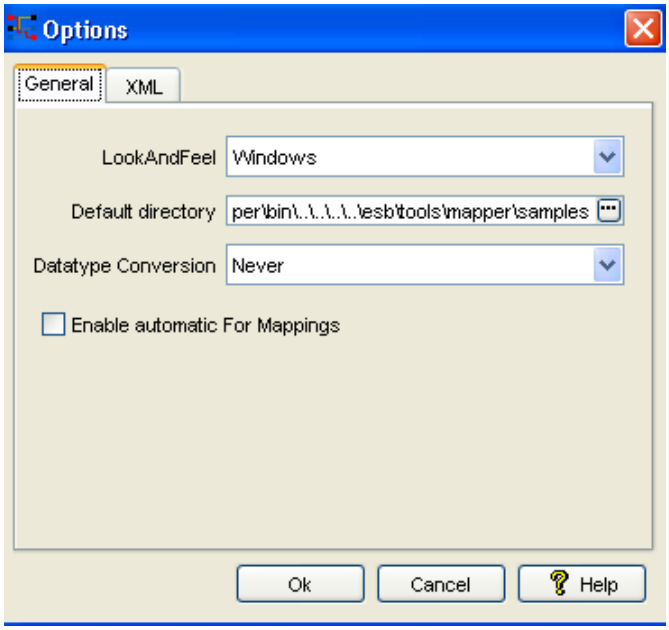


Figure 9.9.1: Options Dialog Box

3. Click the **Ok** button**.** For example, if you select **Metal**, the user interface looks as shown in Figure 9.9.2.

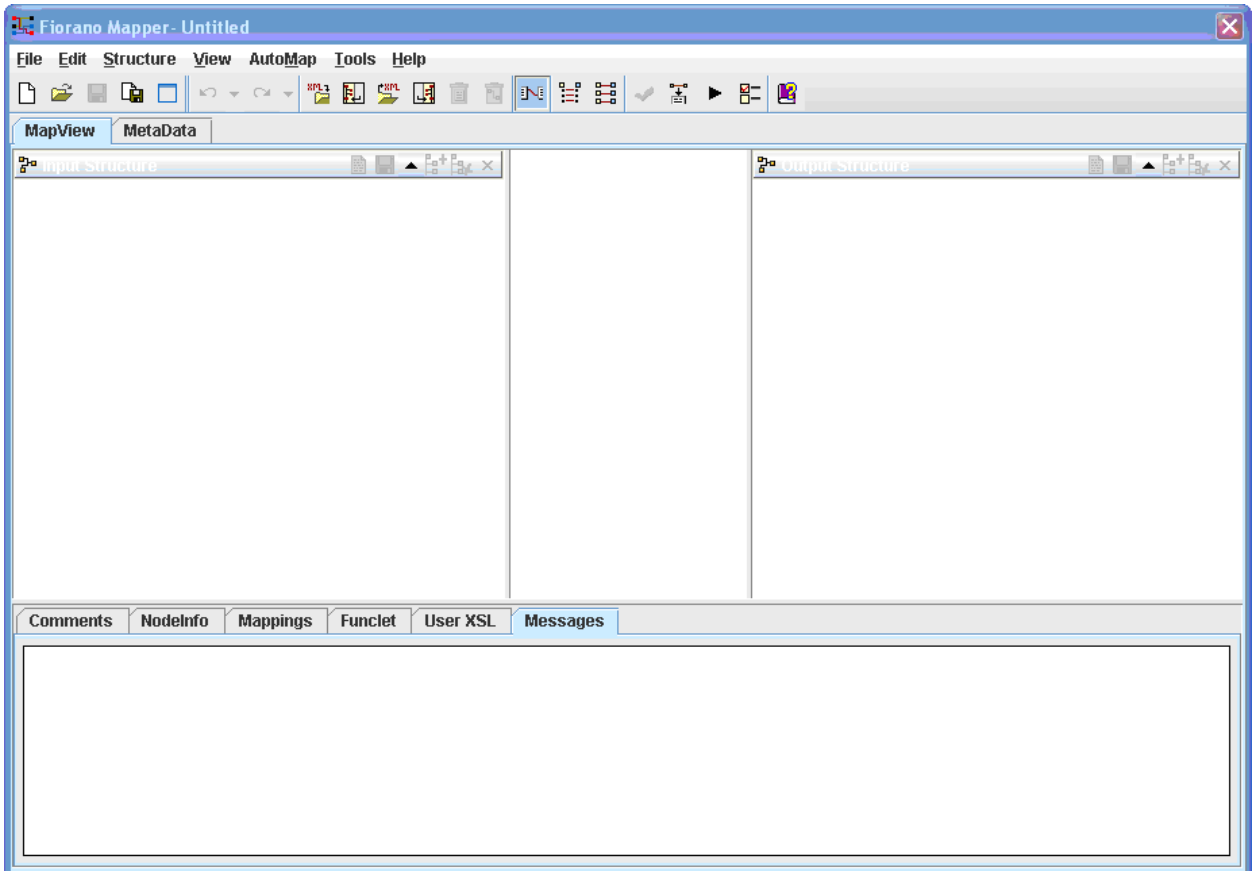


Figure 9.9.2: Fiorano Mapper User Interface

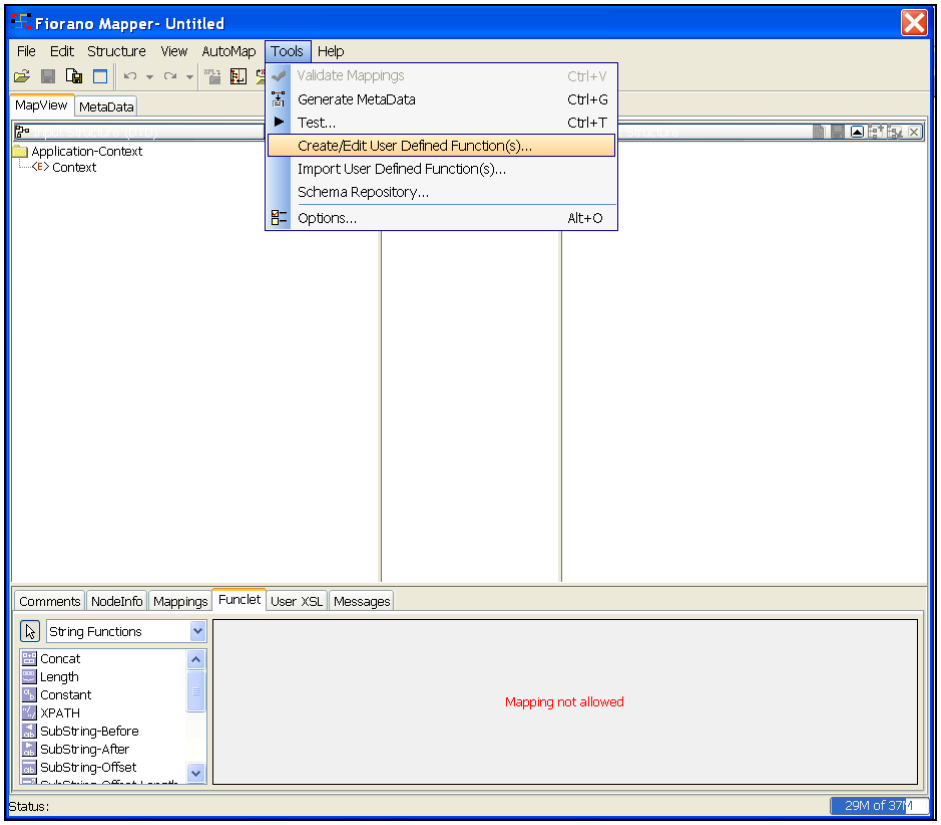
9.10 Fiorano Mapper –Custom Funclets

9.10.1 Creating Custom Funclets

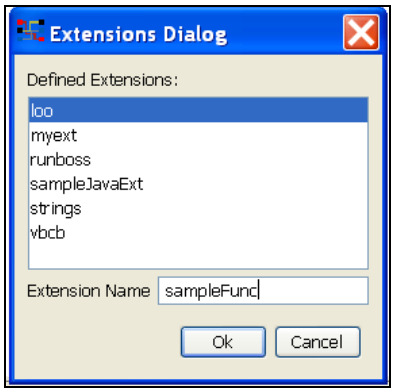
Apart from the prebuilt funclets, user defined funclets can also be integrated in Mapper. Custom functions can be built using Java or JavaScript. The following sections explain how to integrate JavaScript and Java functions in Mapper.

9.10.1.1 Integrating JavaScript functions

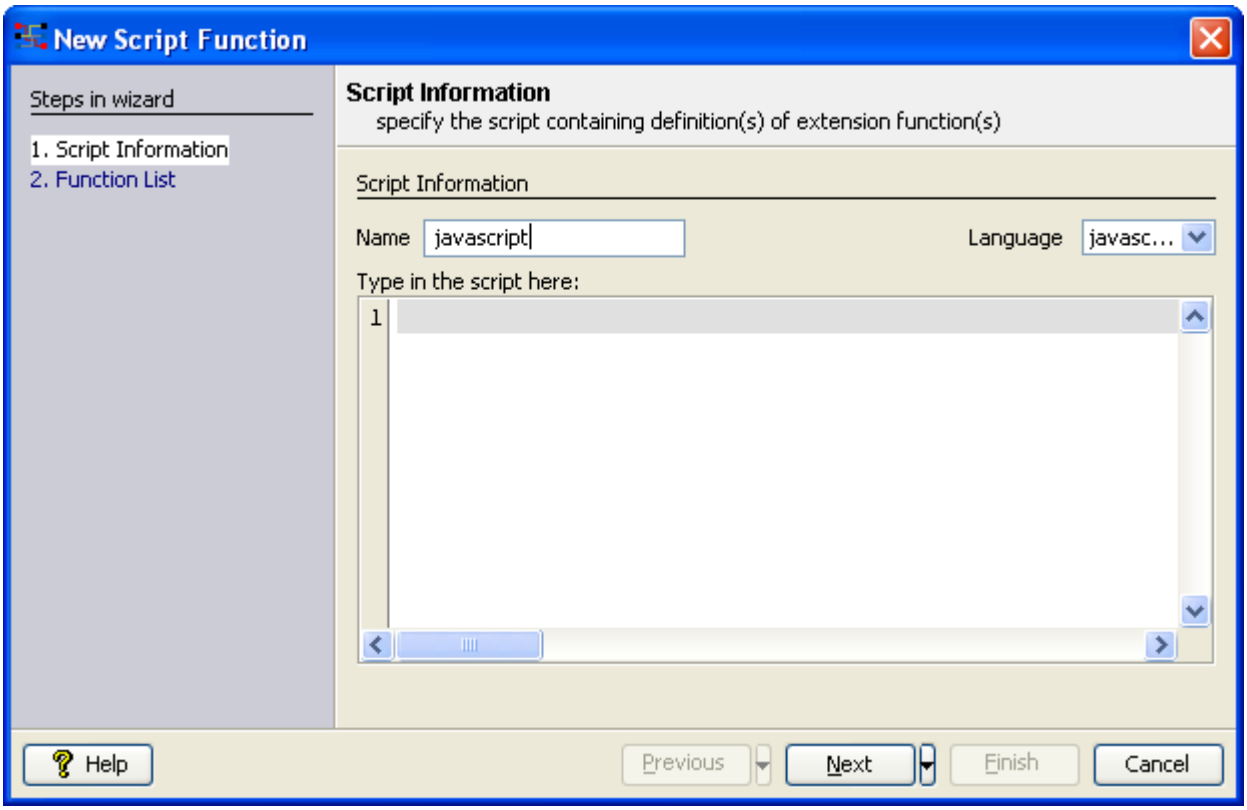
1. Launch the Mapper
2. Select **Create/Edit User Defined Function(s)** from Tools menu. The **Extensions Dialog** appears.



3. Type in an extension name (this can be any unique name). Click **OK**.

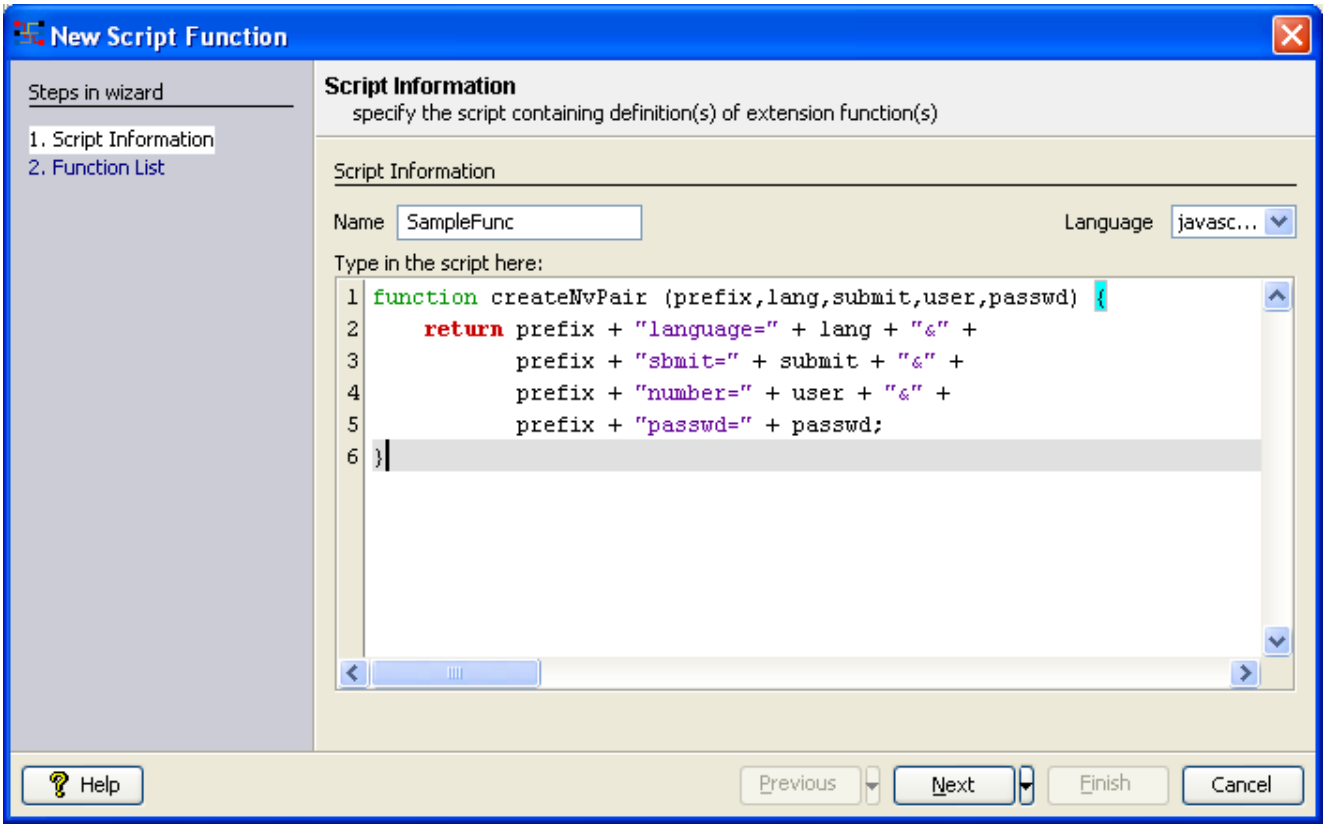


The **New Script Function** wizard is launched. Choose language as **JavaScript**.

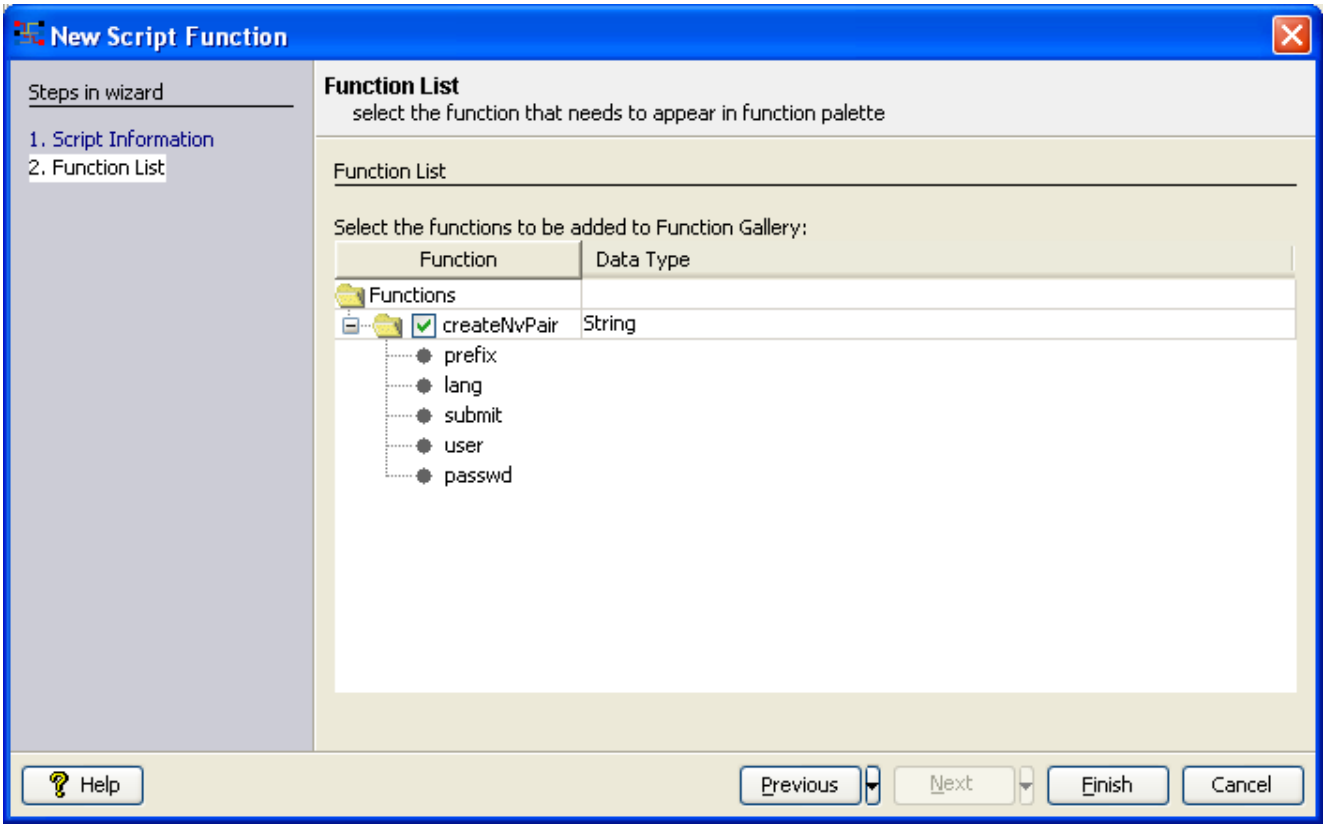


4. Define JavaScript function(s). A sample JavaScript function createNvPair is shown in the Figure below which accepts prefix, language, user and password as arguments and returns an NVPair string.

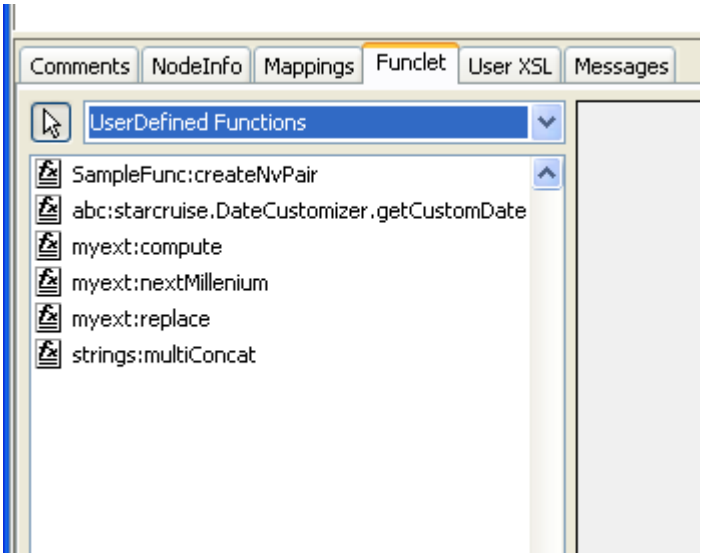
Note: Multiple functions can also be defined in this area.

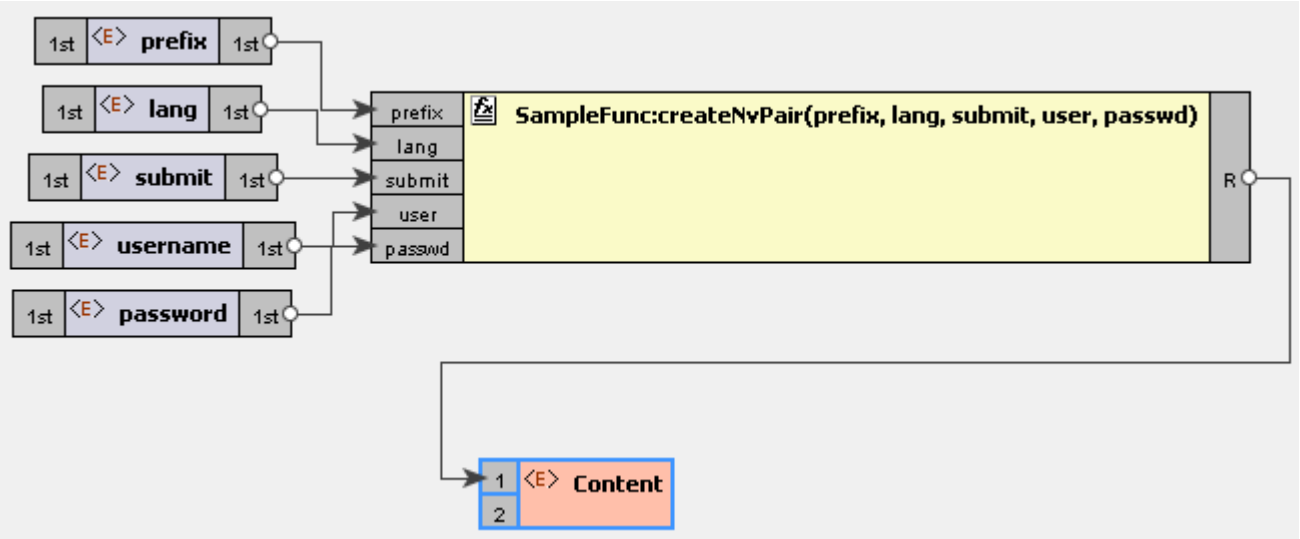


5. Click **Next** and all the functions defined in Script Information panel are displayed in F**unction List** panel. Select the functions to be integrated in Mapper and click **Finish**.



6. Now the newly defined function createNvPair is integrated with the Mapper and is available under **UserDefined Functions** category. This funclet can be used while defining mappings.





9.10.1.2 Integrating Java functions

Creating the jar file

Build the function logic in a java class, create a jar file and place it in the folder FIORANO_HOME/esb/lib/ext.

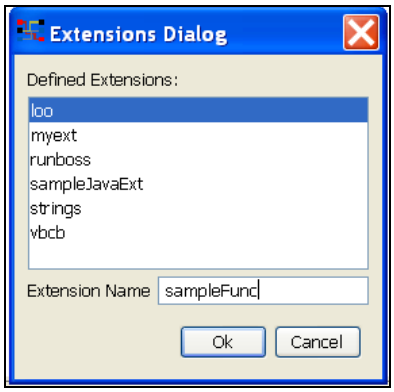
Follow the below points while building the jar file

- Package has to be necessarily mentioned in the java file.
- The java class need not have a main method.
- The class should be declared Public and the methods Public Static
- The class can have multiple methods. All of them will be available as funclets(provided they are public static)

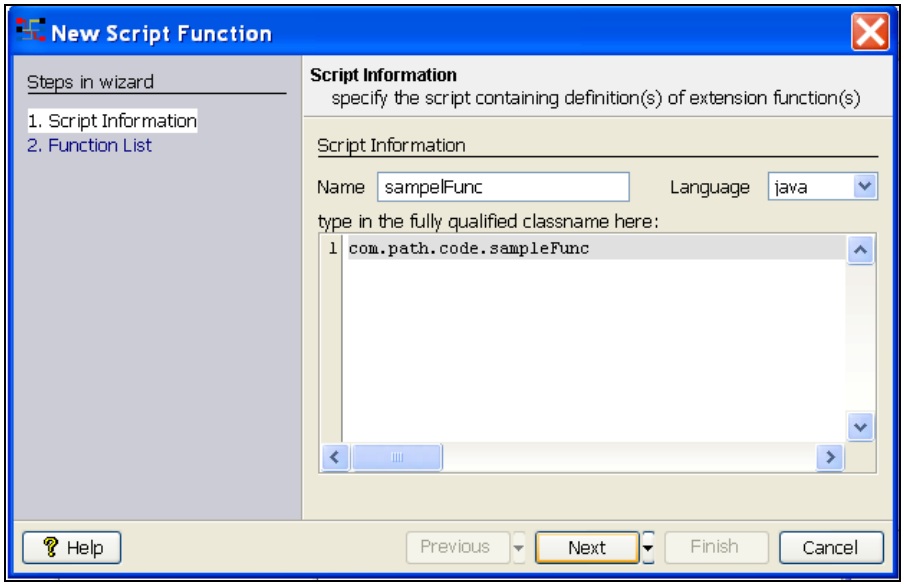
Including the funclets in the Mapper

Note: You cannot add custom java funclets to Mapper when it is invoked from route transformation, it works when invoked using an XSLT component.

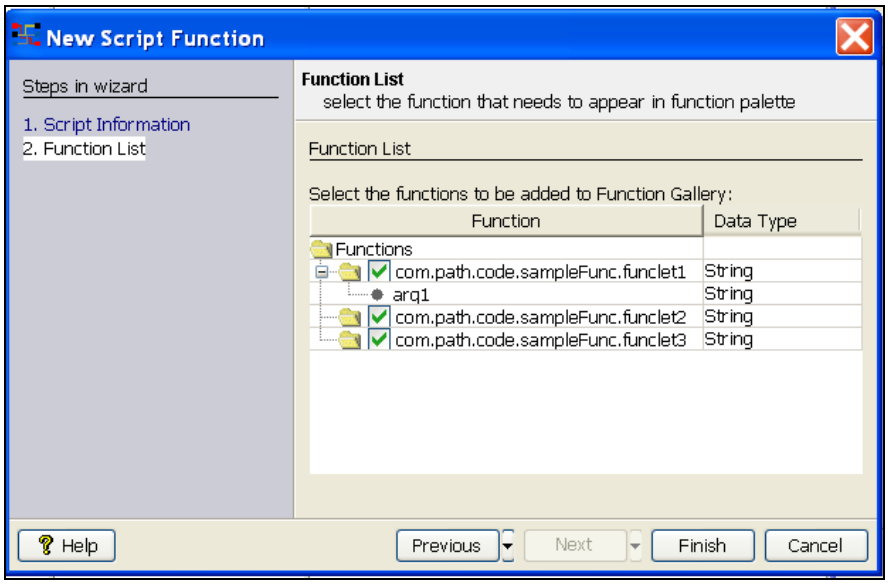
1. Launch the Mapper
2. Select **Create/Edit User Defined Function(s)** from Tools menu. The **Extensions Dialog** appears.
3. Type in an extension name (this need not necessarily be the name of the jar file). Click **Ok**.



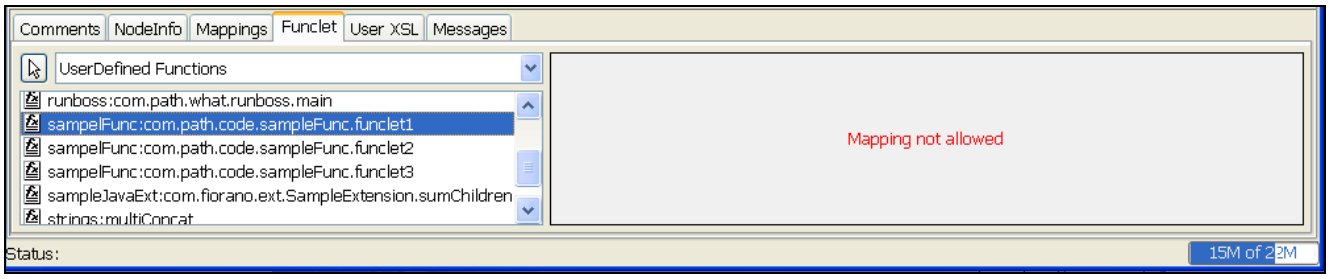
4. The **New Script Function** wizard is launched. Choose language as java. Type in the qualified class name in the text box and click **Next**.



5. The **Function List** panel appears. This lists all the Public static functions available in the class file. Select the desired functions from the functions list and the data type that they return. The data type specified here is used for verifying the mapping connections of that function to other functions. For example, a function that returns a decimal data type cannot be used as input for a function that expects a boolean input value.

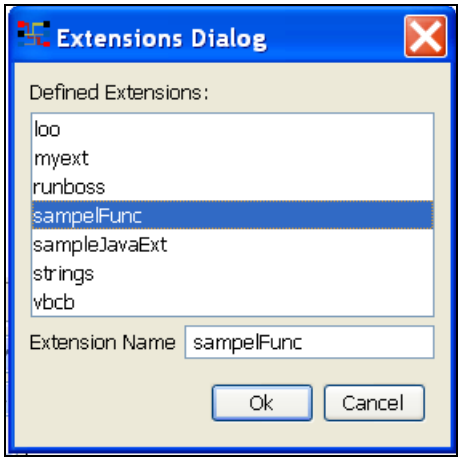


The selected functions are added to the category UserDefined Functions as shown in the following figure.



9.10.2 Editing Existing User Defined Funclets

1. Select **Create/Edit User Defined Function(s)** from Tools menu.
2. In the **Extensions Dialog** box, select an existing extension and click **Ok**.



3. The class name can be edited to reflect a new class. If the same class is to be used, click **Next**.

4. The functions can be selected or deselected. The input and return data types can also be edited. Click **Finish** to save and reflect changes.

9.10.3 Updating UserDefined funclets

If the user defined funclets have to be updated (assuming the functions that are currently used in Mappings are not modified. Only new functions have been added to it), replace the jar file in FIORANO_HOME/esb/ext and restart the Mapper tool.

Follow the usual steps to add new funclets:

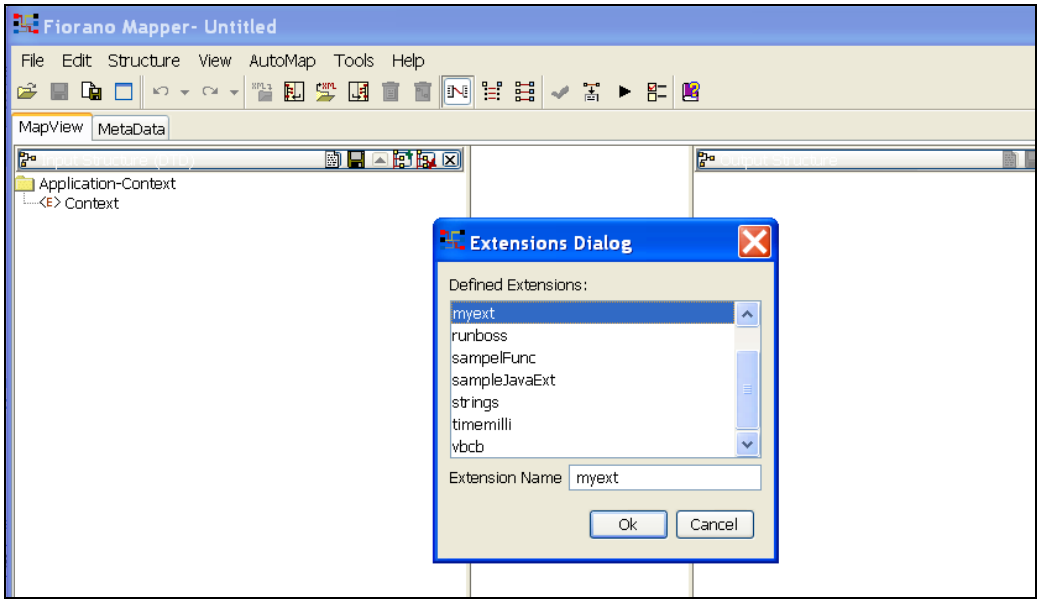
1. Select the name of the extension, go next.
2. Since an existing extension is being modified, class name would be same as before, go next.
3. Check the newly added and modified functions. Finish the wizard to save the settings.

9.10.4 Removing User Defined Funclets.

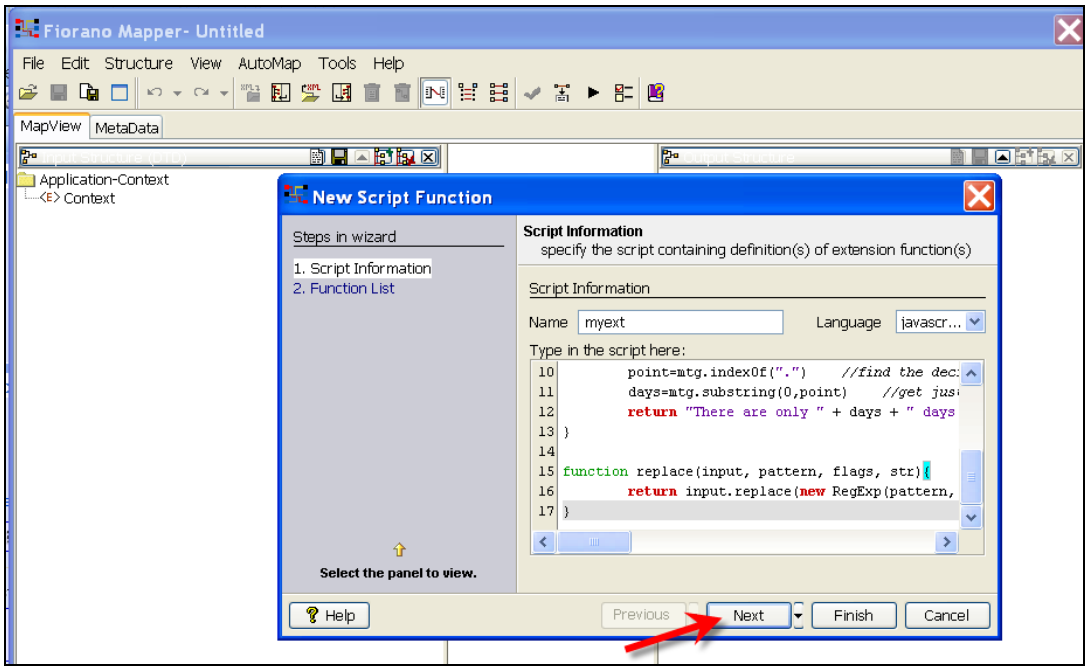
9.10.4.1 Removing User defined funclets from Fiorano Mapper

Follow the steps given below to remove a funclet from an extension. (Please refer to the following sub-section for the procedure to delete an extension.)

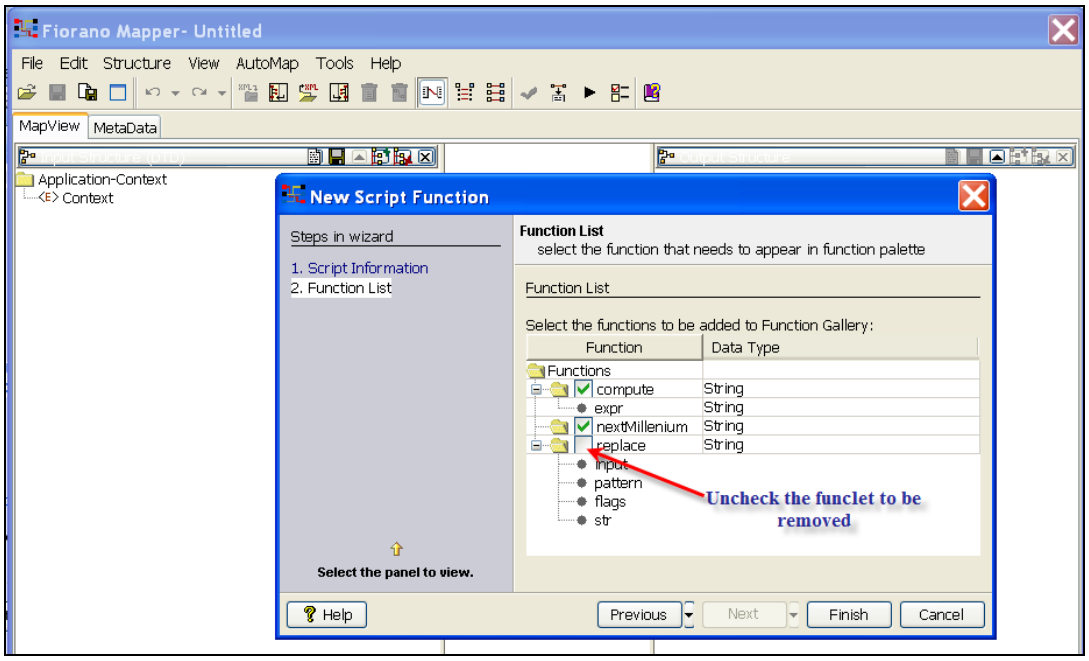
1. Launch the Mapper.
2. Select **Create/Edit User defined Function(s)** from Tools menu.
3. Choose the extension that contains the funclets to be deleted



4. Click **Next** on **Script Information** panel.



5. Uncheck the funclets to be removed.



6. Click **Finish**.

Note: The funclets thus removed can be re-included by following the same steps and choosing the funclet in the Function List panel. The limitation here is that an extension must have at least funclet that is chosen.

9.10.4.2 Deleting entire extension

1. Browse to the folder <FIORANO_HOME>/esb/tools/mapper/bin

2. Open the file functions.repository for editing
3. Search from the tag **Extension** that has the name of the extension to be deleted as the value for its **name** attribute.

```
11  <Extension name="foo" language="java">
12    <Script><![CDATA[one.two.three.four]]></Script>
13    <Function name="one.two.three.four.main" return="String">
14      <Argument name="arg1" datatype="String"></Argument>
15    </Function>
16  </Extension>
17
18  <Extension name="myext" language="javascript">
19    <Script><![CDATA[function compute(expr){
20    return eval(expr);
21  }
22
23  function nextMillenium(){
24      d = new Date()     //today's date
25      mill=new Date(3000, 00, 01, 00, 00, 00)     //Next millennium start date
26      diff = mill-d     //difference in milliseconds
27      mtg = new String(diff/86400000)     //calculate days and convert to string
28      point=mtg.indexOf(".")     //find the decimal point
29      days=mtg.substring(0,point)     //get just the whole days
30      return "There are only " + days + " days remaining to the start of the next millennium.";
31  }
32
33  function replace(input, pattern, flags, str){
34      return input.replace(new RegExp(pattern, flags), str);
35  }]]></Script>
36      <Function name="compute" return="String">
37        <Argument name="expr" datatype="String"></Argument>
38      </Function>
39      <Function name="nextMillenium" return="String"></Function>
40  </Extension>
41
42
43  <Extension name="sampleJavaExt" language="java">
44    <Script><![CDATA[com.fiorano.ext.SampleExtension]]></Script>
45    <Function name="com.fiorano.ext.SampleExtension.sumChildren" return="Decimal">
```

Delete the entire section

4. Re-launch the Mapper tool; the extension would have been deleted.

Chapter 10: Common Components Configurations

This chapter describes the configurations that are common across a most of adapters. However, if there are any additional components specific details for any of the configurations which are not described here, such details can be found in that component help file.

10.1 Component Instance Properties

Component instance properties appear in the **Properties** pane when a component instance of an application is selected. Component instance properties for a SMTP component are shown in Figure 1.

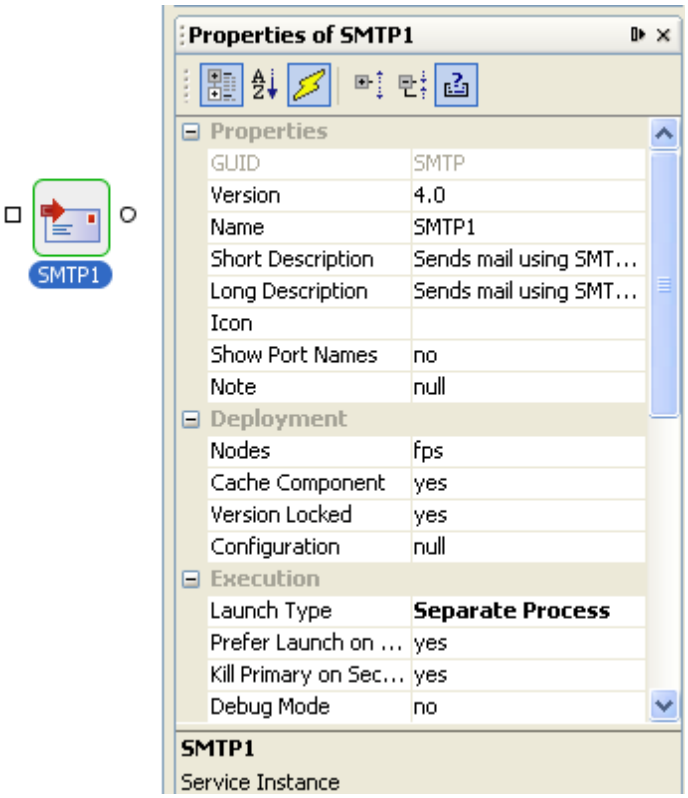


Figure 1: Component instance properties for a SMTP component

10.1.1 Properties

This group contains general properties that identify the component instance or affect the appearance of the component instance.

Version

When there are multiple versions of a component with same GUID, the **Version** property determines the component that this instance represents.

Example: Web Service Consumer has two versions of components, version 4.0 and version 5.0. Version 4.0 uses axis API and version 5.0 uses axis2 API.

Note: It is recommended that the configuration of the component is compatible across versions when developing custom components.

Name

Name of the component instance in the application. This name should be unique in an application.

Short Description

A one line description for the component instance. This description is shown as tool tip for the component. By default, the value defined for **short-description** element in the component's **Service Descriptor** is shown here. If there is no value defined for this property, the default value is shown in tool tip.

Example: For a SMTP component that is configured to send mails using a Fiorano email account, the short description can be changed as shown in Figure 2 to give a more appropriate description.

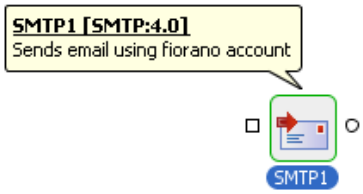


Figure 2: Tool tip showing the changed Short description for SMTP component

Long Description

A detailed description about the component instance. By default, the value defined for **long-description** element in the component's **Service Descriptor** is shown here.

Icon

Icon to be used for the component instance in the application. Click the ellipses button to browse the icon that should be used. Icon with size 32 X 32 is recommended. By default, the value defined for attribute **large** of **icon** element in the component's **Service Descriptor** is used. Removing the value specified against this property will restore the default icon.

Figure 3: SMTP component instance using a custom icon

Show Port Names

Names of ports for the component instance are shown if the value for this property is set to **yes**, otherwise the port names are hidden. When the port names are turned on at an application level, this property has no effect.

Note:

A floating note containing a custom message can be added to component as shown in Figure 4.

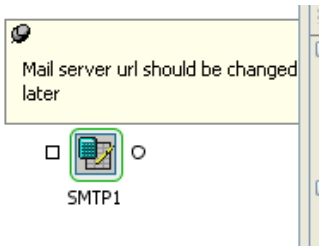


Figure 4: SMTP component with a note containing custom message

10.1.2 Deployment

This group contains properties that effect the way components are deployed in Fiorano environment.

Nodes

A delimited list of Peer Servers from Fiorano network on which this component instance is launched. The component is launched on the first Peer Server in the list that is available in Fiorano network. If the Peer Server on which the component instance is running shuts down, the component instance fails over to the next available Peer Server.

Click the ellipses button to launch an editor to configure the Peer Servers as shown in Figure 5.

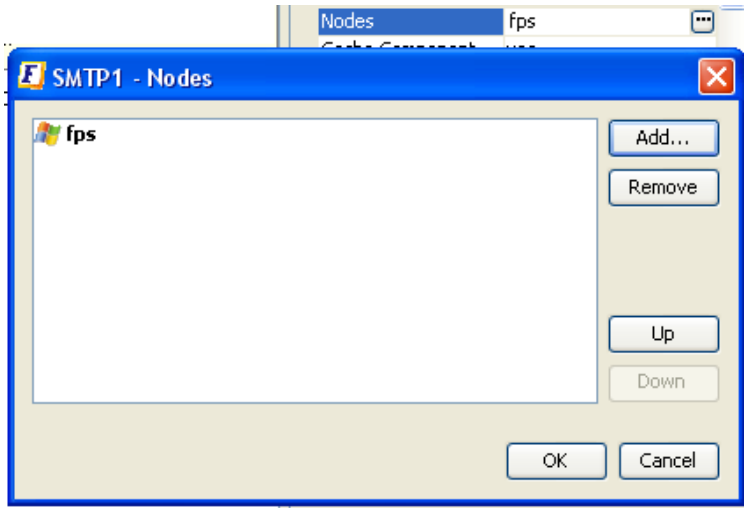


Figure 5: Editor to configure Peer Servers on which the component instance can launch.

7. To add a Peer Server, click the **Add...** button and select Peer Server that has to be added to the list.
8. To remove a Peer Server, select the Peer Server and click the **Remove** button.
9. The order of the Peer Servers can be changed using the **Up** button and the **Down** button.

Note the following:

- If the component instance persists data on the Peer Server on which it is running, then the data is lost when the component instance fails over to a different Peer Server.
- The fail over process is equivalent to manually stopping the component instance changing the Peer Server on which the component instance should be launched and restarting the component.
- Messages that are received by the component instance during the fail over are lost.
- This is not the same as fail over when Peer Server is configured in HA mode.

Cache Component

- **yes**

  The resources required for execution by the component instance are fetched from the Enterprise Server and cached on the Peer Server, if not already done, when the CRC (**Check Resources & Connectivity**) operation is performed.

  A resource can be marked as required for execution in the **Service Descriptor**.

  If the resources are not changed in the Enterprise Server, then they need not fetched every time the application is launched there by reducing the time taken to perform a CRC operation for the application.

- **no**

  The resources required for execution by the component instance are fetched from the Enterprise Server to the Peer Server every time the CRC operation is performed.

  Note: If there are changes done to the component resources, this property should be set to **no** before performing the CRC operation and can be reset to **yes** later.

Version Locked

- **yes**

  The component instance is launched with component version provided for property **Version**.

- **no**

  The component instance is launched with highest available component version at all times.

Configuration

The serialized configuration of the component instance. Any changes made to this are reflected when the component instance is launched or the CPS of the component instance is opened.

Note: It is recommended that the value for this property should not be changed manually.

10.1.3 Execution

This group contains properties that effect launch behavior of the component.

Launch Type

This property specifies how the component instance is launched. There are four possible values.

- **Separate Process**

  The component instance is launched in a separate JVM. It is automatically launched when the application is launched. When this option is selected properties **Debug Mode** and **JVM_PARAMS** are visible.

- **In Memory**

  The component instance is launched in the JVM of the Peer Server. It is automatically launched when the application is launched. When this option is selected properties **Debug Mode** and **JVM_PARAMS** are not visible.

- **Manual**

  The component instance is not launched when the application is launched. It has to be launched manually from the command line.

  To launch the component manually -

  1. Right click the component, select **Execution** from the pop-up menu and click **Save Manual Launch Script...** as shown in Figure 6.

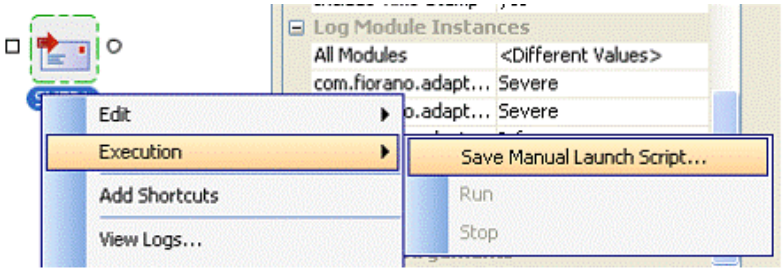


Figure 6: Saving launch script for manual launch

  2. Select the file to save the properties for manual launch and click **Save**.
  3. Follow the instruction mentioned in the window that pops up to launch the component instance.

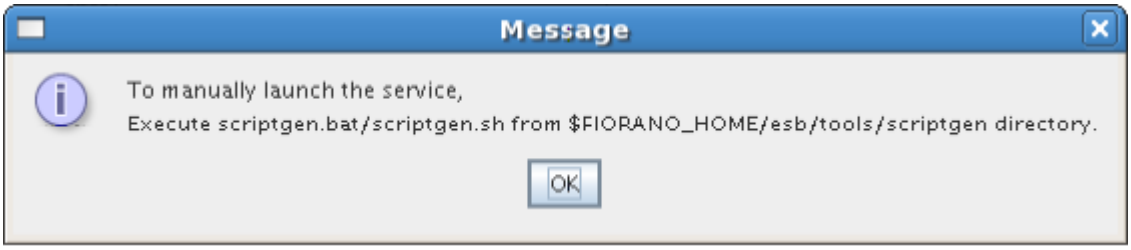


Figure 7: Pop up window showing the steps for launching the component instance manually

4. When this option is selected, properties **Debug Mode** is not visible and the value for property **JVM_PARAMS** is ignored.

- **None**

  If selected, the component instance is never launched. When this option is selected properties **Debug Mode** is not visible and the value for property **JVM_PARAMS** is ignored.

  The boundary of the component is changed to provide a visual clue for the launch type. Figure 8 shows the change in the boundary of component instance for each launch type

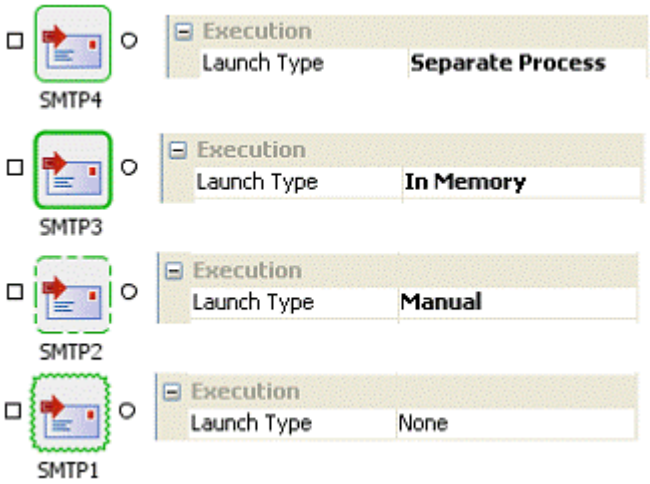


Figure 8: Boundary of component instance with visual clue for launch type

Debug Mode

- **yes**

  The component instance is launched in debug mode. A debugger from any IDE can be attached to the component instance to debug the component instance step by step at runtime. When this value is selected, the property **Debug Port** is visible.

- **no**

  The component instance is not launched in debug mode and a debugger cannot be attached. When this value is selected, the property **Debug Port** is visible.

This property is only used when the property **Launch Type** is set to **Separate Process**. To debug the component instance launched in the memory of Peer Server JVM, debug parameters have to be in the configuration file – %FIORANO_HOME%\esb\fps\bin\fps.conf – of the Peer Server.

Debug Port

The port on which the component waits for instructions from debugger.

When property Debug Mode is set to **yes**, then the property **Debug Port** is set to 5000. It is equivalent to launching the component with the following command line arguments.

-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5000

Note: The component instance can be alternatively launched in debug mode by specifying the required command line arguments in the property **JVM_PARAMS**.

10.1.4 Log Module Instances

This group contains different loggers that are used and their log level configuration.

All the loggers that are used by the component instance are shown as properties in this section. The level at which logging should be performed for each of the logger is defined. There are nine logging level available in the drop-down list – Off, Severe, Warning, Config, Info, Fine, Finer, Finest, and All.

For information about log levels and the effect they have on logs generated, please refer to http://java.sun.com/j2se/1.4.2/docs/api/java/util/logging/Level.html.

The levels of all the loggers can be changed together by selecting required log level for the property **All Modules.**

10.1.5 Runtime Arguments

This group contains a single property **JVM_PARAMS** which contains all command line parameters that are passed to launch the component. This property is used only when the **Launch Type** of the component instance is set to **Separate Process**. Any system properties can be set using this property.

Example: To support regional language specific characters during HTTP transfer, character set encoding of the regional language has to be specified using the system property **file.encoding**. The system property can be provided as **-Dfile.encoding=<required_encoding>** for this property. If there are some other properties already defined, then the value to add should be appended to the existing value separated by a space.

10.2 Port Properties

10.2.1 Input Port Properties

Input port properties appear in the **Properties** pane when an input port of a component instance is selected. Input port properties for a SMTP component are shown in Figure 9.

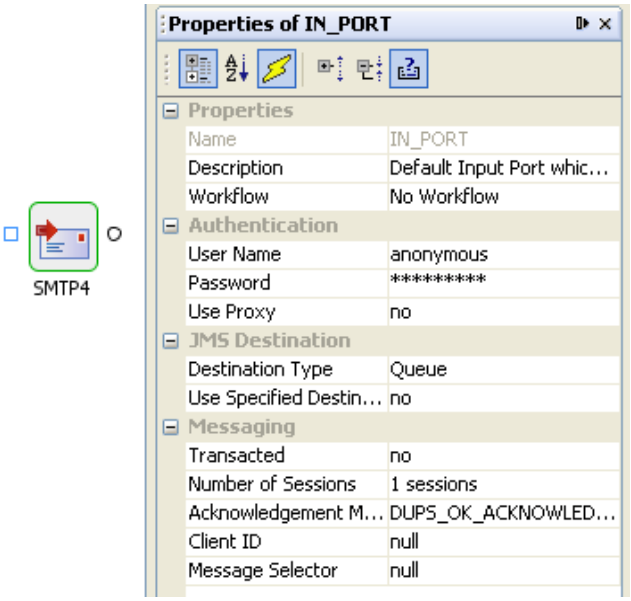


Figure 9: Input port properties for a SMTP component instance

The properties in the **Authentication** group are not used by pre-built components provided by Fiorano. However, they can be used in custom components.

10.2.2 JMS Destination

This group contains properties related to destinations created for the ports of the component.

Destination Type

Specifies whether the destination for the port is a Queue or a Topic. Visual representation of port is changed based on the type of the destination as shown in Figure 10. A square indicates a queue and a circle indicates a topic.

Figure 10: Input port as queue and as topic

- When this property is set to **Topic**, the property **Durable Subscription** is visible.
- When this property is set to **Queue**, the property **Durable Subscription** is not visible.

Use Specified Destination

Destinations for each port in the application are created automatically and associated with a port when the application is launched. The destination name for destination created and associated to a port is constructed using the following function – <Event Process GUID>__<Component instance name>__<Port name>

Example: If an Event Process whose GUID is SAMPLE_APP contains a SMTP component instance with name MAIL_SENDER. The destination created for the input port is SAMPLE_APP__MAIL_SENDER__IN_PORT.

However, if the destination to be used has to explicitly defined, then this property should be set to **yes**.

- When this property is set to **yes**, the property **Destination Name** is visible.
- When this property is set to **no**, the property **Destination Name** is not visible.

Destination Name

The name of topic or queue that has to be associated with the port. The destination is automatically created if it is not present.

10.2.2.1 Destination Encryption

Destination Encryption allows storage of sensitive data from different publishers in a secure way. Encryption implies the transformation of plain text into cipher text by using "key" A key is also used to decrypt the cipher text into plain text.

Destination Encryption Characteristics

In Destination Encryption, all messages sent to a particular destination (topic or queue) are encrypted, thus providing a secure channel of delivery. Component doesn't have to encrypt the message explicitly like in Message Encryption. A destination is marked as encrypted at the time of its creation. All messages published on this destination are encrypted before storing those in database and delivered decrypted to subscribing applications. A client application, therefore, does not have to explicitly decrypt a received message.

Base Implementation

FioranoSOA versions 9.5.0 and upwards supports Destination Encryption. DES (Data Encryption Standards) is used as the default encryption algorithm. In addition to that FioranoSOA supports Base64, TripleDES, AES256, AES128, AES192, RC2-40, RC2-64, and RC2-128

The Destination Encryption function uses the library cryptix.jar provided by Cryptix for generating keys as well as for encryption. This file comes bundled with the FioranoSOA installation. It can be found in the FIORANO_HOME/extlib/cryptix directory of the FioranoSOA installation.

Destination Encryption involves only encrypting the payload of the message and not its JMS headers and properties.

Encrypted Destination Creation

Input/Output port properties appear in the Properties pane when an input port of a component instance is selected. In properties pane, under the JMSDestination tab, for a DISPLAY component are shown in Figure 1.

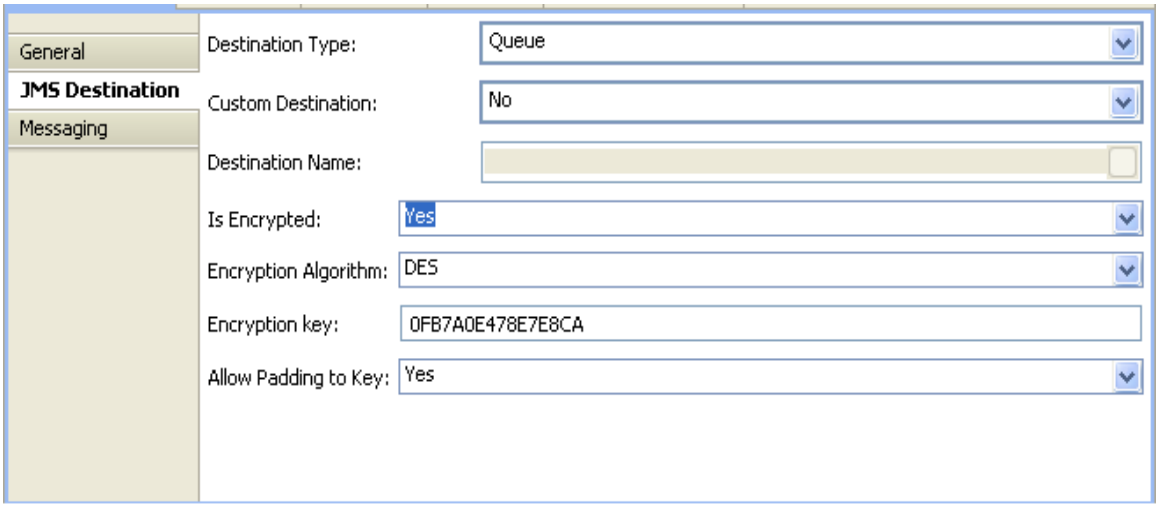


Figure 1 – Port Encryption Configuration

- **Is Encrypted:**
  - **Yes –** Encrypted JMS destination will be created.
  - **No –** Non Encrypted JMS destination will be created.
- **Encryption Algorithm:** The supported algorithms are DES, Base64, TripleDES, AES256, AES128, AES192, RC2-40, RC2-64, and RC2-128.
- **Encryption Key:** Specify key which is used to encrypt the incoming data. Specify 64 digit key for AES256, 48 digit key for AES192, 32 digit key for AES128, 16 digit key for DES, 48 digit key for TripleDES, 10 digit key for RC2_40, 16 digit key for RC2_64 and 32 digit key for RC2_128.
- **Allow Padding to Key**: Choose **yes** to allow padding to the key to make exactly the value required for the mentioned algorithm.

10.2.3 Messaging

This group contains properties related to JMS messaging concepts. In general, pre-built components provided by Fiorano use a single connection and share same session for reading messages on input port and sending messages on the output port.

Transacted

Specifies whether the JMS session is transacted or not.

- **yes** - JMS session is created as a transacted session. Multiple input messages can be grouped into a single transaction. Messages are not sent on the output port of the component until the transaction is complete. They are held in-memory of the component. When this value is selected, property **Transaction Size** is visible.
- **no** - JMS session is created as a non-transacted session. Messages are sent on the output port of the component immediately. When this value is selected, property **Transaction Size** is visible.

  Note: When the output of component is very large, it is not advised to set this property value to **yes**.

  A transaction is based on the number of input messages processed and not based on the number of output messages sent. In cases, where a component sends a large number of output messages for each input request, it is not advised to set this property value to **yes** even if value for property **Transaction Size** is set to 1.

  Example: A File Reader component reading a large binary file or a DB component that returns result set containing a large number of rows.

Transaction Size

Number of input messages that should be processed before committing the transaction.

Number of Sessions

Number of sessions that are created by the component instance to process messages received on the input port. Messages are processed in a separate session by each thread. This property is used to increase the number of threads that can process the requests and thereby increase the through put. However, the number of threads that can at a time is restricted by **Max Pool Size** property in **Connection Pooling Configuration** panel.

Acknowledgement Mode

Specifies the acknowledgement mode that is used to acknowledge messages. Messages are not deleted from destinations until they are acknowledged. When the component fails over because of the HA fail over of Peer Server, all the messages that are not acknowledged are redelivered. The number of duplicate messages in case of fail over can be controlled using this property.

DUPS_OK_ACKNOWLEDGE

Messages are acknowledged after configured number of messages are successfully processed and output messages are sent by the component instance. The number of messages after which the messages are acknowledged can be configured at the following node in FMQ profile – Fiorano > mq > pubsub > TopicSubSystem > DupsOkBatchSize. Number of duplicate messages is utmost the DupsOkBatchSize for each session on each input port of the component instance.

AUTO_ACKNOWLEDGE

Messages are acknowledged after a message is successfully processed and the output message is sent by the component instance. Hence, number of duplicate messages is utmost 1 for each session on each input port of the component instance.

CLIENT_ACKNOWLEDGE

This property is ignored if the property **Transacted** is set to **yes**. When the property **Transacted** is set to **yes**, messages are acknowledged when the transaction is committed. Hence, number of duplicate messages is utmost the number denoted by property **Transaction Size** for each session on each input port of the component instance.

Durable Subscription

Specifies whether a durable subscriber is created on the destination represented by the input port or not. This property is visible only when value for the property **Destination Type** is set to **Topic**.

- **yes -** A durable subscription is created to the topic represented by the input port. Messages sent to the topic are not lost when the component fails over because of the HA fail over of Peer Server. If selected, then the property **Subscription Name** is visible.
- **no -** A durable subscription is created to the topic represented by the input port. Messages sent to the topic may be lost when the component fails over because of the HA fail over of Peer Server.

Note: If the Destination Type is Queue, then messages are not lost when the component fails over because of the HA fail over of Peer Server.

Following components do not create durable subscriptions and hence have to use only queues as input ports – WSStub and HTTPStub.

Subscription Name

Subscription name that should used be when creating a durable subscriber. This should be a unique name for the Peer Server on which the component instance is running.

Client ID

Unique ID for the JMS connection created by this port. This property is not used by pre-built components provided by Fiorano as there is only one connection created for each component instance.

Message Selector

Specifies a condition (JMS Message Selector) to select only a particular set of messages.

10.3 Output Port Properties

Output port properties appear in the **Properties** pane when an output of a component instance is selected. Output port properties for a SMTP component are shown in Figure 11.

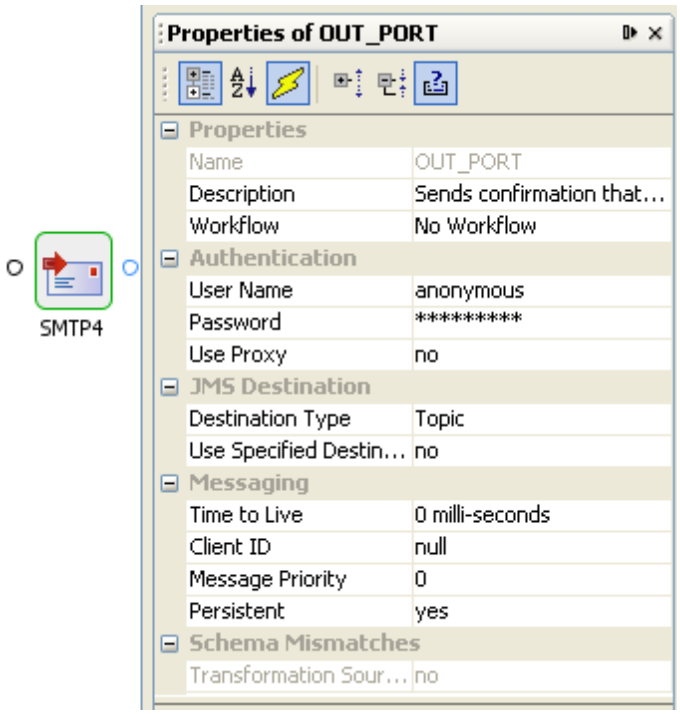


Figure 11: Input port as queue and as topic

The properties in the **Authentication** group are not used by pre-built components provided by Fiorano. However, they can be used in custom components.

10.3.1 JMS Destination

This group contains properties related to destinations created for the ports of the component. For more information, refer to section **JMS Destination** under **Input Port Properties**.

10.3.2 Messaging

Time to Live

Specifies the time in milliseconds for which the message is available from the system after it is sent from the producer. The message is discarded on the expiration of this time. 0 (zero) milliseconds indicate infinite life time.

Example: Consider an application contains four component instances. Value for this property is set to 30000 milliseconds on the output port of the first component instance. This means, once the message is sent on the output port of first component instance, the message should be consumed by the fourth component instance in 30 seconds. If it is not consumed in 30 seconds, then the message will be lost and will never reach the fourth component.

Client ID

Unique ID for the JMS connection is created by this port. This property is not used by pre-built components provided by Fiorano as there is only one connection created for each component instance.

Message Priority

The priority of the message sent, can be any integer value from 0 (zero - lowest) to 9 (nine - highest). The default priority is 4.

Persistent

Specifies whether the producer on this port sends persistent messages or not. If this property is set to **yes**, all messages are persisted to message store. If this property is set to **no**, the messages are not persisted and there may be message loss noticed during the HA fail over of the Peer Server.

10.3.3 Preventing message loss

To avoid message loss in Fiorano Event Processes, change the following properties -

f) On each output port, set value of property **Persistent** to **yes**.

g) On each route, set value of property **Durable** to **yes**.

h) On each input port that is a topic, set value of property **Durable Subscription** to **yes**.

i) Whenever a JMS* component is used, make sure all messages sent are persistent and all subscriptions to topics are durable. These properties can be set in CPS of JMS components.

10.3.4 Components with implicitly defined JMS messaging properties

Some of the less used components in Fiorano have Acknowledgement Mode, Transaction Type and Transaction Size implicitly defined and will not pick the values from input port.

- Aggregator AUTO_ACKNOWLEDGE, true, 1
- Chat AUTO_ACKNOWLEDGE, false, 1
- DiskUsageMonitorService AUTO_ACKNOWLEDGE, true, 1
- Display AUTO_ACKNOWLEDGE, false, 1
- ExceptionListener AUTO_ACKNOWLEDGE, false, 1
- Feeder AUTO_ACKNOWLEDGE, false, 1
- HTTPReceive AUTO_ACKNOWLEDGE, false, 1
- HTTPStub DUPS_OK_ACKNOWLEDGE, false, 1
- Join AUTO_ACKNOWLEDGE, from command line, 1
- SAPR3Monitor AUTO_ACKNOWLEDGE, false, 1
- Sleep AUTO_ACKNOWLEDGE, false, 1

- Timer AUTO_ACKNOWLEDGE, true, 1
- WSStub DUPS_OK_ACKNOWLEDGE, false, 1
- XMLVerification AUTO_ACKNOWLEDGE, true, 1

10.4 Managed Connection Factory

Use Connection details from input

The parameters to create the connection can be specified in the input message when this property is selected. If this property is selected the validation errors in the managed connection factory panel of the CPS are treated as warnings. So user can bypass this step without giving valid configuration and complete the configuration of the component. If valid properties are not provided even in the input message exception will be thrown at runtime.

Note:

- If this option is chosen, An additional element ConnectionFactorySettings is added to the schema of the input port of component. Properties that are used to create the connection for a specific component are present under this element.
- For the components EJBAdapter and WebServiceConsumer, the user cannot proceed with the configuration of the component by giving invalid configuration, even if this option is chosen.

Connection Pool Params

Defines the connection pool settings for the component. Creating a connection to external systems like Database or FTP Server or HTTP Server is typically a resource extensive and time consuming process. Configuring a connection pool reduces the overhead of creating a connection on each request.

Click the ellipses button to launch an editor to configure connection pool parameters as shown in Figure 12

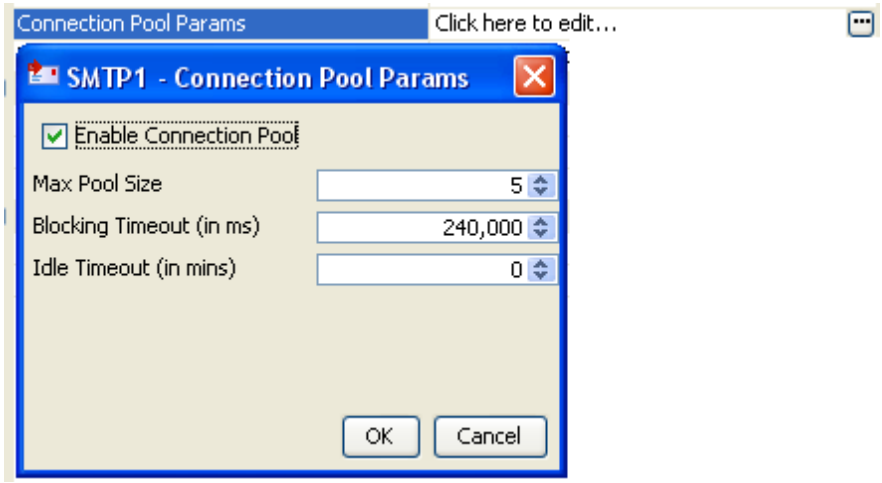


Figure 12: Connection pool configurations

- **Enable Connection Pool**

If this property is selected, the connections created are cached in a pool for subsequent use. When the connection pool is disabled it implies that the connection should not be cached and a new connection will be created for each request.

Enabling connection pool property will reduce the time spent in creating a new connection for every input request.

Properties **Max Pool Size**, **Blocking Timeout** and **Idle Timeout** are enabled only when this property is selected.

- **Max** Pool **Size**

  The maximum number of connections that can be cached in the pool.

- **Blocking Timeout (in ms)**

  The time in milliseconds after which the call to fetch a connection from the pool will timeout, if there is no unused connection available. Connection will not be created after timeout.

- **Idle Timeout (in mins)**

  Specifies the maximum number of minutes that an idle (unallocated) connection can remain in the pool before being removed to free resources.

Proxy Settings

Click the ellipses button to launch an editor to configure proxy configurations as shown in Figure 13.

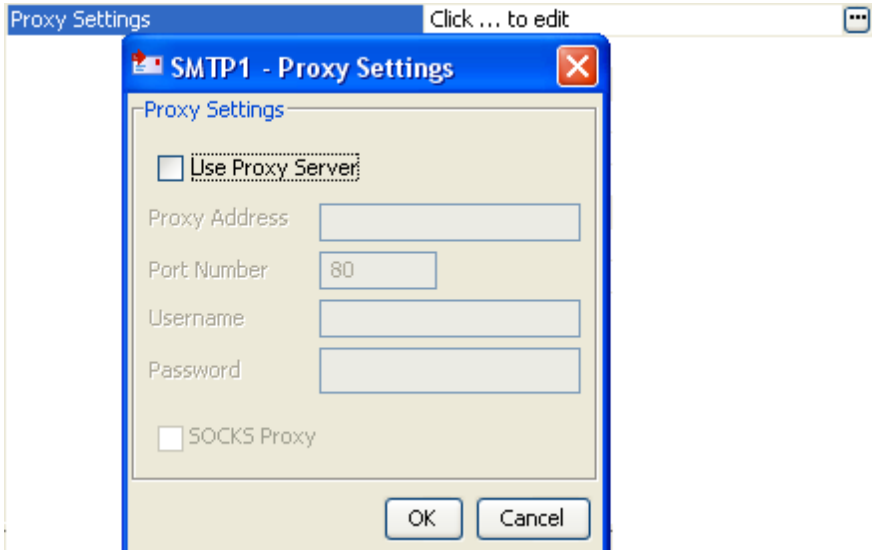


Figure 13: Proxy configurations

Use Proxy Server

Select this option if the connection has to be established using a proxy server. Properties **Proxy Address**, **Port Number**, **Username**, **Password** and **SOCKS Proxy**.

- **Proxy Address**

  The IP address or the host name of the machine where the proxy server is running.

- **Port Number**

  Port number on which the proxy server is running.

- **Username**

  The user name with which the user can login to the proxy server.

- **Password**

  Password for the user name provided.

- **SOCKS Proxy**

  Enable this property to use SOCKS protocol to connect to the proxy server.

Note: When the property **Use Connection details from input** is chosen, an element **ProxySettings** will be added to the schema of the input port of the component as shown in figure below to provide the proxy details in the input message.

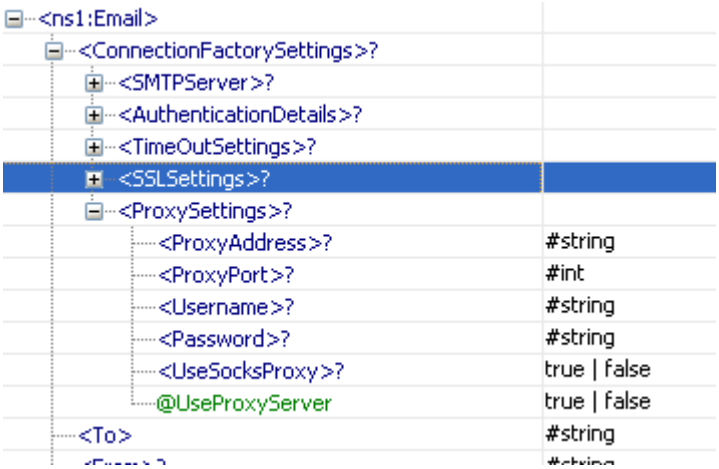


Figure: Schema changes for proxy settings when Use Connection details from input is set

10.4.1 SSL Security

Click the ellipses button to launch an editor to configure SSL configurations as shown in Figure 14.

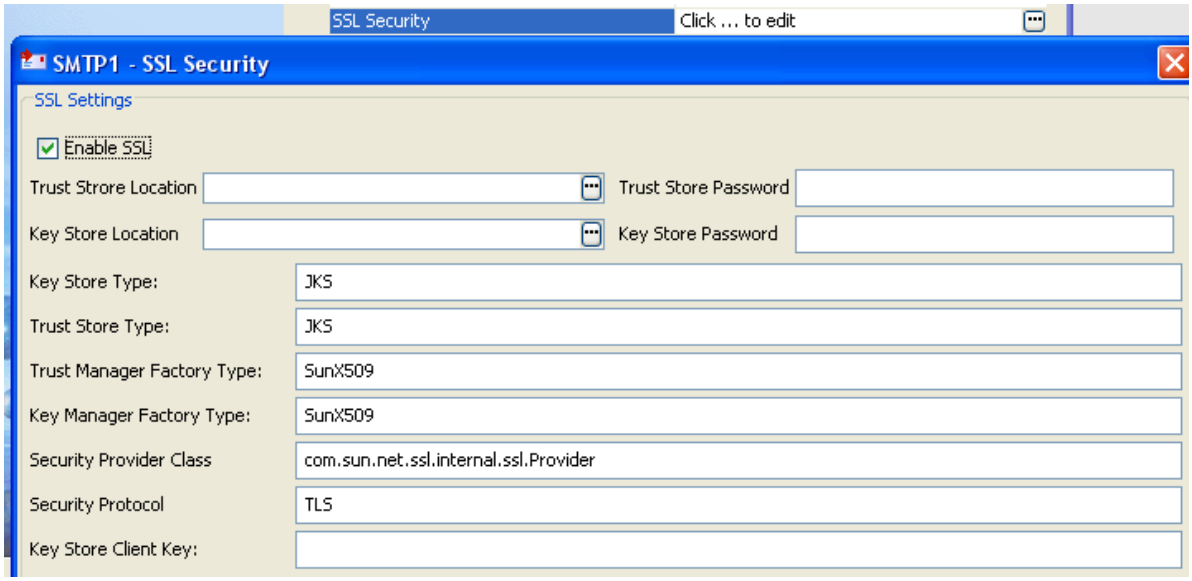


Figure 14: SSL configurations

- **Enable SSL**

  Select this option to enable SSL Settings. Rest of the properties in this editor are enabled and configurable only when this property is checked.

- **Accept Server Certificate**

  Select this option to accept server certificate without validating the certificate.

- **Ignore Hostname Mismatch**

  Select this option to accept server certificate even if hostname in the certificate does not match with the hostname in the request URL.

  Note: Accept Server Certificate and Ignore Hostname Mismatch properties are useful only in consumer type components like HTTPAdapters, WebServiceConsumer etc.

- **Trust Store Location**

  Location of the trust store file. TrustStore is a file where digital certificates of trusted sites are stored and retrieved for authentication during an SSL connection. TrustStore is used to authenticate a server in SSL authentication.

- **Trust Store Password**

  Password of the specified trust store.

- **Key Store Location**

Location of the key store file. The KeyStore is used by the component for client authentication.

- **Key Store Password**

  Password of the specified key store.

- **Key Store Type**

  Type of the Key Store whose location is specified by **Key Store Location.**

- **Trust Store Type**

  Type of Trust Store whose location is specified by property **Trust Store Location.**

- **Trust Manager Factory Type**

  Algorithm for the Trust Manager Factory.

- **Key Manager Factory Type**

  Algorithm for the Key Manager Factory.

- **Security Provider Class**

  Determines Security provider class.

- **Security Protocol**

  Determines Security protocol

- **Key Store Client Key**

  Determines Key Store Client Key

Note:

- For more information on SSL Configurations, please refer to Section 3.11 – Public Key, Cryptographic Keystore and Truststore.
- When the property Use Connection details from input is chosen, an element SSL Settings will be added to the schema of the input port of the component as shown in figure 16 to provide the SSL details in the input message.

<ns1:Email>	
<ConnectionFactorySettings>?	
<SMTPServer>?	
<AuthenticationDetails>?	
<TimeOutSettings>?	
<SSLSettings>?	
<TrustStoreLocation>?	#string
<TrustStorePassword>?	#string
<KeyStoreLocation>?	#string
<KeyStorePassword>?	#string
<KeyStoreType>?	#string
<TrustStoreType>?	#string
<TrustManagerFactoryType>?	#string
<KeyManagerFactoryType>?	#string
<SecurityProviderClass>?	#string
<SecurityProptocol>?	#string
<KeyStoreClientKey>?	#string
@EnableSSL	true | false
<ProxySettings>?	

Figure: Schema changes for SSL settings when Use Connection details from input is set

10.5 Interaction Configurations

Validate Input	no
Cleanup resources (excluding connectio...	yes
Target Namespace	http://www.fiorano.com/fesb/activity/SMTP1
Monitoring configuration	disabled: 5000 milli seconds

Figure 15: Common configurations in Interaction Spec panel

Validate Input

- **no -** The input is not validated and is directly processed by the component. Exception may occur while processing if the message is invalid.
- **yes -** The input is validated against the structure specified on the input port of the component. If it is not valid, exception occurs and further processing is not done.

Cleanup resources (excluding connection) after each document

A component creates various objects to process business logic. Some of these objects are connection objects or are related to connection where as other objects are not related to connection but are required to process business logic. Holding these objects in-memory all the time will make lesser memory available that can be freed and deleting these objects to free up space results in higher processing time as the objects have to be recreated. Hence, the objects related to business logic can be removed from time to time.

- **no -** When this option is selected, objects that are not connection related are destroyed and recreated for each request.

- **yes -** When this option is selected, objects that are not connection related are not destroyed and are reused for each request.

Note: When a connection object is destroyed all objects are recreated on subsequent request.

Target Namespace

Two or more XML schema having same namespace will cause problems if there are elements which are defined with same name. Schema set on the input and output ports of the component are in some created by the component. To avoid the clash of elements from different schema, the schema generated by the component use the value provided for this property to compute the namespace for input or output schema.

Monitoring Configuration

When monitoring is enabled for a component the following statistics are sent to **FPS_USER_EVENTS_TOPIC** at configured intervals of time

- **Minimum execution time**

  The minimum amount of time taken to process any message during the last publish interval.

- **Maximum execution time**

  The maximum amount of time taken to process any message during the last publish interval.

- **Count**

  Number of messages processed during the last publish interval.

- **Throughput**

  Rate at which messages are processed during the last publish interval.

Click the ellipses button to launch an editor to configure monitoring configurations as shown in Figure 16.

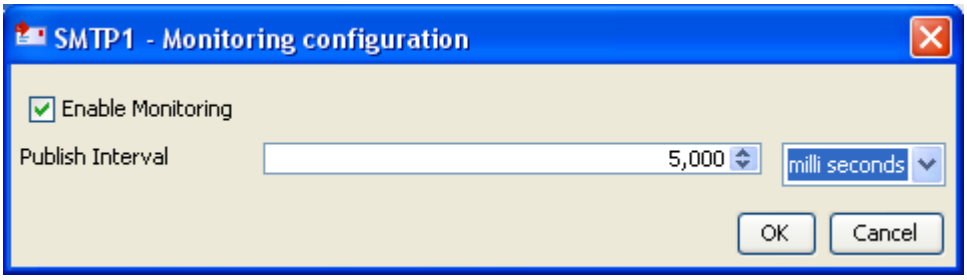


Figure 16: Monitoring configurations in Interaction Spec panel

- **Enable Monitoring**

  Select the check box to enable monitoring for request execution time.

- **Publish Interval**

The time interval after which monitoring statistics are computed and sent.

To view the monitoring statistics in dashboard, open FES profile and navigate to **FES > Fiorano > Esb > Events > FESEventsManager**. Click on **FESEventsManager** and set the property **ListenForUserEvents** to **yes** in the properties window.

10.5.1 Scheduler Configurations

A component can be scheduled to execute fixed request at configured intervals of time. When the component is configured to run in scheduler mode the component will not have input port. However, messaging properties that are usually configured on the input port can be configured in **Transport Configurations** panel.

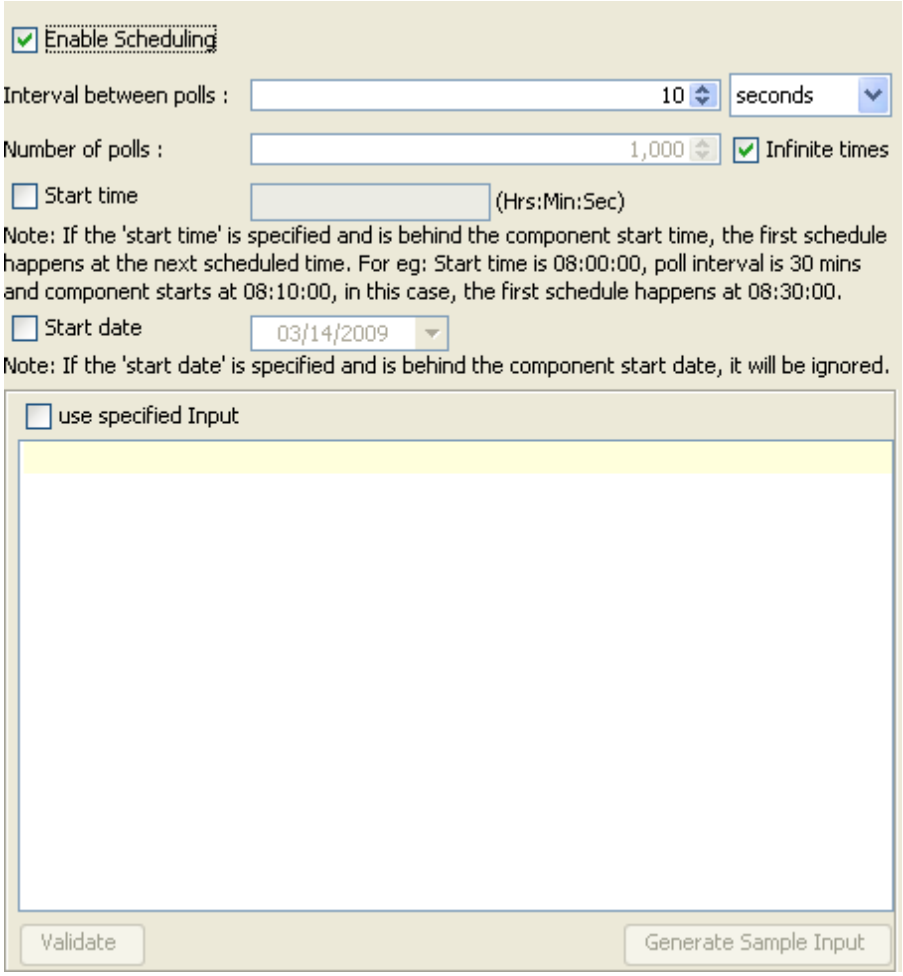


Figure 17: Scheduler configurations panel

- **Enable Scheduling**: Select the check box to run the component in the scheduling mode.
- **Interval between polls**: Specifies time interval between successive requests.
- **Number of polls**: Number of times the input request is executed.

- **Infinite times**: If this option is enabled, then the number of times the input request is executed will be infinite.
- **Start time**: The polling start time. If the specified start time is earlier then the component start time, the first schedule will happen at the next scheduled time. For example, start time is 08:00:00, poll interval is 30 minutes, and component starts at 8:10:00, the first schedule will happen at 08:30:00.
- **Start date**: The polling start date. If the specified start date is earlier then the component start date, then it will be ignored and input messages are sent at next scheduled date.
- **Use specified Inpu**t: Select the check box to configure input that is repeatedly executed.
- **Validate**: Validates the specified input against the structure specified on the input port.
- **Generate Sample Input**: Generates the sample input for the structure specified on the input port.

10.5.2 Advanced Configuration

Advanced scheduling information can be configured in the configuration property sheet of BC components as shown in figure below.

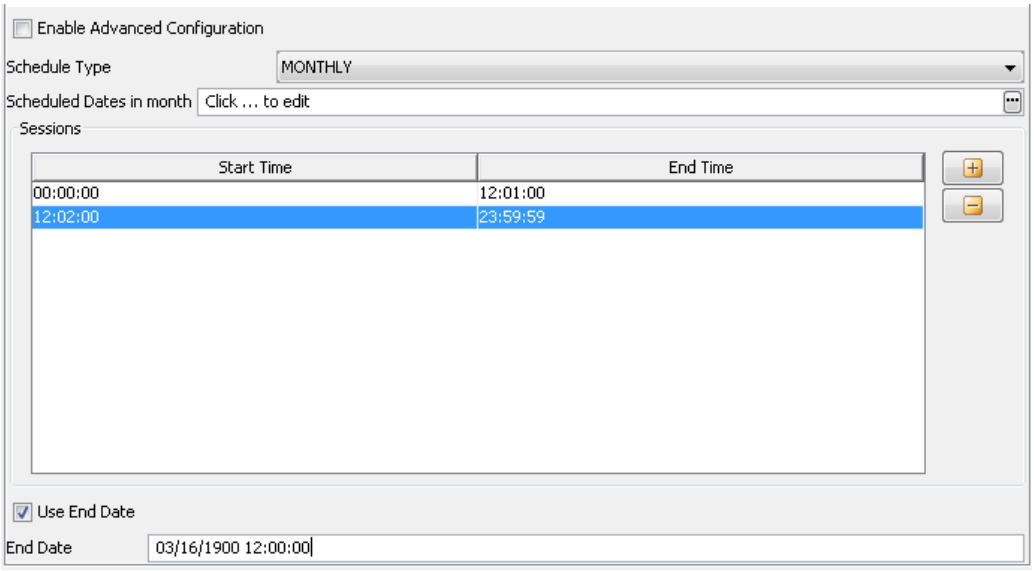


Figure: Advanced Scheduling configuration

Enable Advanced Scheduling

This option can be chosen if advanced scheduling options like scheduling on specific days, dates and during certain time periods needs to be done.

Scheduling Type

The scheduling type can be chosen depending on the way the scheduling of messages needs to be controlled.

The different scheduling types and their working are as below.

DAILY

The message is sent every day between the start and end time as defined in the **sessions** table. The times have to be in hh:mm:ss format.

WEEKLY

The message is sent on pre defined days of a week. The days in week can be chosen on clicking the ellipsis against the property **Scheduled Days in Week.** The messages are sent only during the intervals defined by the **sessions** panel.

MONTHLY

The message is sent on pre defined dates in a month. The days in week can be chosen on clicking the ellipsis against the property **Scheduled Dates in Month.** The messages are sent only during the intervals defined by the **sessions** panel.

Use End Date

When this option is chosen the scheduling will stop on the date as defined against **End Date.** The date has to be in **MM/dd/yyyy HH:mm:ss** format.

Note

When finite repeat count and end date are both used the scheduling will stop based on which event happens earlier.

10.6 Transport Configurations

Transport Configurations panel is used to configure messaging properties when the component is configured in scheduling mode. These properties are configured on the input port when the component is not in scheduling mode.

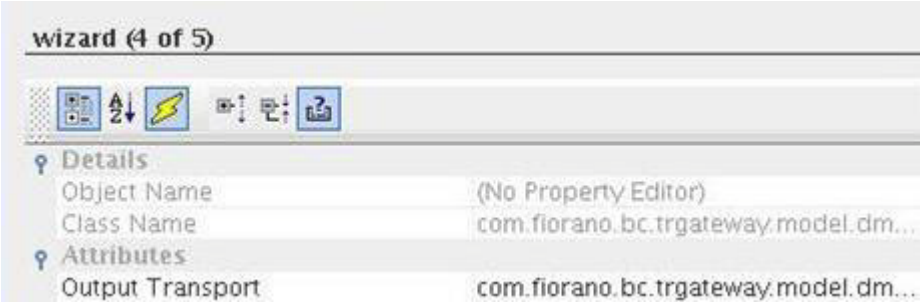


Figure 18: Transport configurations panel

Output Transport

Click the ellipses button to launch an editor for configuring transport properties. The properties in the editor are shown in Figure 19.

Acknowledgement Mode	1
class	com.fiorano.bc.trgateway.model.dmi.tr.JM...
Enable request-reply	no
Transacted	no
Transaction Size	1

Figure 19: Transport properties

- **Acknowledgement Mode**

  Valid values for this field are 1 to indicate AUTO_ACKNOWLEDGE, 2 to indicate CLIENT_ACKNOWLEDGE and 3 to indicate DUPS_OK_ACKNOWLEDGE. Any other value will throw an exception and component will not be started.

  For information on acknowledgement modes, refer to section **Acknowledgement Mode** in **Input Port Properties**.

- **Enable request-reply**

  If this property is set to **yes**, then responses will be sent to JMSReplyTo port if it is present, else response is sent on the output port of the component.

- **Transacted**

  For information on this property, refer to section **Transacted** in **Input Port Properties**.

- **Transaction Size**

  For information on this property, refer to section **Transaction Size** in **Input Port Properties**.

10.7 Error Handling

Errors that occur in the component are classified into five categories – JMS Error, Response Generation Error, Request Processing Error, Connection Error and Invalid Request Error. Actions that have to be taken when an error occurs are defined in the Error Handling panel.

Note: Some of the panels or actions are not available for some components and hence are not visible in those components.

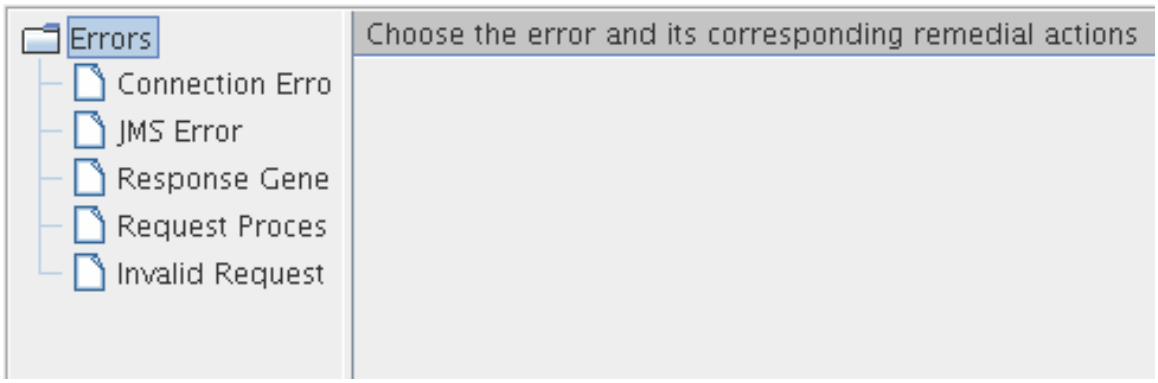


Figure 20: Error handling

10.7.1 Connection Error

This property in Error Handling Panel will be visible only if the Managed Connection Factory panel is present. Presence of Managed Connection Factory implies that the component makes a connection to external system.

Example: Components like FileReader and FileWriter do not create any connections and hence they do not have this property in the CPS.

Errors that occur because of the connection to an external system cannot be made or because the connection to an external system is lost are categorized under the category **Connection Error**.

Example: Trying to connect to an external web site when the network connection is not active.

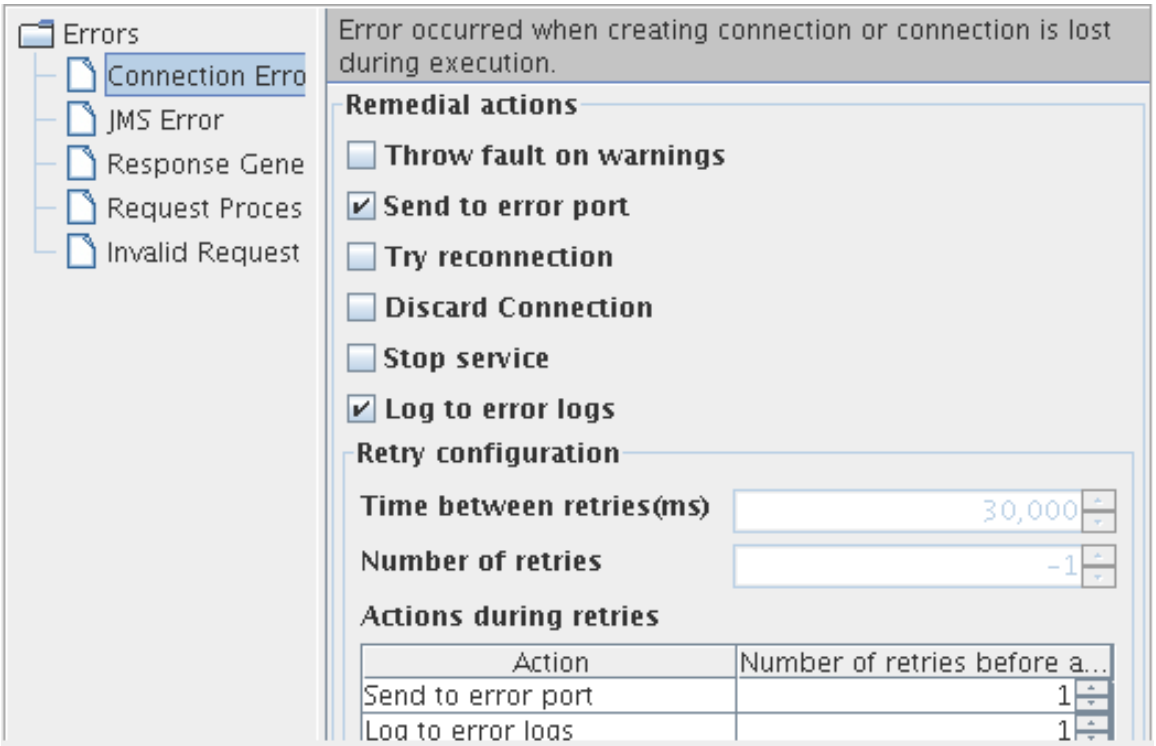


Figure 22: Available actions for Connection Error category

Remedial Actions

Actions that can be taken when a **Connection Error** occurs is shown in Figure 22.

- **Send To Error Port**

  The component will send the error on the ON_EXCEPTION output port of the component if this action is enabled. By default, ON_EXCEPTION port is present in all components that support error handling. If this option is unchecked, then the **Retries before sending error** property in **Advanced Settings** group is disabled.

- **Try reconnection**

  The component will re-execute the input request with a new connection, if this action is enabled. Configuring for retries is explained in **Retry Configuration** section.

- **Discard Connection**

  The component removes the connection from the connection pool as soon as a connection error occurs. Subsequent request uses a new connection from the connection pool.

- **Stop Service**

  The component is stopped when an error occurs if this action is enabled.

10.7.2 JMS Error

Errors that occur in transport (JMS)

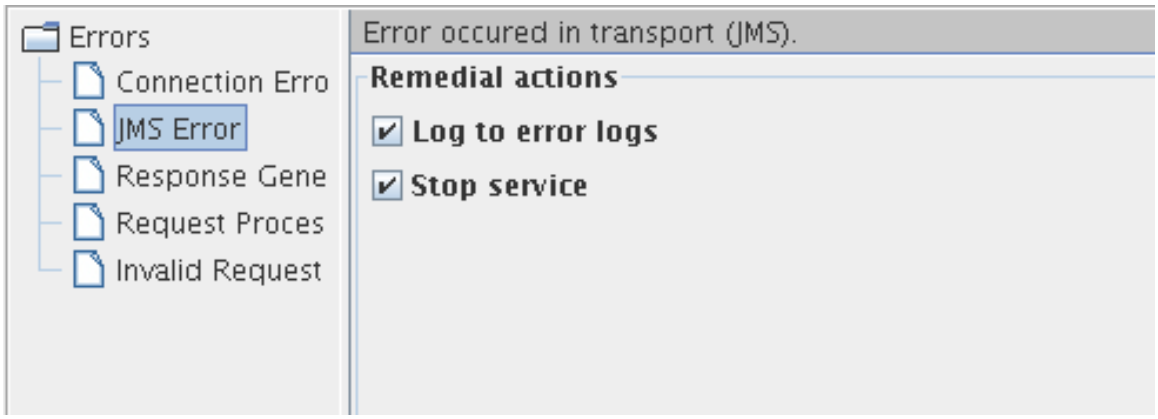


Figure: Available actions for JMS Error category

Remedial Actions

Actions that can be taken when a **JMS Error** occurs is shown in Figure 23.

Log to error logs

Logs the exception and trace to error logs.

Stop service

The component is stopped when an error occurs if this action is enabled.

10.7.3 Response Generation Error

Errors that occur while building a response for the processed request.

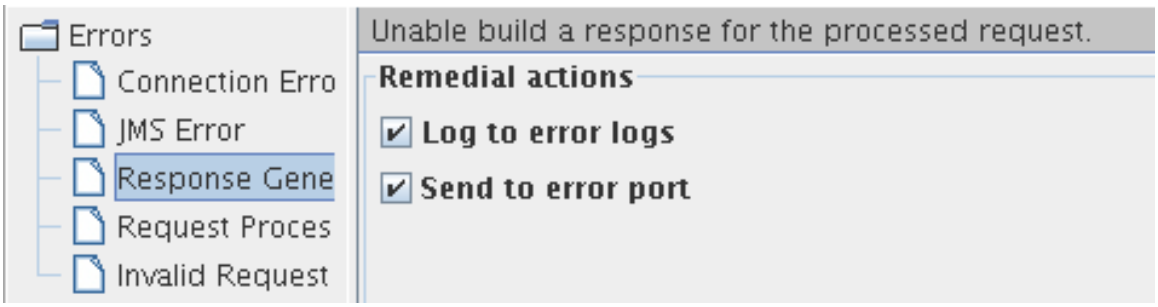


Figure: Available actions for Response Generation Error category

Remedial Actions

Actions that can be taken when a **Response Generation Error** occurs is shown in Figure 24.

Log to error logs

Logs the exception and trace to error logs.

Send to error port

The component will send the error on the ON_EXCEPTION output port of the component if this action is enabled. By default, ON_EXCEPTION port is present in all components that support error handling. **Retries before sending error** property in **Advanced Settings** group is disabled if this option is unchecked.

10.7.4 Request Processing Error

Request Processing Errors are categorized based on the following conditions -

- The error occurs after input message is successfully parsed and understood.
- The error is not a result of connection problems.

Example: In case of FTPGet, a Request Processing Error occurs when the specified file (to be downloaded) in the input request is not present in FTP Server.

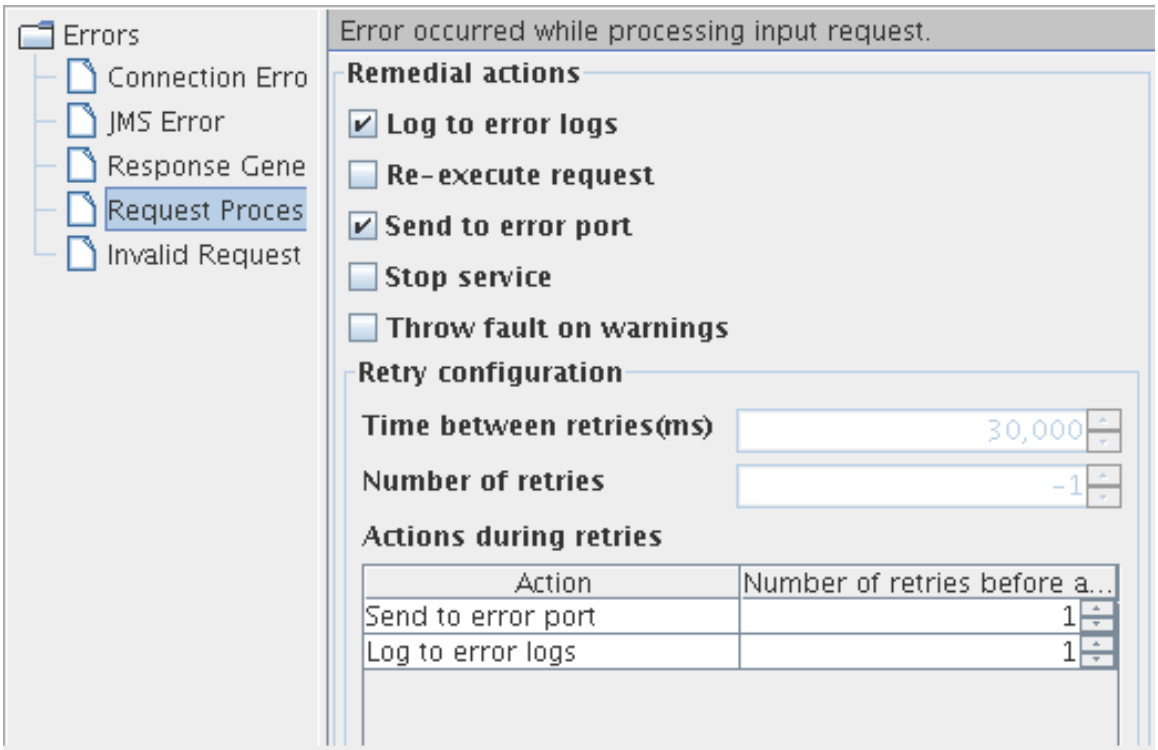


Figure 21: Available actions for Request Processing Error category

Remedial Actions

Actions that can be taken when a **Request Processing Error** occurs is shown in Figure 21.

- **Log to error logs**

  Logs the exception and trace to error logs.

- **Re-execute Request**

  The component will re-execute the input request if this action is enabled. Configuring for retries is explained in section **Retry Configuration** section. This action should be enabled only for errors that may be rectified over time.

  Example: Error in file reader because a file is not found. If the required file should be placed by another process, then the file not found error can be rectified over time and hence can be retried.

- **Send To Error Port**

The component will send the error on the ON_EXCEPTION output port of the component if this action is enabled. By default, ON_EXCEPTION port is present in all components that support error handling. **Retries before sending error** property in **Advanced Settings** group is disabled if this option is unchecked.

- **Stop Service**

  The component is stopped when an error occurs if this action is enabled.

- **Throw fault on warnings**

  In some cases, a problem in the component which is not severe is treated as a warning. Such warnings are just logged by default. The component will treat such warnings as errors, if this property is enabled.

  Example: When the FileReader is configured to read files with a particular pattern for file names, a warning is logged if there are no files whose names match the pattern configured. If the FileReader is polling a directory, then it is an inherent assumption that files are not always present and hence treating it as warning is appropriate. But if the file reader is not in scheduler mode, then absence of files has to be treated as an error.

10.7.5 Invalid Request Error

Errors that occur when parsing the input request are categorized under Invalid Request Error. Remedial actions are different for EDBC and BC components.

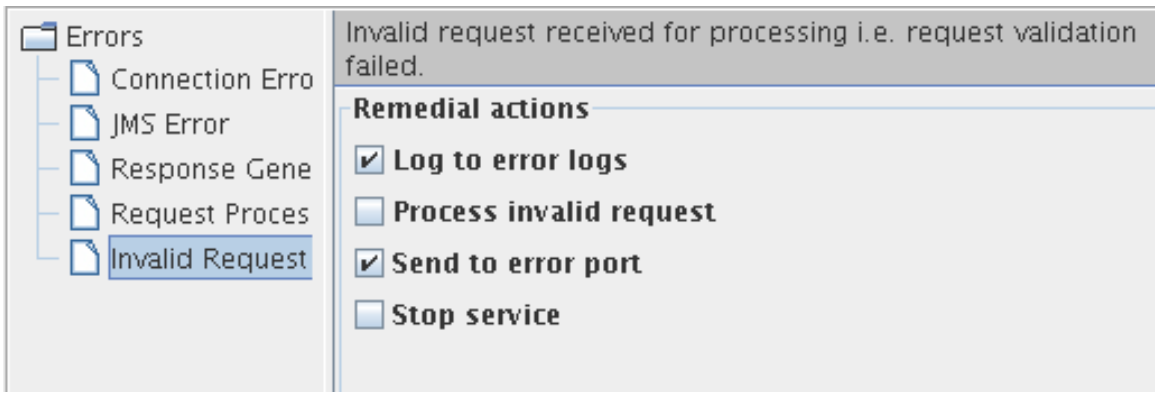


Figure 23: Available actions for Invalid Request Error category

EDBC Components

Remedial Actions

- **Log to error logs**

  Logs the exception and trace to error logs.

- **Process invalid request**

  Do not stop processing in case request is invalid. Continue Processing.

- **Send To Error Port**

  The component will send the error on the ON_EXCEPTION output port of the component if this action is enabled. By default, ON_EXCEPTION port is present in all components that support error handling. If this option is unchecked, then the **Retries before sending error** property in **Advanced Settings** group is disabled.

- **Stop service**

  The component is stopped when an error occurs if this action is enabled.

BC Components

Remedial Actions

Actions that can be taken when an **Invalid Request Error** occurs.

- **Send To Error Port**

  The component will send the error on the ON_EXCEPTION output port of the component if this action is enabled. By default, ON_EXCEPTION port is present in all components that support error handling. **Retries before sending error** property in **Advanced Settings** group is disabled if this option is unchecked.

- **Do not stop service**

  If this property is **not** checked, when an invalid input is sent to the component, the component will be stopped immediately. This property is checked by default.

  Example: In case of SMTP, if the input message is not valid according to the schema set on its IN_PORT, an exception occurs and the component will be stopped only if this property is unchecked.

10.7.6 Retry Configuration

When **Re-execute Request** is enabled for **Request Processing Error** or when **Try Reconnection** is enabled for **Connection Error**, the **Advanced Settings** group containing configurations for retries is visible.

Time between retries(ms)

The time interval between two successive retries.

Number of retries

The number of times the component should retry the request. This property is enabled only if **Infinite** check box is unselected.

Infinite

If the check box is selected, the component will continuously retry the request until the request is process successfully. when this option is selected, the property **Number of retries** is disabled and its value is ignored.

Retries before sending error

This property is enabled only if **Send To Error Port** action is enabled. If **Send To Error Port** action is enabled and if the value for this component is a number **n**, then the component sends an error on the ON_EXCEPTION port after every **n** retries.

10.8 Schema Editor

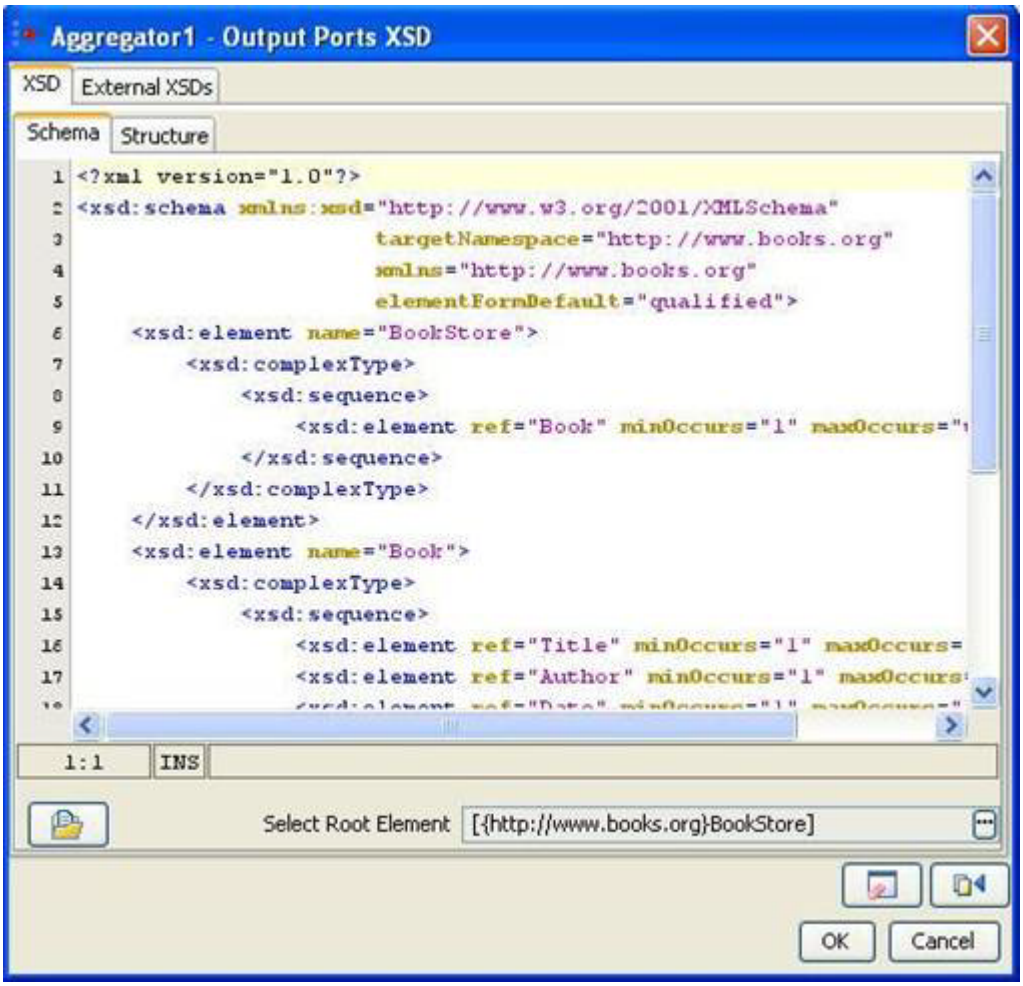


Figure 24: Schema Editor

Schema Editor is used to configure schemas that are required for the functionality of a component.

In general,

- XSDs and DTDs can both be provided in the schema editor. Some components allow only XSDs.

- Only one root element can be selected. Some components allow selecting multiple root elements.
- When a DTD is provided in the schema editor, the External XSDs tab is disabled.

XSD – Schema tab

XSD/DTD can be provided in the text area in the **schema** tab shown in figure 24. Schemas that are present on the file system can be loaded by clicking on **Load** button . This opens a file browser which enables navigation to the required schema on the file system. The file type can be chosen as XSD or DTD in the filechooser.

Root element can be selected by clicking on **Select Root Element** button . A list containing all the elements present in the schema will be displayed as shown in Figure 25. A root element (multiple root elements, in some cases) should be selected from that list of elements. The selected root element(s) will be displayed in the schema editor next to **Select Root Element** text.

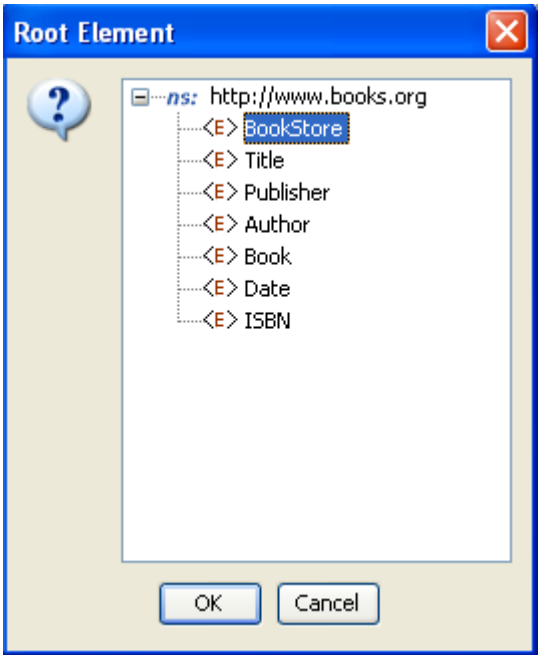


Figure 25: Selection of root element

XSD – Structure tab

The structure tab displays a tree structure of the schema provided as shown in figure 26. The structure depends on root element.

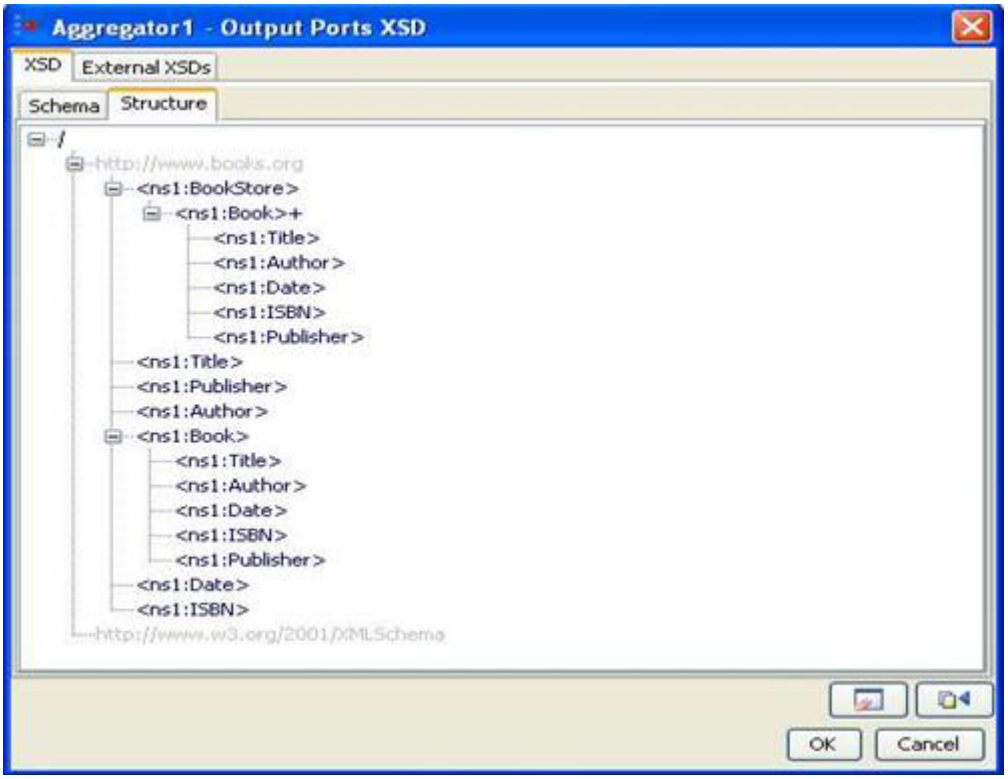


Figure 26: Structure of the schema when no Root Element is chosen

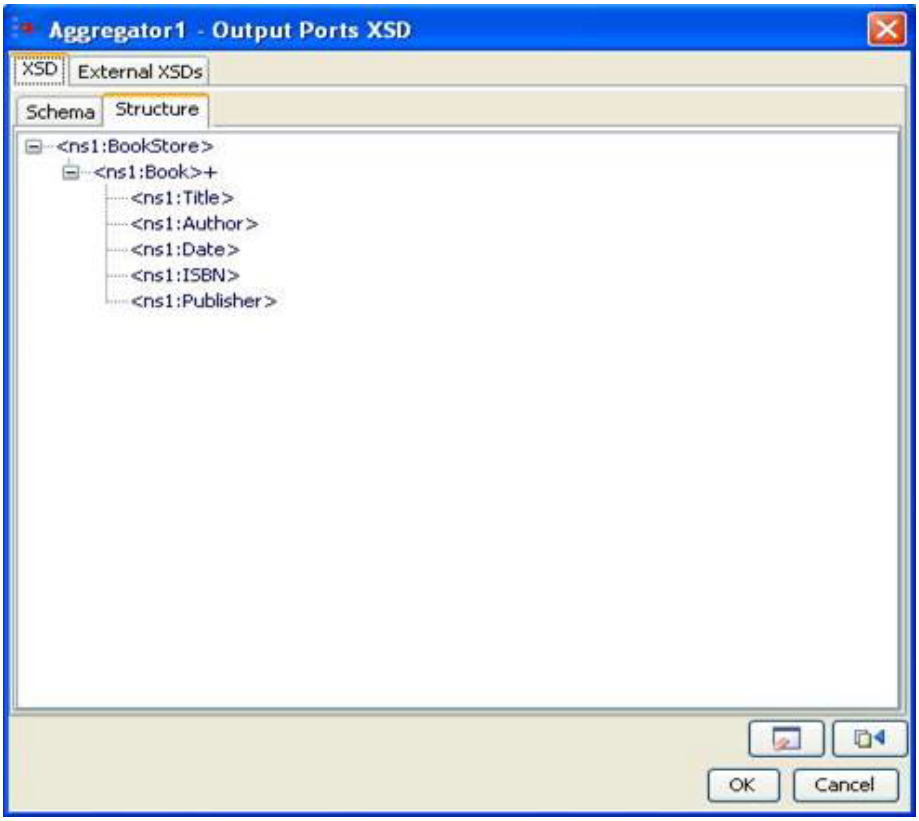


Figure 27: Structure of the schema when Bookstore is chosen as Root Element

The structure of the entire schema is displayed if none of the root element is selected. If root element is selected (as Bookstore in the Figure 27), the structure of that element is displayed.

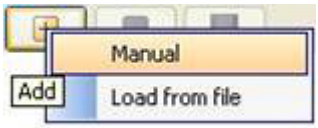
External XSDs Tab

If there are any imported schemas in the schema provided in XSD-Schema tab, they can be resolved by adding them as the external XSDs here. Any number of external schemas can be added here.

Note: Imported schemas can also be resolved by adding the schemas in Schema Repository.

Schemas provided as external XSDs must have target namespace defined.

Click on the **Add** button to add the external schema. Select an option from **Manual** or **Load from File**.



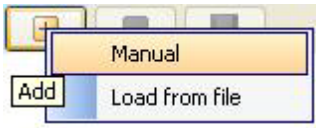


Figure 28: Adding external schema

- **Manual** - The text editor on the right is editable only when Manual option is selected. The schema has to be provided manually in the text editor.
- **Load from File** - Opens a File Chooser to browse the required external schema.

After loading the schema in the text editor, click on **Save** button to save the schema. The schema will be added to the list of external XSDs only when it is saved.

To remove a schema, select the corresponding namespace and click **Remove** button .

To view a schema, select the corresponding namespace and the schema can be viewed in the text editor.

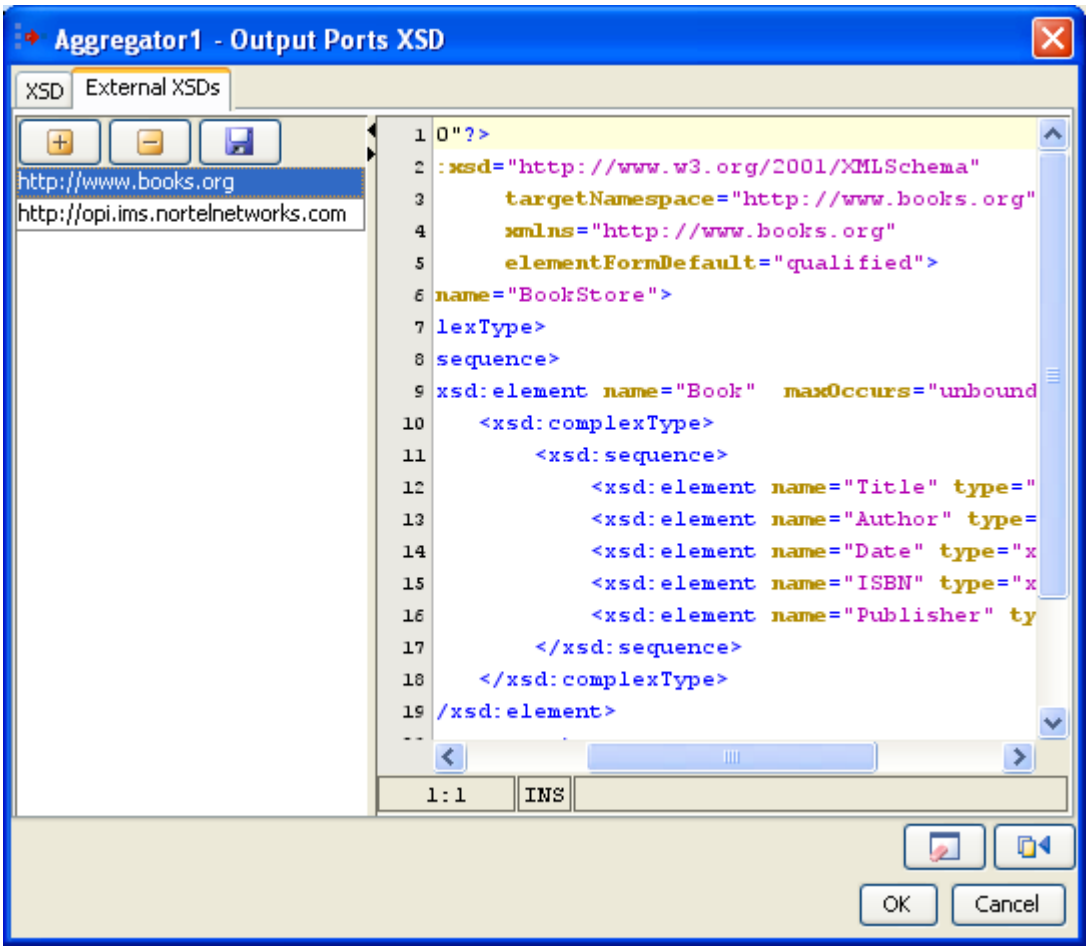


Figure 29: Configuring external schema

Clear

On clicking **Clear** button, the schema, external schemas, root element and structure present in the schema editor will be cleared.

Fetch from Connected Source

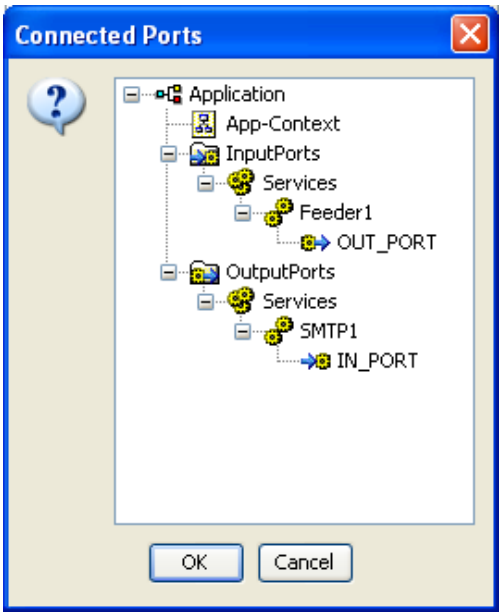


Figure 30: List of connected ports from which schema should be fetched

On clicking **Fetch from Connected Source** button, a list of ports (which have schema set on them) of the components connected to this component are displayed. Application Context of the event process is also listed, if defined. On selecting one of the ports or application context, the schema present will be set as schema in the schema editor.

10.9 Schema Repository

Schema Repository is used to store schemas that are imported in schemas used by different components/event processes. The imported schemas referred from anywhere in an Event Process/component can be stored here so that they are resolved even when they are not added explicitly. Hence, schemas which are imported across multiple event processes/components can be stored in the schema repository.

To add schemas to the Schema Repository, perform the following steps.

- In Studio, navigate to Tools -> Schema Repository. This opens a Schema Repository editor as shown in Figure 31 using which schemas can be added to schema repository.

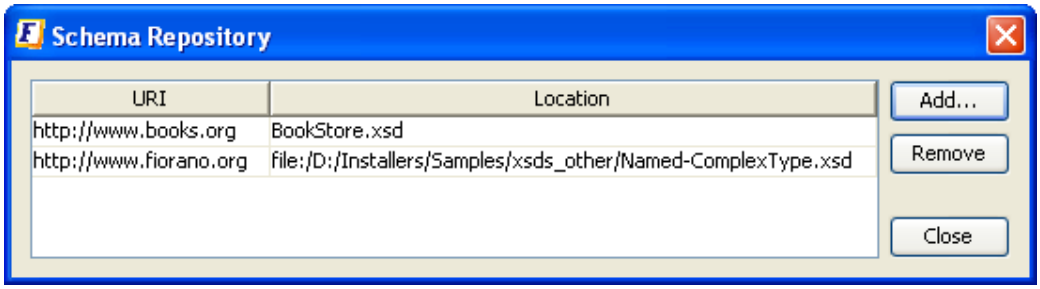


Figure 31: Schema repository editor

- Click on **Add...** Button to add schemas to the repository, **Customize Add...** editor shown in Figure 32 will be opened.

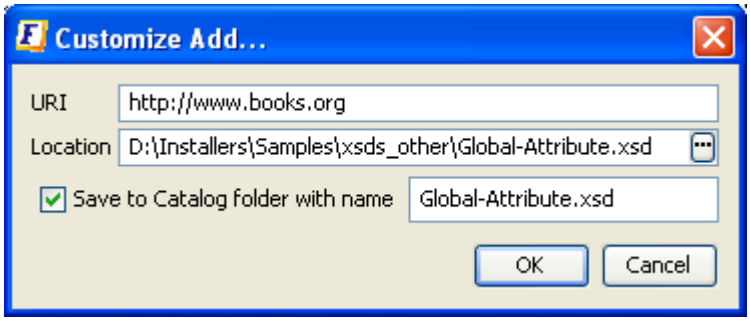


Figure 32: Adding XSD to the schema repository

- Click on ... to browse the required XSD.
- Select an XSD and click **OK**

The values URI, Location, schema name will be automatically updated.

- The URI value should not be an empty field. In case, if the schema has a target namespace, URI should be same as the target namespace of the XSD.
- The Location field displays the absolute path of the schema file.
- If the schema is to be copied and saved in the location <FIORANO_HOME>/xml-catalog/user, select the field **Save to Catalog folder with name** and specify a name with which the file has to be saved.
- If **Save to Catalog folder with name** is not selected, the file is not copied to the location <FIORANO_HOME>/xml-catalog/user and will be referred from its original location.
- Click on **OK** to close **Customize Add...** editor.

A new row specifying the URI and Location of the XSD will be added in the table.

- To remove a schema from the schema repository, select a row from the table and click **Remove**.

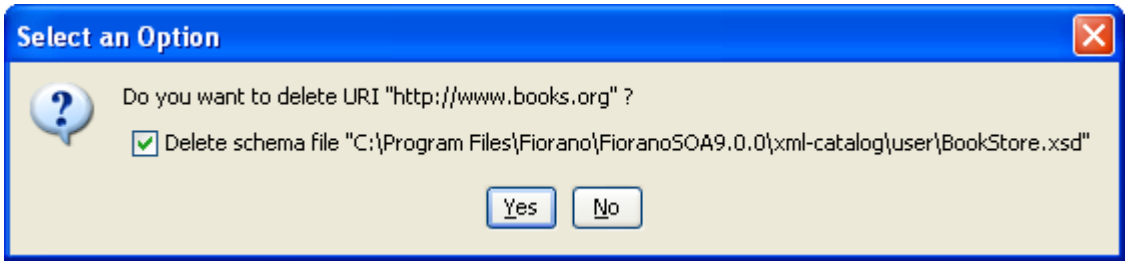


Figure 33: Removing XSD from the schema repository

- The option '**Delete schema file**' specifies whether to delete the file from the system or just to remove the schema from xml-catalog. Select the check box to remove the file completely.
- In case, if the file is not copied to <FIORANO_HOME>/xml-catalog/user, the file will be deleted from its original location if this option is selected.

10.10 XPath Editor

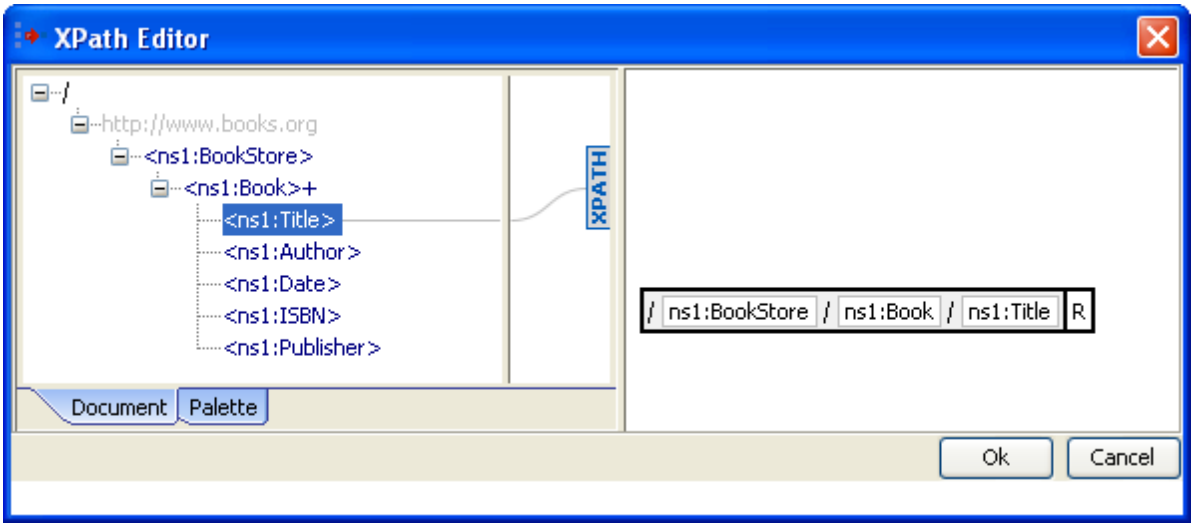


Figure 34: XPath Editor

XPath Editor can be used for specifying path expressions to identify nodes in an XML document and for specifying conditions. The list of elements from schema provided are shown in the left panel of the editor. An XPath Editor with sample schema is shown in Figure 34.

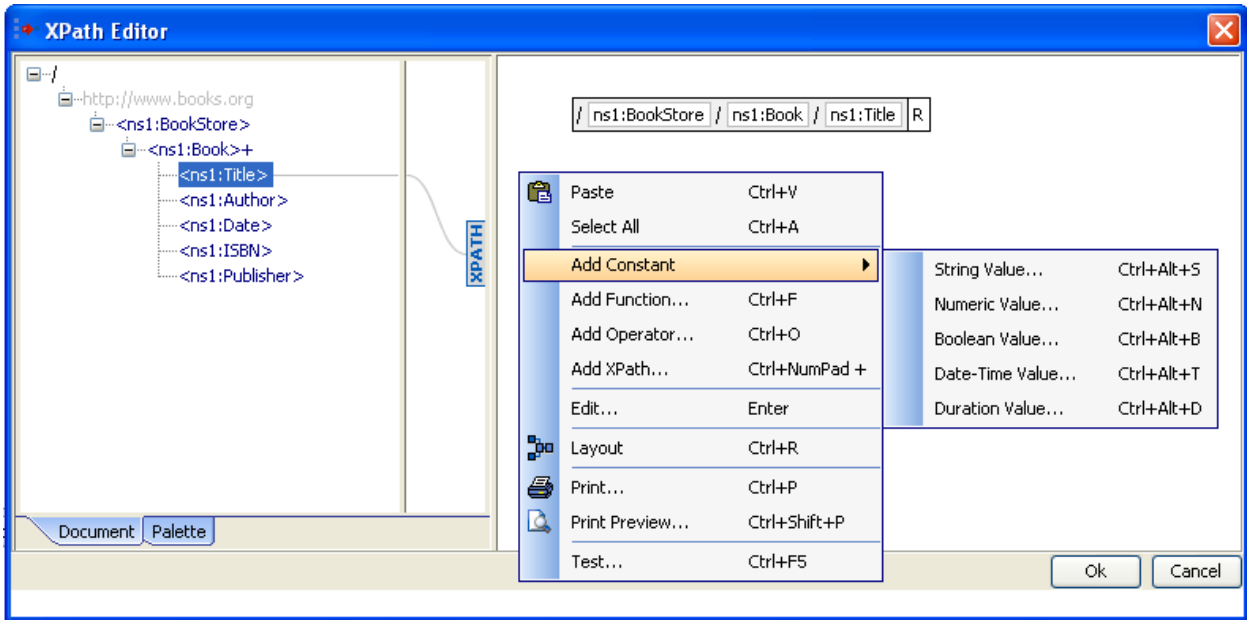


Figure 35: Adding a constant in XPath Editor

An element can be selected by simple drag and drop onto the right panel. An XPath expression may consist of different constant values, functions or/and operators. These can be added easily by right clicking on the right panel and selecting the option based on the requirement as shown in Figure 35.

Adding a Function: A function can be added either by right clicking on the right hand side panel --> Add Function or by selecting from the list available in the **palette** tab which is present in the left panel as shown in figure 36.

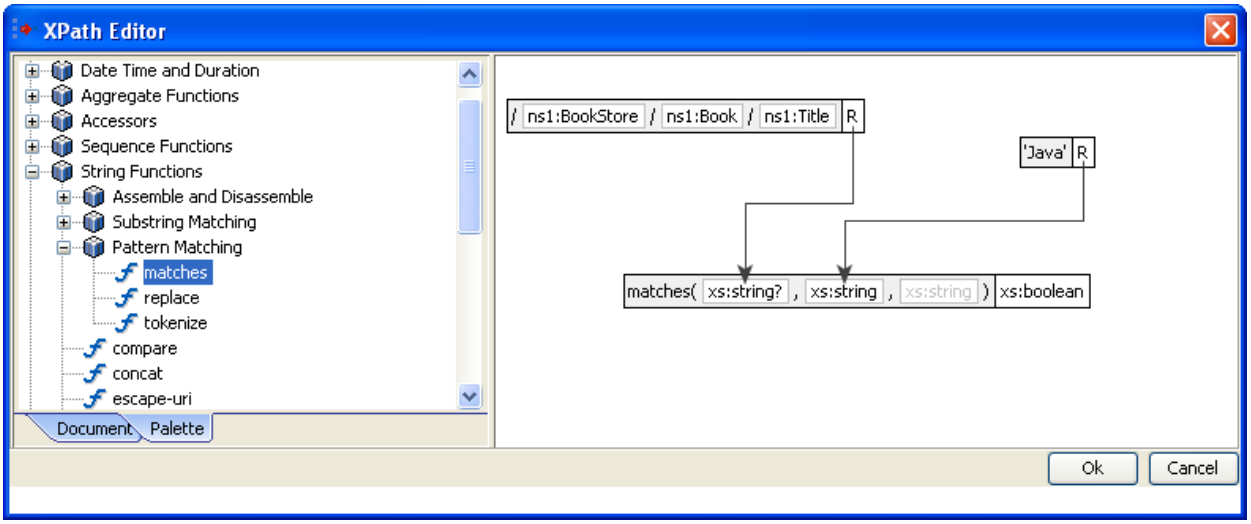


Figure 36: XPath Editor – Palette containing different XPath functions

A string function **matches** which takes two arguments and returns a boolean value is shown in the Figure 36.

Adding a Constant value: Supported types of constants are String, Boolean, Numeric, Date-Time, and Duration.

Example: Addition of a boolean value can be done as described below

7. Right-click on the right panel. Select Add Constant -->Boolean Value.
8. Select the value as shown in Figure 37.

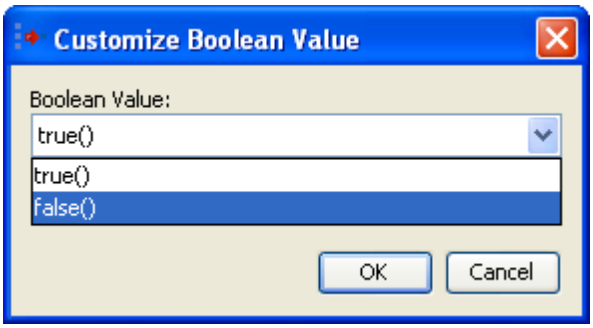


Figure 37: Adding a boolean constant

Addition of a Operator:

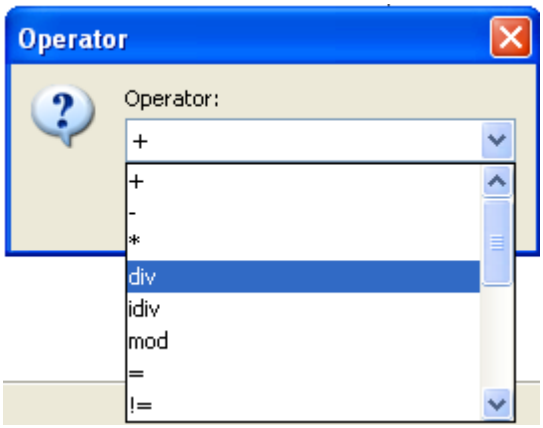
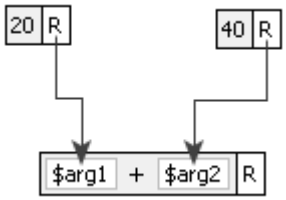


Figure 38: Adding an operator

9. Right-click on the right panel. Click on **Add Operator**.
10. Select the operator as shown in Figure 38.



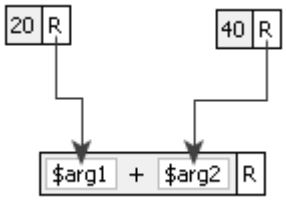


Figure 39: Add(+) operator

Figure 39 illustrates a sample Xpath expression using a '+' operator. It contains two numeric constant values which are passed as arguments to the operator.

10.11 Named Configurations in CPS

Named Configurations may be used in eStudio CPS of components. Configurations can either be loaded from or saved to the repository from CPS itself.

If named configuration is not to be used in the CPS, the text box against Named Configuration shown in the below figures should be left blank and no Named Configuration will be used.

10.11.1 Load Named Configuration

Named Configurations which are created earlier either from the Configuration Repository view or from the CPS itself can be re-used in other service instances.

To view all the named configurations of a particular service or resource type, use the drop-down in named configurations editor present in the CPS. A list of all configurations of similar type present in the repository is shown. Select a configuration name from the list to use that named configuration in a given service instance.

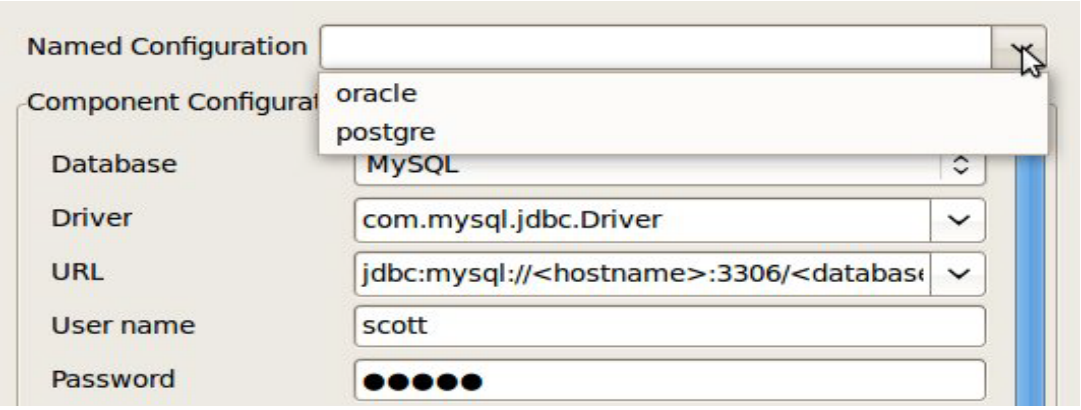


Figure 1: Load Named Configuration

After selecting the named configuration from drop down, the UI will be disabled and the user will not be able to edit the configuration.

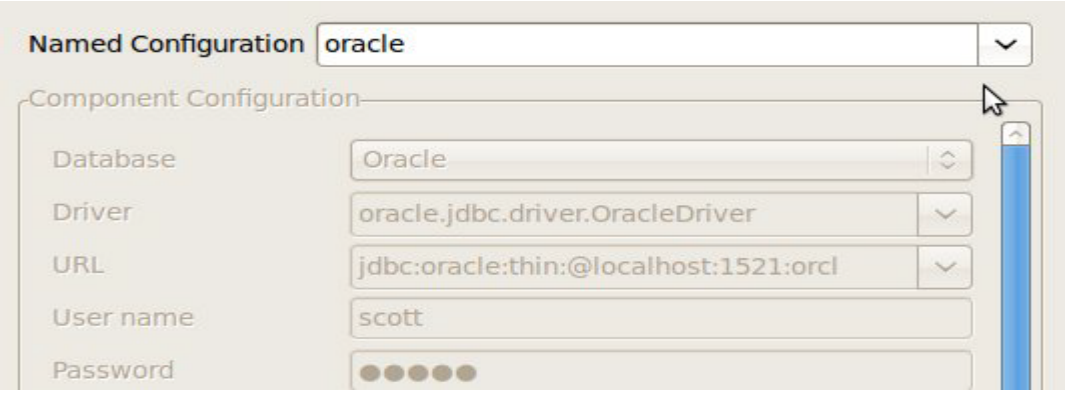


Figure 2: UI disabled after loading Named Configuration

10.11.2 Save Named Configuration

To save named configuration to the repository, specify a name against Named Configuration which is not present in the drop-down list. On finishing the CPS, the configuration is saved to the repository. The UI will be enabled in this case and the user can edit the configuration.

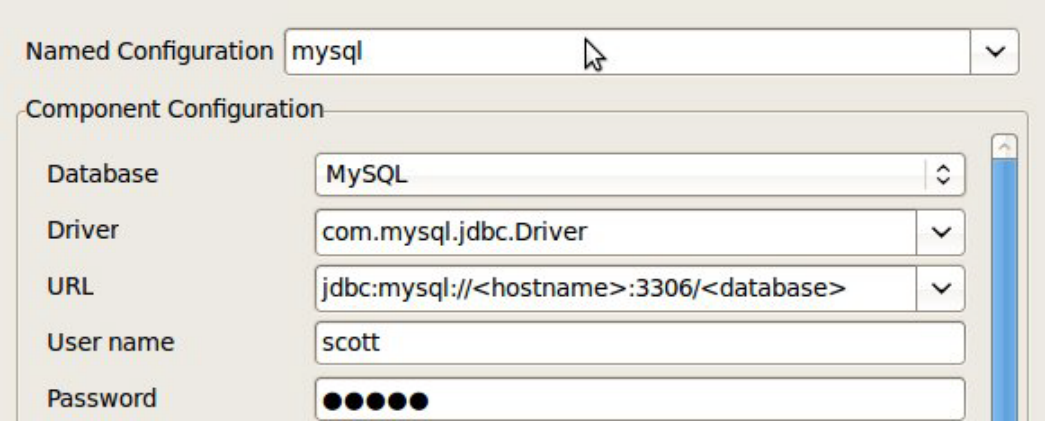


Figure 3: Save Named Configuration

Chapter 11: Monitoring

Fiorano Monitoring enables you to define various policies like backlog monitoring policy and low memory policy. With backlog monitoring one can define the policy over the number of pending requests for a component and raise notifications or alerts appropriately. Low memory monitoring enables you to define policy for the memory usage of servers and components and receive alerts when the specified threshold is reached. The Monitoring features provides a runtime governance mechanism with the ability to generate instant alerts which are delivered via multiple transports including email. This helps administrators keep track of possible errors and also monitor performance bottlenecks at runtime in production systems.

11.1 Configuring Transports

Every alert uses a specific transport for notification. Each Transport has set of transport specific parameters for the message to be carried on that transport. For instance, for an SMTP Transport, the details of the SMTP Mail server together with authentication information are required. These settings can be common across all SMTP Alerts within the system or particular for different cases. The transport setting has default values which are used if the alerts do not specify the settings explicitly.

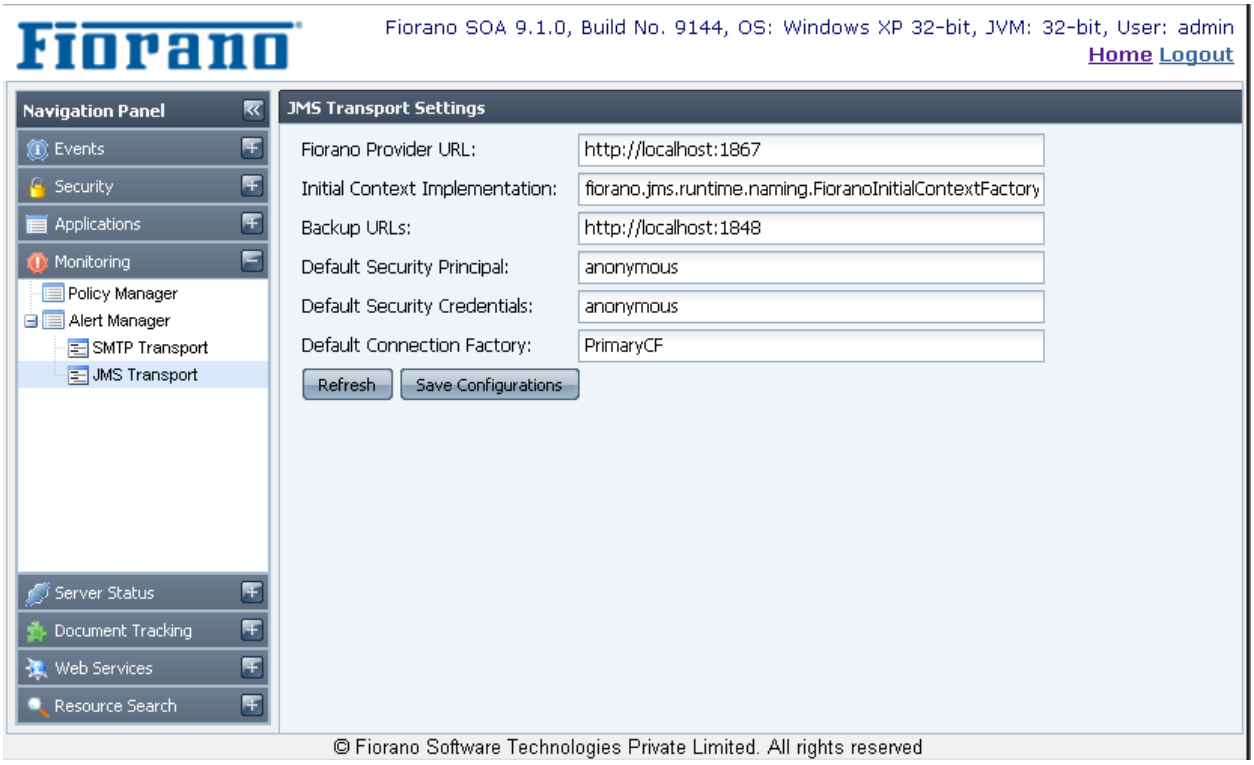
The supported transports are listed under the *Navigation Panel -> Monitoring -> Alert Manager* on the Fiorano Web Dashboard. Clicking on the transport lists the transport specific properties on the right hand side of the page.

11.1.1 SMTP Transport

Properties	Description
SMTP Server	The URL or the IP address of the SMTP Server to connect to for sending the email
SMTP Port	The port on which the SMTP Server is running
Username	Authentication to be used to send the email.
Password	Credentials of the authentication

The username and password configured must have the permission to send emails on this mail server. It is best to create a Fiorano Admin login account for Fiorano SOA and to use that account for all Fiorano notifications. If the communication to the SMTP Server has to go through a proxy server, then use the proxy parameters and **Save** the configuration. All saved values will be used by default for all SMTP Alerts if there is no specific setting provided on the alert.

11.1.2 JMS Transport



Properties	Description
Fiorano Provider URL	The internal URL (or the MQ Layer) of the Fiorano Server to which the message will be published
Initial Context Implementation	The implementation class of the JNDI initial context.
Backup URL	The semi colon separated backup URL(s) of the Fiorano Server if the primary URL is not available. This should point to the secondary server in case of an HA Setup.
Default Security Principal	The default username of the principal to authenticate.
Default Security Credential	The default security credential of the principal to authenticate.
Default Connection Factory	The name of the default uniform connection factory to be looked up by this transport while publishing the message into the transport.

The authentication information provided should have rights to publish the messages to the server. In case the server is set up in a HA mode, the backup URL should contain the URL of the secondary server. You can over-ride these transport settings and provide specific settings for each JMS alert when it is created.

11.2 Managing Alerts

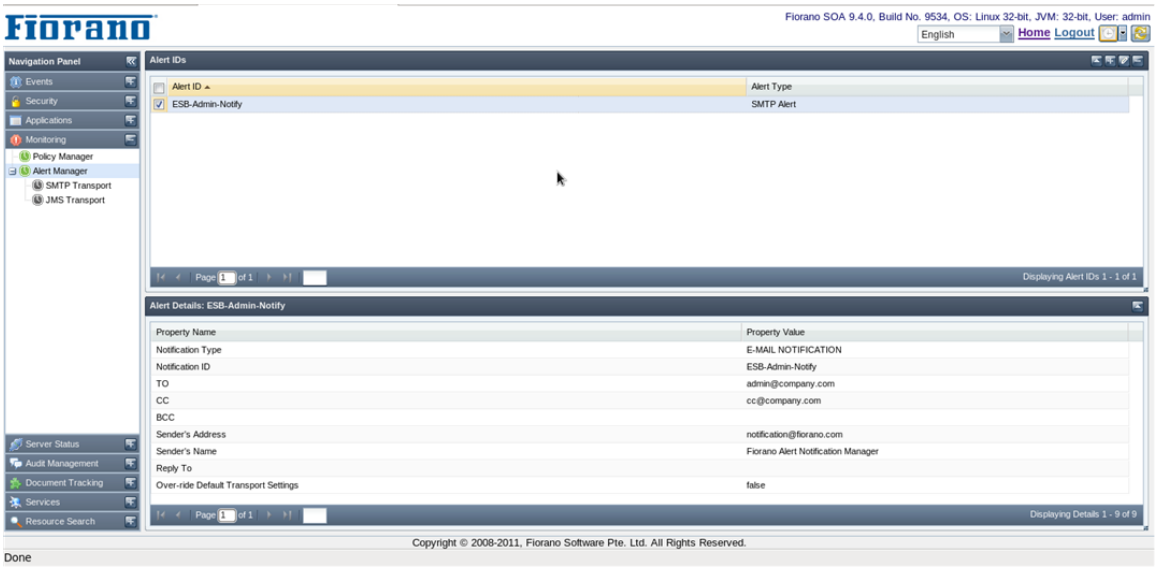
Alerts are a combination of a message and a transport for delivering that message. A message could be in different formats such as plain text, XML, etc and transports could any of the supported ones. Alerts created can be re-used across different policies. The alert manager screen lists all the alerts configured in the system along with the details of the alerts as shown below.

The Main panel lists the alerts and the Transport Type. This is a paged data grid and the pages can be navigated by using the next and previous buttons. A Refresh button reloads the alerts from the Alert Repository. Clicking on an alert displays the details of the alert.

Please refer to section (Enterprise Repository) Alert Repository section to know how the alerts are stored in the Fiorano Enterprise Repository and recommendations on backing them up and migrating.

11.2.1 SMTP Alert

Alert messages are delivered as an Email via the SMTP Transport protocol. This is used to notiy users asynchronously about certain events occurring in the system. Default message content will be in a readable **text** format. The content can be changed to reflect a context specific message. This transport is not ideal to start or execute a workflow.



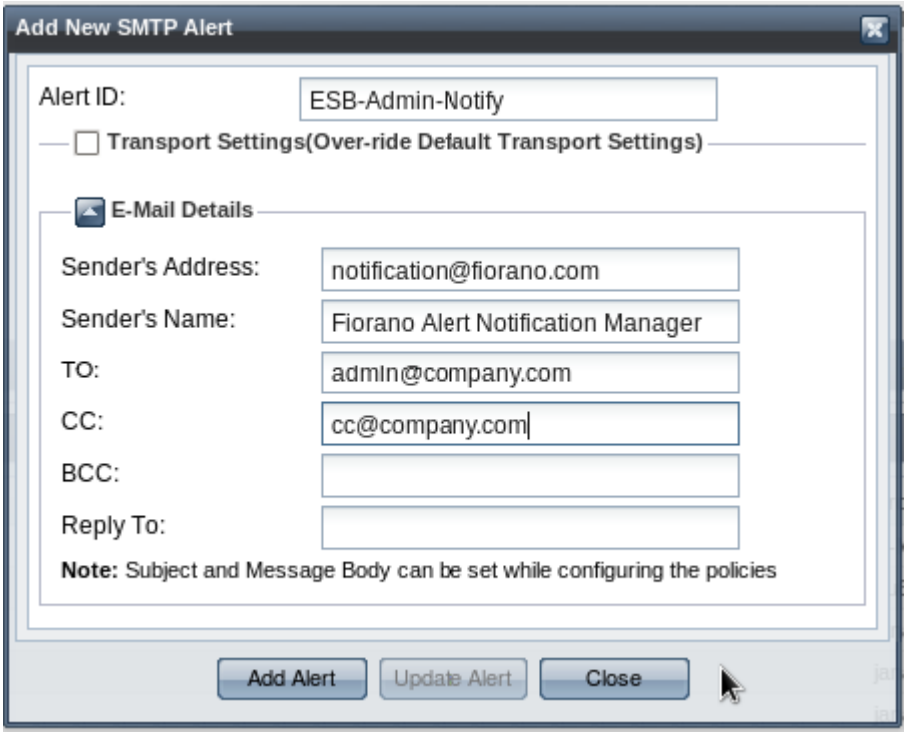
Property Name	Description

Property Name	Description
From Name	The sender Name as it appears on the email client.
NotificationID	Unique identifier of the notification. one should use meaningful IDs to make it easier to reuse this alert in multiple policies.
Message Content Type	Text/Plain. The content type of the message to be delivered.
Message Body	The actual Message content to be delivered.
To	The intended recipient of the notification.
UseTransport	Boolean to indicate whether the default transport settings to be used or the alert specific transport settings have to be used. If true, it means that the alert specific setting has to be used.
Subject	Short description of the content / subject of the message.
CC	The list of recipients to be carbon copied.
BCC	The list of recipients to be black carbon copied.

11.2.2 Creating New SMTP Alerts

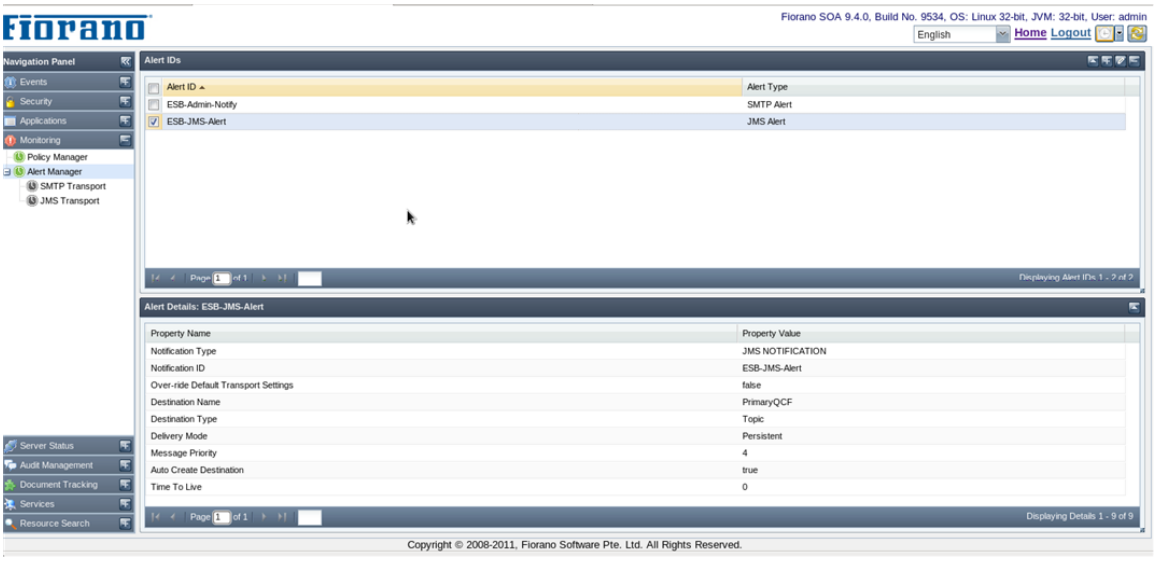
By clicking on the `Add` button on the top right panel, users can add new alerts. A drop–down list of available transports is listed and on selecting a transport, a panel which describes the message format for that transport is displayed.

If you check `Transport Settings`, the Alert Manager will use the transport parameters specified in this alert and the default transport settings are ignored.



11.2.3 JMS Alert

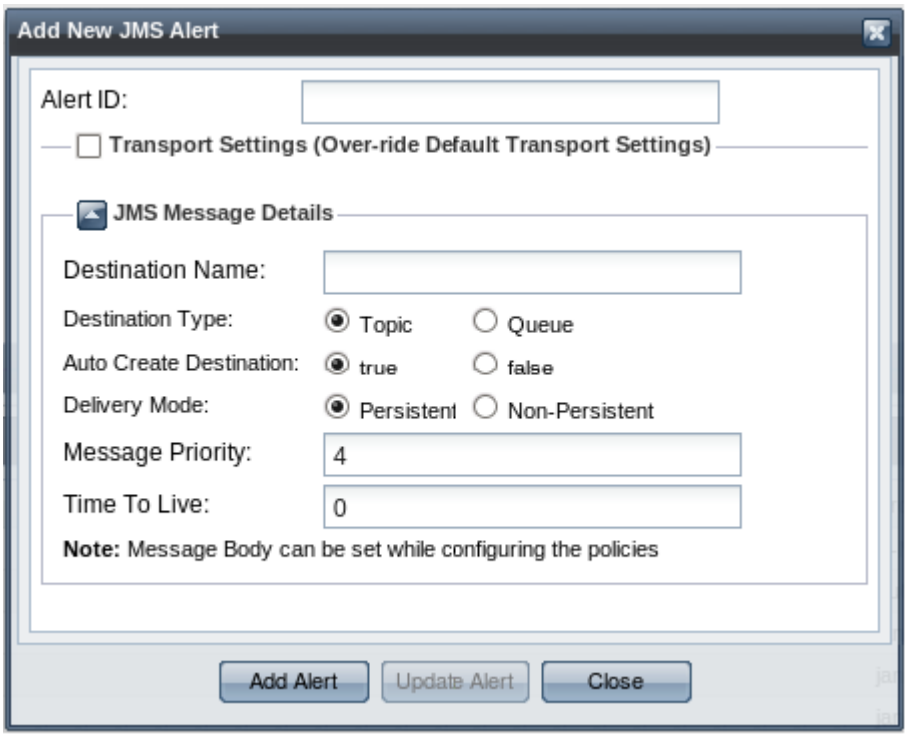
This alert will be used to notify an end-user or a program via the JMS transport asynchronously. This transport is typically used to start a workflow based on this alert condition, and the default message content is in XML format which can be read / parsed by a system and set of workflow actionables can be executed. The Fiorano Event Process can also be used to execute this workflow with the inbound listening to the destination defined in this alert.



Property Name	Description
Connection Factory	The name of the unified connection factory which needs to be used for creating the connection with the specified server.
Destination Name	The name of the destination on which the Notification message has to be published. This can be a topic or a queue and the corresponding destination type has to be specified.
Message Content Type	Text/Plain. The content type in which the message content has to be delivered.
Notification ID	The identifier for this notification.
Content MIME	The MIME of the content. Text/xml by default.
Message Body	The content of the notification.
Delivery Mode	The Mode of delivery. Persistent or non-persistent. Persistent messages are stored in the hard drive if the in-memory buffer is full and are guaranteed to be delivered.
Backup Provider URL	The URL of the backup server to connect if the primary URL is not available.
TimetoLive	TimetoLive property of the JMS Message created.
AutoCreateDestination	Specifies whether to create the destination if it is not present in the target server.
Destination Type	The type of the target Destination (Queue or Topic).
Message Priority	The priority of the JMS Message created.
Username	The username of the principal which will be authenticated while making connection with the server.
Password	The credential of the principal which will be authenticated while making connection with the server.
Use Transport	Specifies whether the transport settings specified in this alert to be used or the

Property Name	Description
	default server settings to be used. If true, the settings local to this alert will be used.

11.2.3.1 Adding a New JMS alert



Click the `Add` button on the top right panel to add new alerts. A drop-down list of available transports is listed. On selecting the JMS transport, the Add New JMS Alert window is displayed as shown in the figure above:

11.2.3.2 Editing/Removing Alert

Alerts can be edited by clicking on the edit icon and can be removed by clicking the remove icon. Once the alert is edited, it will be saved into the repository and will be automatically picked up from the next execution of the alert. `AlertID` and the `Alert Type` cannot be modified. An Alert with the same and different type can be created by deleting the old alerts with the same id. When an alert is deleted and a policy is dependent on it, the policy execution will fail with error logs notifying the user about the system cannot find the specified alert.

11.3 Managing Policies

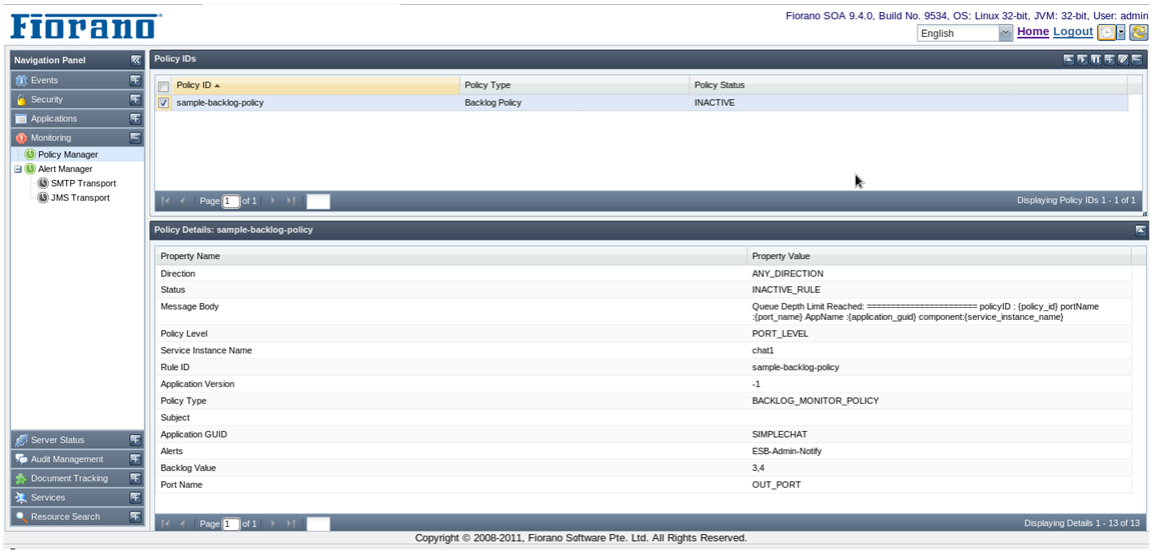
The Main panel lists the policies together with type and current state of each policy. A Policy can be in one of three states: ACTIVE, INACTIVE or ERROR. When a policy is successfully applied it is in ACTIVE state and when there is an error it is in ERROR state. When the user has suspended the policy execution, the policy becomes INACTIVE. The policy list displayed is apaged data grid and the pages can be navigated by using the next and previous tool buttons. A Refresh button reloads the policies from the Policy Repository. Clicking on the policyID displays the details of the policy.

11.3.1 Managing Backlog Policies

The Backlog Policies monitor the backlog request count and raise notifications based on them. Multiple backlog policies can be created and applied at different levels like the Event Process level, Component Level or Port level. The policy applied on an Event Process will indirectly apply to all the components within that event process and will translate automatically to all monitorable ports* of the components.

Note: Monitorable ports are the destinations over which backlog depth monitoring is possible. Currently monitoring is possible only on queues. Future versions of the product may extend backlog monitoring support for topics as well.

The Main panel lists the policies together with type and current state of each policy. A Policy can be in one of three states: ACTIVE, INACTIVE or ERROR. When a policy is successfully applied it is in ACTIVE state and when there is an error it is in ERROR state. When the user has suspended the policy execution, the policy becomes INACTIVE. The policy list displayed is apaged data grid and the pages can be navigated by using the next and previous tool buttons. A Refresh button reloads the policies from the Policy Repository. Clicking on the policyID displays the details of the policy.



Parameters	Description

Application GUID	The event process on which the policy is applied.
Service Instance Name	The service instance name on which the policy is applied if the policy is applied at SERVICE_LEVEL or PORT_LEVEL.
Port Name	The port name on which the policy is applied if the policy is applied at the PORT_LEVEL
Policy Level	The level at which the policy is targeted. It could be targeted for the Event Process / Component / Port level. Event Process translates to all components and component-level translates to all ports of the component.
Direction	The direction of reaching the specified depth. This could be ANY, UP or DOWN. If notification has to be raised when the pending requests are reducing, then DOWN will be specified otherwise UP will be specified. ANY is for both cases.
Depth	The integer value indicating the count of pending requests.
Subject	Short description of the content / subject of the message which is sent along with the alert.(Applicable only for SMTP alerts).
Message Body	The actual Message content to be delivered .
Alerts	The list of alert IDs which will be executed when the condition for this policy is satisfied.

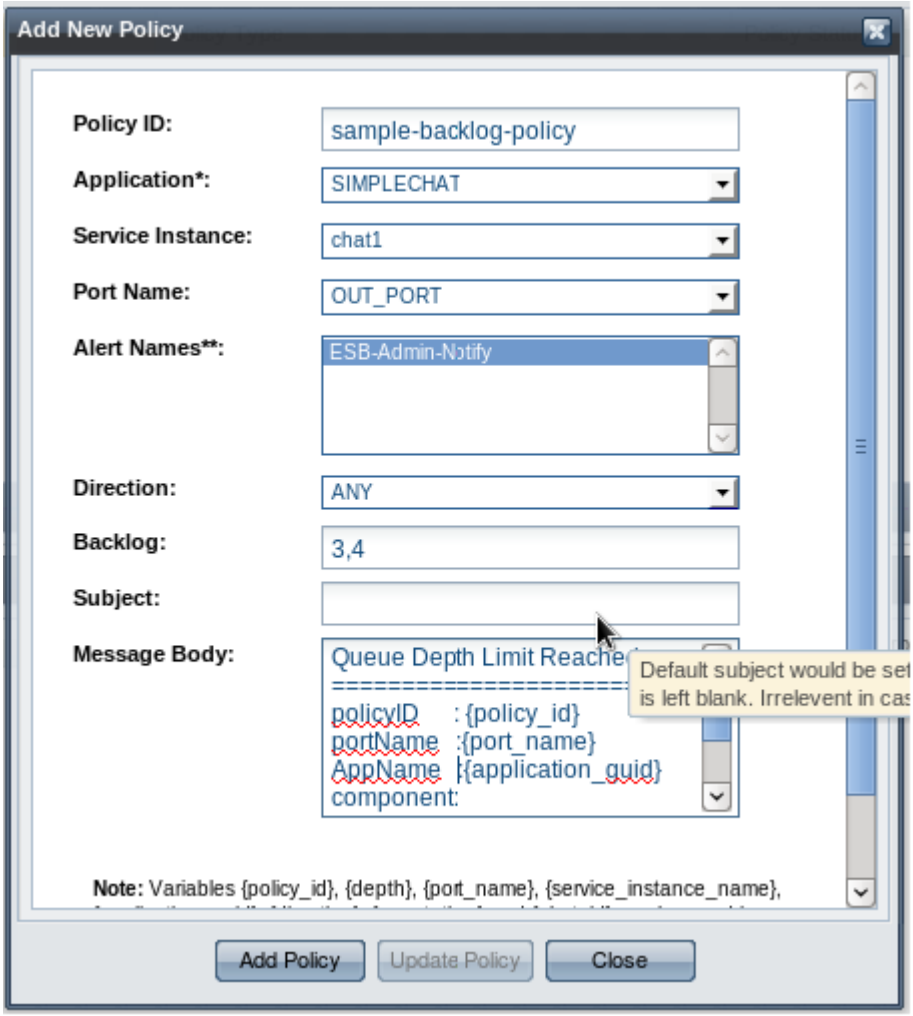
BackLog Policy has a set of variables which can be used in the Subject and MessageBody. Variables help build a context sensitive message / alert in contrast to a constant one. Variables are specific to types of policies. If there is a variable defined which cannot be resolved by a particular kind of policy, it will remain un-resolved in the final message. The table below lists the variables specific to backlog Policy.

Variable Name	Description
{policy_id}	The Policy ID for which this Alert is getting executed. Every time this alert is executed, this variable will get resolved.
{depth}	The count of pending requests for which the policy was configured.
{port_name}	The Port Name at which the requests-pending count reached the {depth} level in Direction {direction}. The port name has the following format: EVENT_PROCESS_ID__SERVICE_INSTANCE_NAME__PORT_NAME
{Service_instance_name}	The name of the service instance for which the requests-pending count reaches the {depth} level in Direction {direction}. The Service Instance has the following format: EVENT_PROCESS_ID__SERVICE_INSTANCE_NAME

{application_guid}	The event process within which the pending requests for {service_instance_name} / {port_name} reached the {depth} level in Direction {direction}.
{direction}	The direction in which the backlog depth was reached.
{event_time}	The date and time at which the backlog depth reached the specified {depth}
{alert_id}	This Alert ID can optionally be inserted in the message body and subject with this variable.

Enabling backlog monitoring policy effect the performance on the concerned peer server (the peer on which destinations exist over which backlog monitoring policies are applied). As such, performance has to be empirically observed after turning on backlog monitoring.

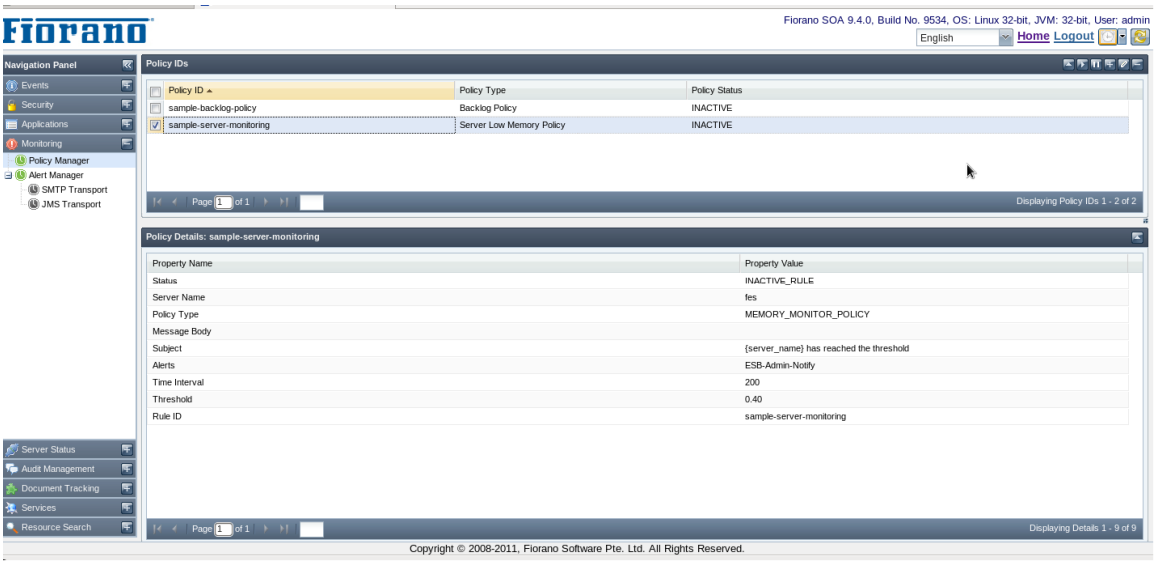
11.3.2 Creating Policy



By clicking the `Add` button, you can add new policies. A drop-down menu of policy types is displayed and on choosing the backlog policy, a panel backlog is displayed and a new policy can be added.

11.3.3 Managing Low Memory Policies

The Low Memory Policies monitor the memory usage of servers and components and raise alerts when the specified threshold is reached. The alerts are raised when memory usage crosses the threshold in any direction.



Parameters	Description
Server	Server on which the policy is applied.
Application	Name of the appliccation which has the component for which the policy is applied.
Service Instance	Component(Service instance) name on which the policy is applied
Threshold Value	The alert is raised if the memory usage reaches this threshold value.
Time Interval	Time interval(in seconds) between two successive same direction alerts.
Subject	Short description of the content / subject of the message which is sent along with the alert.(Applicable only for SMTP alerts).
Message Body	The actual Message content to be delivered .
Alerts	The list of alert IDs which will be executed when the condition for this policy is satisfied.

Low Memory Policy Type has a set of variables which can be used in the Subject and MessageBody. Variables help build a context sensitive message / alert in contrast to a constant one. Variables are specific to types of policies. If there is a variable defined which cannot be resolved by a particular kind of policy, it will remain un-resolved in the final message. The table below lists the variables specific to server Low memory Policy.

Variable Name	Description
{policy_id}	The Policy ID for which this Alert is getting executed. Every time this alert is executed, this variable will get resolved.
{max_memory}	Maximum memory available.
{threshold}	Decimal value
{server_name}	The name of the server.
{time_interval}	The minimum time interval between two successive same direction alerts.
{max_allowed_memory}	The Maximum memory allowed for the server.(max_memory*threshold)
{consumed_memory}	The memory used by the server.
{event_time}	The date and time at which the threshold has been reached.
{alert_id}	This Alert ID can optionally be inserted in the message body and subject with this variable.

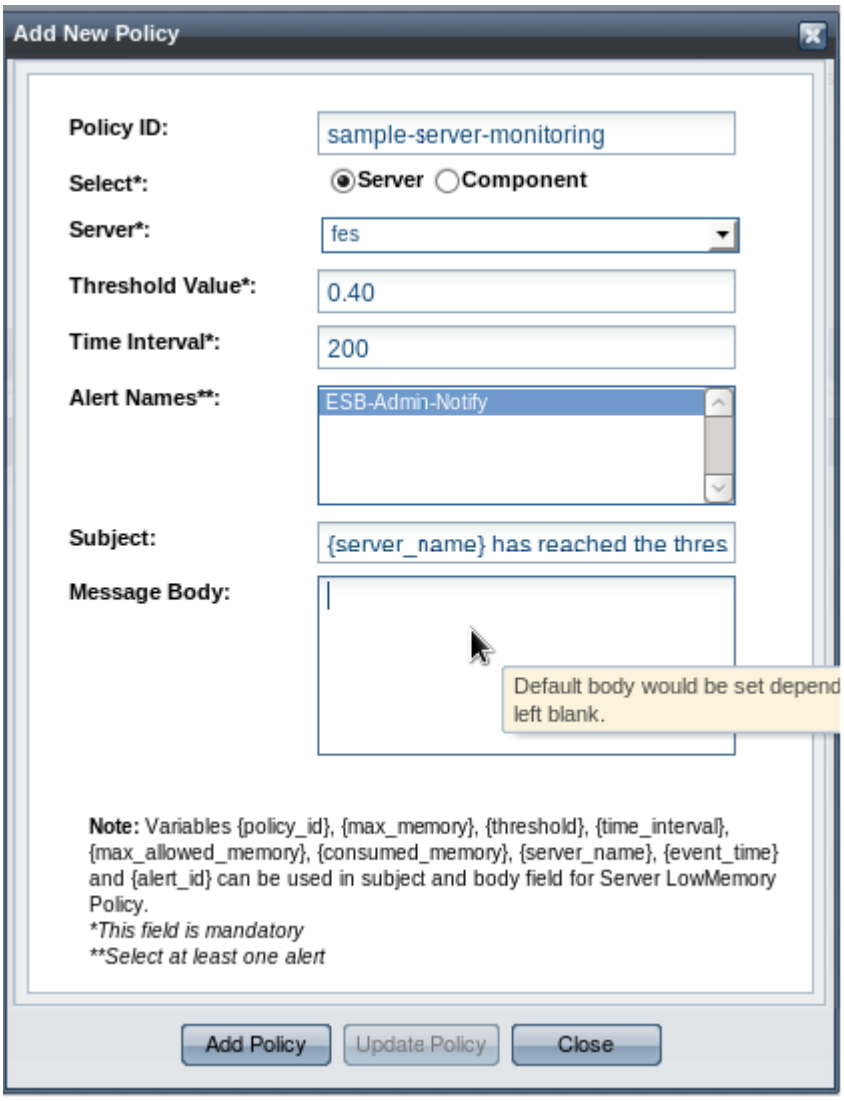
The table below lists the variables specific to Component Low memory Policy.

Variable Name	Description
{policy_id}	The Policy ID for which this Alert is getting executed. Every time this alert is executed, this variable will get resolved.
{max_memory}	Maximum memory available.
{threshold}	Decimal value
{application_guid}	The name of the Application which has the component for which the policy is applied.
{service_instance_name}	The name of the component for which the policy is applied.
{time_interval}	The minimum time interval between two successive same direction alerts.

{max_allowed_memory}	The Maximum memory allowed for the server.(max_memory*threshold)
{consumed_memory}	The memory used by the server.
{event_time}	The date and time at which the threshold has been reached.

11.3.3.1 Creating Low Memory Policy

By clicking the `Add` button, you can add new policies. A drop-down menu of policy types is displayed and on choosing the low memory policy, a window is displayed and a new policy can be added. If you want to create the server low memory policy select the server radio button. If you want to create the component low memory policy select the component radio button.



11.3.4 Managing Policy Executions

Once a policy is created, the policy could be applied on the system and suspended from the top panel by clicking on **Apply Policy** or **Suspend Policy** button respectively. Suspending the policy execution will result in no alerts being generated and the monitors for backlog events for this policy from different peer servers will be switched off. A Policy cannot be edited while it is running; it has to be suspended before you edit a policy. The policy details will be fetched and refreshed with every activate action. Multiple policies can be selected and activated/suspended simultaneously.

The Policy attributes can be edited except for the policyID and the type of the policy. A different type of policy can be created with an ID after deleting the older policy with the same ID. An Active policy cannot be deleted. The policy has to be suspended from execution before it can be removed from the system.

11.3.5 Editing/Removing Policy

Policies can be edited by clicking on the edit icon and can be removed by clicking the remove icon. Once the policy is edited, it will be saved into the repository and will be automatically picked up from the next execution of the alert. `PolicyID` cannot be modified. We can not the edit the Policy which is in ACTIVE state. We have to suspend the policy if it is active before editing the policy.

Chapter 12: Audit Management

Fiorano Audit Management provides visibility into the system and more importantly provides a means of accountability of the changes performed on the system. In certain cases, it can also help in detecting security violations and conducting a post-mortem analysis on the logged information helps narrow down and find the root causes of problems.

Fiorano Audit Management enables you to define policies that specify the type of user/system actions that need to be monitored. The information about each such action (once selected to be audited) is stored in a central audit events database maintained on the Enterprise Server. The database stores audit events generated locally as well as events that are generated within connected Peer Servers.

Audit Events are broadly classified into following categories:

1. Authentication Event
2. Authorization Event
3. Security Database Modification Event
4. Event Process Repository Modification Event
5. Service Repository Modification Event
6. Event Process Life Cycle Event
7. Component Life Cycle Event
8. Principal Store Synchronization Event
9. Alert Repository Modification Event

Each of the above audit events has the following fields:

Field Name	Field Description
User Name	The name of the user who took the action represented by this event. User name will be null/empty if the action was taken directly by the system without a specific user request.
Source Server Name	Name of the server in which the event was generated. As all events in ESB network are stored within the Enterprise Server, this field can be used to distinguish the events that were generated locally within the Enterprise Server or those which were generated by the Peer Servers.
Event Generation Time	The time at which the event was generated.
Event Status	The result of the action (Success or Failure) taken by the user.

Field Name	Field Description
Event Category	One of the following categories based on the severity of the action being performed: • Information • Warning • Error
Event ID	Each event is assigned an ID which represents the type of specific action being performed e.g. Event ID '201' represents new user creation operation.
Event Description	This field provides an abstract description of the action taken by the user. This field might be empty for some events if the fields above are sufficient to describe the event properly.

A more detailed description of each of these audit events is provided in section 12.4 (Description of Audit Events).

12.1 Audit Policies

The decision to audit a particular action by the server is taken based on a policy defined for that particular event. These policies (also refered to as Filters. Storage Policies or Audit Policies) are rules that define certain conditions and the server records all audit events that match these conditions. The policies can be thought of as configuration tools that users have at their disposal to fine-tune the details of events to be recorded. For each type of audit event defined above, a maximum of one policy can be defined. The policies allow users to choose selective actions to be audited using filters defined in the policy. For example,a user may choose to audit only those authentication attempts that were not successful (incorrect username/password combination used at the time of login).

By default, no audit policies are enabled. Users can view a list of passive policies in the Dashboard and enable them selectively. Once policies are activated, events/actions within Enterprise Server/Peer Servers matching those policies will be audited. Users also have a choice of deleting selected audit events and deleting all audit events that match specified criteria(s). Both these actions, are protected by permissions. By default, only administrators have been assigned these privileges. More information on audit permissions is available in section 12.3 Audit Security Permissions.

The following policies are available for each type of audit event. As already mentioned, each policy handles one particular type of audit event and determines whether the audit event corresponding to the policy will be raised/handled by the system (this also depends upon the filters chosen within the audit policy).

- **Authentication Policy**: audits authentication events.
- **Authorization Policy**: audits authorization events.
- **Security Database Modification Policy**: audits Security Database Modification events.

- **Event Process Repository Modification Policy**: audits Event Process Repository Modification events.
- **Service Repository Modification Policy**: audits Service Repository Modification events.
- **Event Process Life-cycle Change Policy**: audits Event Process Life-cycle events.
- **Component Life-cycle Change Policy**: audits Component Life-cycle events.
- **Principal Store Synchronization Policy**: audits Principal Store Synchronization events.
- **Alert Repository Modification Policy**: Alert Repository Modification events.

Each of the policies above defines the following filters using which users can enable selective actions to be audited.

Event Status Filter

Each action that results in an audit event can be classified either as a Successful or a Failed attempt. This filter allows users to selectively enable **Success** and/or **Failure** attempts.

Event Category Filter

Each action that results in an audit event can also be classified as one of Information, Warning or an Error attempt based on the severity of the action performed. This filter allows users to selectively enable **Information, Warning** and/or **Error** attempts.

Event ID Filter (This filter is not applicable for authentication and authorization audit events).

Each audit event generated by the system (other than authentication and authorization events) is assigned a unique identifier. The Event ID field is used to classify audit events based on the type of action being performed, for example, a new user creation action is assigned an event ID of '201' and Event Process update action is assigned an Event ID of '102' and so on. For more information on list of event IDs available for each of the audit events, see section 12.4 Description of Audit Events. This filter thus allows users to selectively enable one or more of these actions to be audited.

12.1.1 Enabling Audit Policies

To start recording a particular audit event, perform the following steps:

1. Navigate to the **Audit Management -> Policies** panel in dashboard and choose the policy that applies to that event. The bottom panel in the displayed page will show the filters available for the audit policy.
2. Choose the actions that you want to audit and click **Save Changes**.
3. On successful execution, the policy status will be shown as 'ACTIVE' (See figure 12.1). You can click **Discard Changes** at any time to re-fetch the policy information from the server and discard all changes performed to that point.

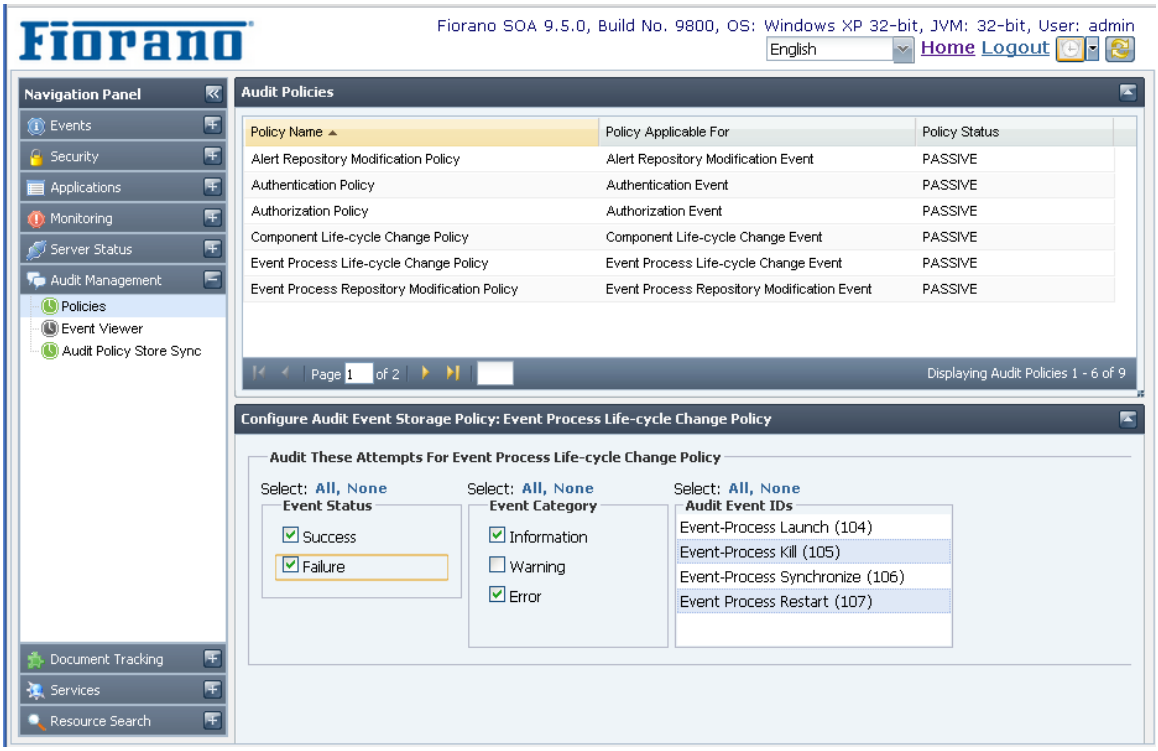


Figure 12.1: Audit Policies Panel

12.1.2 Disabling Audit Policies

To disable an active policy, select the policy. unselect all parameters under the policy and click **Save Changes**. On successful operation, the policy status turns to 'PASSIVE'.

12.1.3 Audit Policy Store Synchronization

Whenever a change is made in any of the audit policies, the change is propagated to all connected peer servers where the same change is made to the policy. Users can view the result of this operation (known as an "audit storage policy synchronization" operation) by navigating to the **Audit Management -> Audit Policy Store Sync** panel on the dashboard (as illustrated figure 12.2). A synchronized audit policy store means that the audit policies stored within the Enterprise Server are active in all peer servers as well. If stores are not synchronized, the status field will display **false** and the reasons for which synchronization could not be achieved are displayed under the **Comment** column. Users may perform corrective actions in such cases. For instance, a user can force a re-synchronization of audit policies with a particular peer server by clicking on the image shown under the **Synchronize** column.

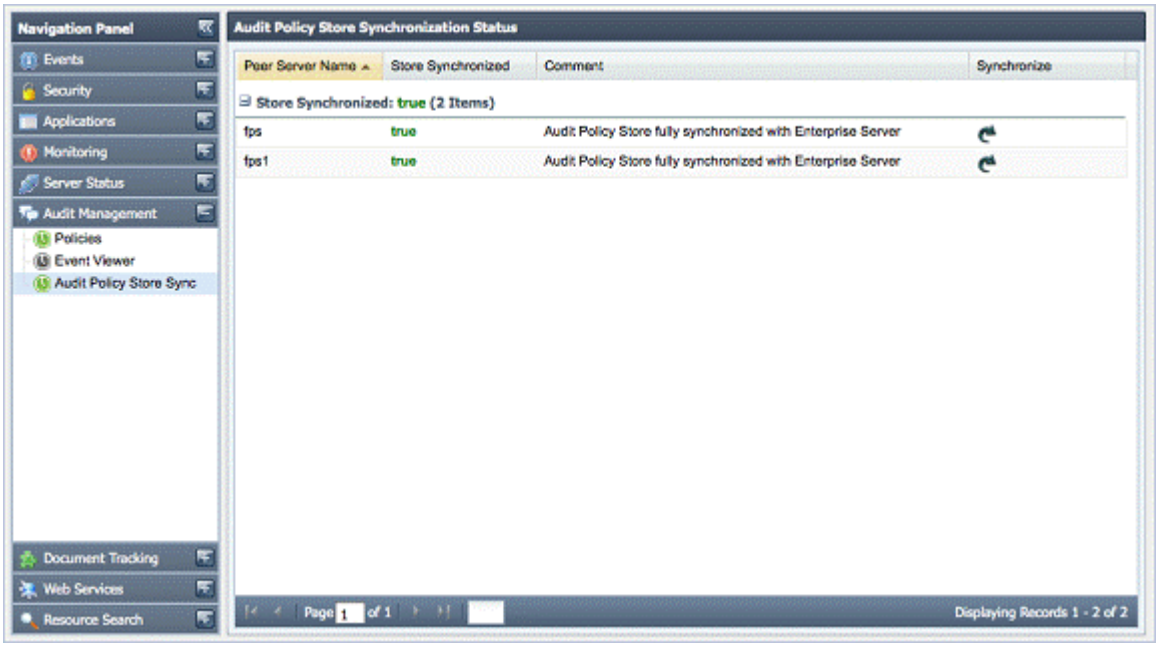


Figure 12.2: Audit Policy Store Synchronization Status

12.2 Handling Peer Server Audit Events

The audit events generated within Peer Servers are sent to the Enterprise Server over a JMS channel and stored in a persistent database accessable by the Enterprise Server. If the Enterprise Server is not available when an audit event is generated within a Peer Server, the Peer Server stores that event locally in a temporary persistent data store. Such events are redelivered to the Enterprise Server as soon as the connection between Peer Server and Enterprise Server is restored. This temporary persistent data store is located in the Peer Server's runtime database as the directory $FIORANO_HOME/runtimedata/PeerServers/<PeerProfileName>/FPS/run/cspCache/ESBX__SYSTEM__PEER__<PeerName>.pubsub/FES_AUDIT_EVENT_TOPIC.

Note: In case of a shared HA server, as the location is: $DB_PATH/cspCache/ESBX__SYSTEM__PEER__<PeerName>.pubsub/FES_AUDIT_EVENT_TOPIC.

12.3 Audit Security Permissions

Changes in the audit database and audit policy store are protected via permissions. By default, administrators have the privilege to create, update and delete audit events. These permissions can be changed via the dashboard as explained in section 2.5 of the dashboard documentation. These actions are governed by the following two permissions:

- PERMISSION TO CREATE OR MODIFY OR DELETE AUDIT STORAGE POLICIES
- PERMISSION TO DELETE AUDIT EVENTS

12.4 Description of Audit Events

This section describes various audit events defined in the previous section. In addition to the common fields described above, each audit event has some additional fields as discussed below.

12.4.1. Authentication Event

This event is raised when a user tries to log into or create a connection with the server. This event can also be used to check if someone is trying to hack into the system by trying some random username/password combinations. This event has one additional field named 'Encrypted Password' which is the encrypted version of the password that was used at the time of login. Multiple failed login audit events with different values in this field might indicate that someone is trying to hack into the system.

Additional properties of authentication event:

Field Name	**Field Description**
Encrypted Password	Provides the encrypted version of the password used to login into the system.

12.4.2. Authorization Event

This event is raised when a user tries to perform an operation that is protected by permissions granted to the user (e.g. looking up a connection factory, creating a JMS connection, launching an event process, changing audit policies, deleting audit events, etc.). This event has a few additional fields that describe the permission being checked and the resource/entity being used.

Additional properties of authorization event:

Field Name	Field Description
Authorized Entity Name	Represents the name of the resource/entity with which this operation is concerned. This can be the name of a connection factory, a topic, a queue, a connection factory, an event process or a global level ESB permission etc.
Authorization Entity Type	Specifies the type of the resource/entity type withwhich this operation is concerned i.e. a topic, a queue, a connection factory, an event process or a global level ESB permission etc.
Permission Authorized	Specifies the permission which was authorized on the resource/entity specified by the above two fields. E.g. 'LAUNCH PERMISSION'/'VIEW PERMISSON' for an Event Process OR, 'SUBSCRIBE'/'UNSUBSCRIBE' permission for a topic/queue.

12.4.3. Security Database Modification Event

This event is raised whenever there is a modification in the security database of the server, for instance: a new user creation, user deletion, group creation, addition of a member to a group, granting a specific permission to a user etc. All these actions are classified by unique event IDs described below:

Event ID	Operation being performed
201	Addition of a new user
202	Deletion of a user
203	Changing password of a user
204	Addition of a new group
205	Deletion of a group
206	Addition of new member to a group
207	Deletion of a member from a group
208	Granting one or more MQ permissions to a user
209	Denying one or more MQ permissions to a user
210	Revoking one or more MQ permissions assigned to a user
211	Deletion of an ACL
212	Deletion of an ACL entry (positive/negative entry)
301	Granting an ESB Global Permission to a user
302	Revoking an ESB Global Permission assigned to a user
303	Modification in ESB Global permissions i.e. either grant/revoke. This ID is used only in those cases where it is not possible to determine whether the operation is a grant and/or revoke operation.
304	Granting one or more ESB application permissions to a user
305	Denying one or more ESB application permissions to a user
306	Revoking one or more ESB application permissions assigned to a user
311	Resetting the Enterprise Network Security Datastore

This event has some additional fields to describe the security resource/entity being modified by the operation and user/group with which this operation is concerned (if applicable).

Additional properties of the Security Database Modification Event:

Field Name	Field Description
Security Entity Name	Specifies the name of the security resource/entity that was directly affected by this operation. E.g. Topic Name (if a topic ACL is being modified), Event Process Name (if and Event Process ACL is being modified), etc.

Security Entity Type	Specifies the type of the security resource/entity based on the 'Security Entity Name' field. For instance, this field can take one of the following values: USER GROUP TOPIC_ACL QUEUE_ACL CONNECTION_FACTORY_ACL GLOBAL_ACL APPLICATION_ACL
Sub Entity Name	Specifies the name of the user/group for which this operation was performed. E.g. the user name to whom some permission was granted/denied or the user/group name that was added to a group.
Sub Entity Type	Specifies the type of the sub entity as specified by 'Sub Entity Name'. This field can take one of the following values: USER GROUP BOTH_USER_AND_GROUP The third value, that is, BOTH_USER_AND_GROUP is used in cases where it is difficult to determine whether the principal(s) concerned are users/groups.
Security Event ID	Specifies the ID of the security audit event. This field takes one of the values as described in the previous table.

12.4.4. Event Process Repository Modification Event

This event is raised whenever there is a modification in the Event Process Repository e.g. when changes are made to an Event Process, the deletion of an Event Process, an addition of an Event Process to repository etc. All these actions are classified by unique event IDs as described in the table below.

Note: Since the Event Process repository is only maintained in Enterprise Server, only Enterprise Server can raise these events.

Event ID	Operation being performed
101	Creation of a new Event Process
102	Update of an existing Event Process
103	Deletion of an Event Process from the repository

This event has some additional fields to describe the properties of the application and its repository.

Additional properties of the Application (i.e. Event Process) Repository Modification Event:

Field Name	Field Description
Application GUID	Specifies the GUID of the application with which this operation is concerned
Application Version	Specifies the version of the application with which this operation is concerned
Application Categories	Specifies all the categories to which the application belongs
Application Count	Specifies the number of applications present in the application repository at the time this event was raised
Application Repository Event ID	Specifies the event ID of the operation as described in the previous table.
File Store Size	Specifies the application repository size at the time this event was raised. Note: This field is not used as of SOA 9.2 and its value is always zero. Future versions of the product might populate a value for this field if so required.

12.4.5. Service Repository Modification Event

This event is raised whenever there is a modification in the component repository of either the Enterprise Server or any of the Peer Servers. Each Peer Server maintains its own local repository of components that gets modified at the time of the CRC (Connectivity and Resource Check) operation. Component Repository modification events have been classified into the following categories:

Event ID	Operation being performed
151	Creation/Registration of a new service to the component repository
152	Updating an existing service present in the component repository
153	Deletion of a service from the component repository
156	Addition of a resource to a component
157	Deletion of a resource from a component
158	Modification in service definition (that is, in the ServiceDescriptor.xml file)

This event has some additional fields to describe the properties of the service being modified by the action taken, together with its effect on the component repository.

Additional properties of the Component Repository Modification Event include:

Field Name	Field Description
Component GUID	Specifies the GUID of the service with which this operation is concerned.

Component Version	Specifies the version of the service with which this operation is concerned.
Component Count	Specifies the number of services present in the local component repository at the time this event was raised.
Component Repository Event ID	Specifies the event ID of the operation as described in the previous table.
Effected Event Processes	Specifies all the running Event Processes that are affected by this operation. For example, when a service is deleted from the Peer Server's component repository, the Event Processes using that service at that time will be affected by this operation.
File Store Size	Specifies the component repository size at the time this event was raised. **Note**: This field is not being populated as of SOA 9.2 and its value is always zero. Future versions of the product may populate relevant values for this field

12.4.6. Event Process Life Cycle Event

This event is raised whenever there is a change in the status of an Event Process, for instance a transition from from stopped to launched state, etc.. Application Life Cycle events are classified into the following categories:

Event ID	Operation being performed
104	Event Process being launched
105	Event Process being killed/stopped
106	Event Process being synchronized
107	Event Process being restarted

This event has some additional fields to describe the properties of the Event Process concerned.

Additional properties of Application Life Cycle Event:

Field Name	Field Description
Application GUID	Specifies the application GUID with which this operation is concerned
Application Version	Specifies the version of the application with which this operation is concerned
Application Categories	Specifies all the categories to which the application belongs
Application Launch Time	Specifies the time at which the Event process was launched
Application Life Cycle	Specifies the event ID of the operation as described in the

Event ID	previous table
Destinations Status	Specifies the status of the destinations (topics/queues) created for components in the peer server. This field can take one of the following values: DESTINATIONS_CREATED DESTINATIONS_CREATION_ERROR DESTINATIONS_DELETED DESTINATIONS_DELETION_ERROR UNKNOWN

12.4.7. Component Life Cycle Event

This event is raised whenever there is a change in Component life cycle e.g. from stopped to running state or vice-versa. Component Life Cycle events have been classified into following categories:

Event ID	**Operation being performed**
154	Service instance launched
155	Service instance stopped

This event has some additional fields to describe the properties of the concerned Component instance.

Additional properties of Component Life Cycle Event:

Field Name	**Field Description**
Component GUID	Specifies the GUID of the service with which this operations concerned
Component Version	Specifies the version of the service with which this operation is concerned
Component Instance Name	Specifies the component instance name as specified in the Event Process
Application Name	Specifies the name of the Event Process to which this service instance belongs
Application Version	Specifies the version of the Event Process to which this component belongs
Component Life Cycle Event ID	Specifies the event ID of the operation as described in the previous table
Component Launch Time	Specifies the time at which component instance was launched

12.4.8. Principal Store Synchronization Event

This event is raised whenever a peer server synchronizes its principal store with that of Enterprise Server. On successful connection, the Enterprise Server sends a command to the Peer Server to synchronize its Principal Store with that of the Enterprise Server. This command can also be issued via the dashboard to force principal store synchronization. Additionally, this event is also raised whenever there is a change in the principal store of the Enterprise Server (e.g. addition of a user/group), in which case the Enterprise Server sends event to the Peer Server(s) to synchronize their principal store(s). This event records the result of these synchronization operations and has been classified into following sub-categories:

Event ID	Operation being performed
401	Addition of a new user
402	Deletion of a user
403	Changing password of a user
404	Addition of a new group
405	Deletion of a group
406	Addition of a new member to a group
407	Deletion of a member from a group
408	Complete Principal Store Synchronization

This event has some additional fields to describe the security resource/entity being modified by the operation and the user/group with which this operation is concerned (if applicable).

Additional properties of Principal Store Synchronization Event:

Field Name	Field Description
Peer Server Name	Specifies the name of the Peer Server with which synchronization was performed.
Security Entity Name	Specifies the name of the security resource/entity that was directly affected by this operation. E.g. Group Name (if group is being modified/created/deleted), User Name (if a user is being modified/created/deleted).
Security Entity Type	Specifies the type of the security resource/entity as specified by the 'Security Entity Name' field. E.g. this field can take one of the following values: • USER • GROUP
Sub Entity Name	Specifies the name of the user/group for which this operation was performed. E.g. the user/group name that was added to a group.
Sub Entity Type	Specifies the type of the sub entity as specified by 'Sub Entity

	Name'. This field can take one of the following values: • USER • GROUP
Principal Store Synchronization Event ID	Specifies the ID of the principal store synchronization audit event. This field takes one of the values as described in the previous table.

12.4.9. Alert Repository Modification Event

This event is raised whenever there is a modification in the alert repository e.g. when changes are made to an Alert, the deletion of an Alert, an addition of an Alert to repository etc. Alert Repository modification events have been classified into the following categories:

Event ID	Operation being performed
901	Creation of an alert to the Alert Repository
902	Deletion of an alert from the Alert Repository

This event has some additional fields to describe the properties of the alert being modified by the action taken, together with its effect on the alert repository.

Additional properties of the Alert Repository Modification Event include:

Field Name	Field Description
Alert Name	Specifies the Name of the alert with which this operation is concerned.
Alert Type	Specifies the Type of the alert (SMTP/JMS) with which this operation is concerned.
Alert Count	Specifies the number of alerts present in the local alert repository at the time this event was raised.
Alert Repository Event ID	Specifies the event ID of the operation as described in the previous table.
Effected Policies	Specifies all the running policies that are affected by this operation. For example, when an alert is deleted from the Enterprise Server's alert repository, the Policies using that alert at that time will be affected by this operation.

12.5 Audit Event Viewer

Audit events can be viewed by navigating to the **Audit Management -> Event Viewer** panel in the dashboard. The event viewer panel provides a capability to filter audit events based on all the fields available in the audit event. Events are retrieved from the Enterprise server in a bunch of size N, where N is the page size (configurable via the bottom toolbar in the grid). For example, to search for audit events, click the **'Filter Events'** button located on the right hand side of the table header and then choose the audit filters for which you want matching events.

The event selection filter window allows adding multiple audit event filters via the **'Add More Audit Event Filter(s)'** option. With this option, a list of available filters is shown where each filter refers to one particular type of audit event. An additional filter, the '**Global Audit Event Filter'** allows searching for all audit events by applying filters on the common properties of audit events. Creating multiple audit event filters would mean that events matching one of those filters would be retrieved and shown within the audit event viewer. A snapshot of filter panel with **'Global Audit Event Filter'** shown in figure 12.3.

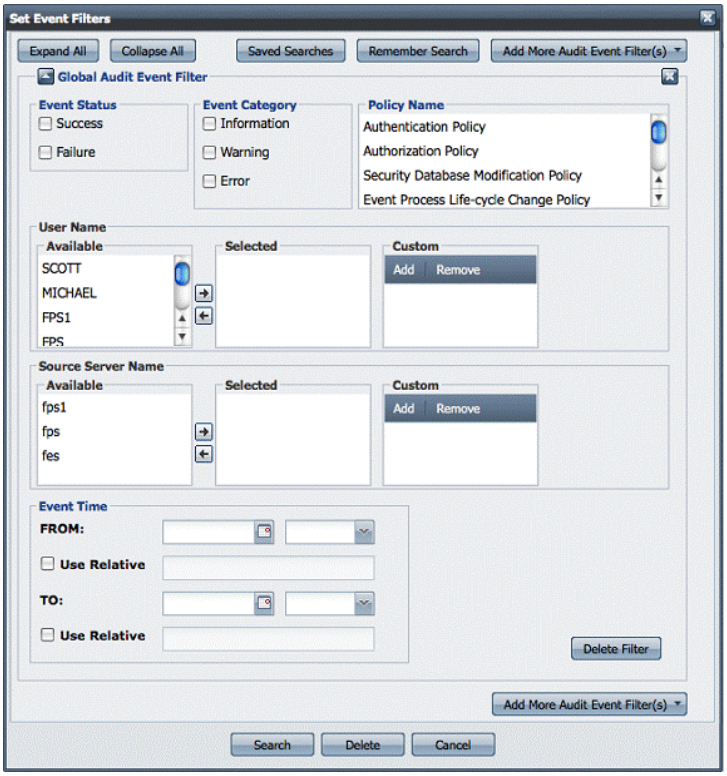


Figure 12.3: Set Event Filters

As illustrated in figure 12.3, most of the common properties that apply to all audit events can be filtered using this panel. Not selecting anything out of a particular filter property is equivalent to selecting everything. The field **'Custom'** under *'User Name'* and *'Source Server Name'* allows the addition of filter values that are not shown under the available values. While searching events, a union of values under the '**Selected'** and **'Custom'** fields is used to match the events that would be shown in the Event Viewer grid panel. Using the *'Event Time'* filter events can also be filtered based on the time at which they were generated. This filter allows you to specify either absolute or a relative value for time (by selecting the *'Use Relative'* checkbox). The relative values of time are evaluated in the server with respect to current time. The relative values of time field accepts values in following format.

- **Now** – Present Time
- **1h** – One hour ago
- **1d** – One day ago
- **1w** – One week ago
- **1m** – One month ago
- **1y** – One year ago

Thus, legal values for the relative date field would be 1d, 2w, 5m, and so on. On clicking the 'Search' button, a typical Event Viewer panel is populated with events as shown in figure 12.4.

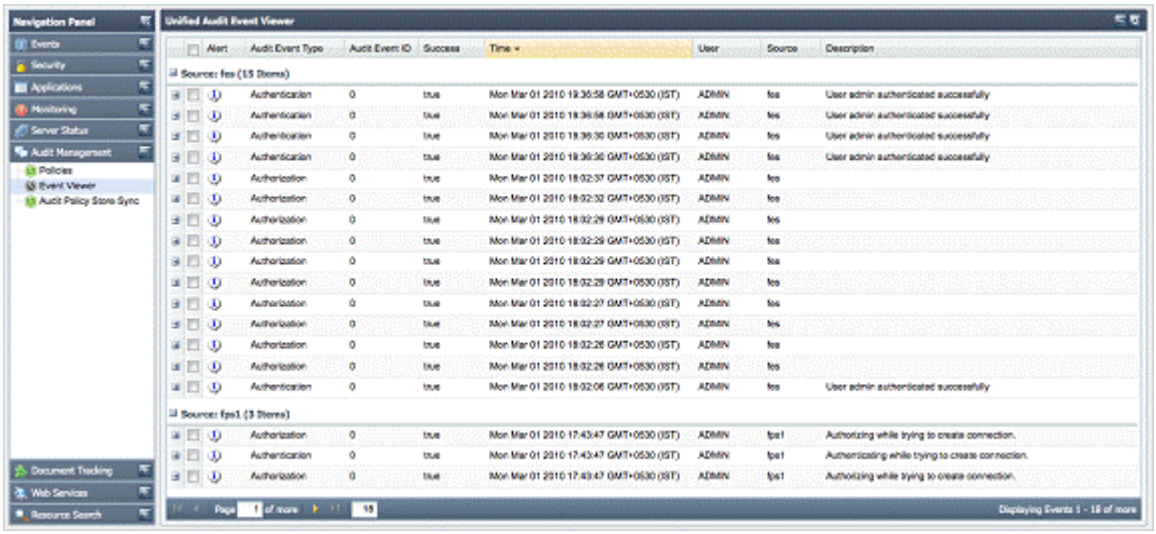


Figure 12.4: Unified Audit Event Viewer

12.5.1 Remembering Search Preferences

As Audit search filters become complex, it becomes difficult to define filters each time a user wants to search for events of interest. The **'Remember Search'** option (as seen in the filter panel snapshot of figure 12.3) eases this task by saving search preferences, allowing them to be used later . On clicking this button, another window pops up as shown in figure 12.5, asking for the name under which the current selected preferences should be saved. You can enter any meaningful name here that briefly describes the filters selected and click OK. The search will be saved in the server to be used later.

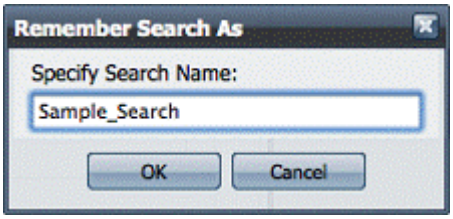


Figure 12.5: Remember Search As

12.5.2 Using Saved Search Preferences

By clicking the **'Saved Searches'** button in the filter panel, a user can view all the saved searches present in the server (as shown in figure 12.6). Clicking are particular search name executes the search and the corresponding results are displayed. If there is a need to edit some of the search preferences before execution, select the **'edit'** option. The filter properties are populated in the filter panel window where user can modify them.

Note: Editing search preferences this way will not actually affect the search filter saved in the server. To over-write the search filter with new preferences, save the modified filter again using the same name as used before.

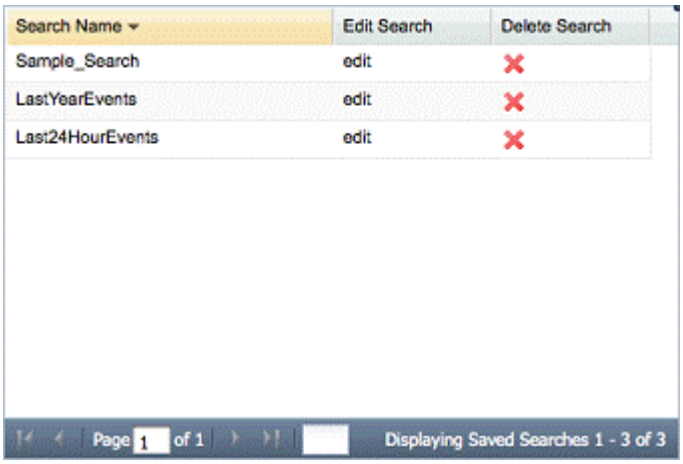


Figure 12.6: Using Saved Searched

Chapter 13: SOA Best Practices

13.1 Development Model

13.1.1 Event Process Development

A business process should be typically composed of multiple Event Processes (EP's). Each EP performs a specific activity. The EP's taken together solve a business problem. EP's have the following properties:

- Consist of not more than 10-15 components for easier management.
- Each EP can be composed by a specific developer.
- The EP uses **port bindings** to communicate between each other. that is, if the Business Process definition requires data to flow from EP1 to EP2 then the OUT_PORT of the last component in EP1 is bound to a specific destination (say, Status_Update). The IN_PORT of the first component of EP2 is then bound to the above destination (Status_Update).

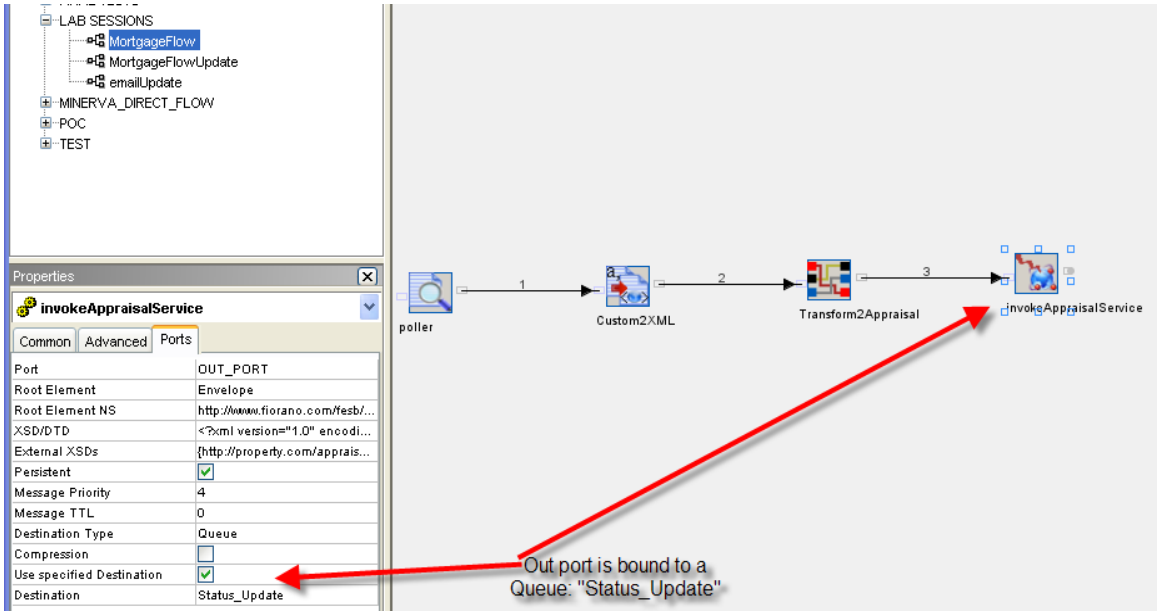


Figure 13.1.1: Event Process 1 (MortgageFlow)

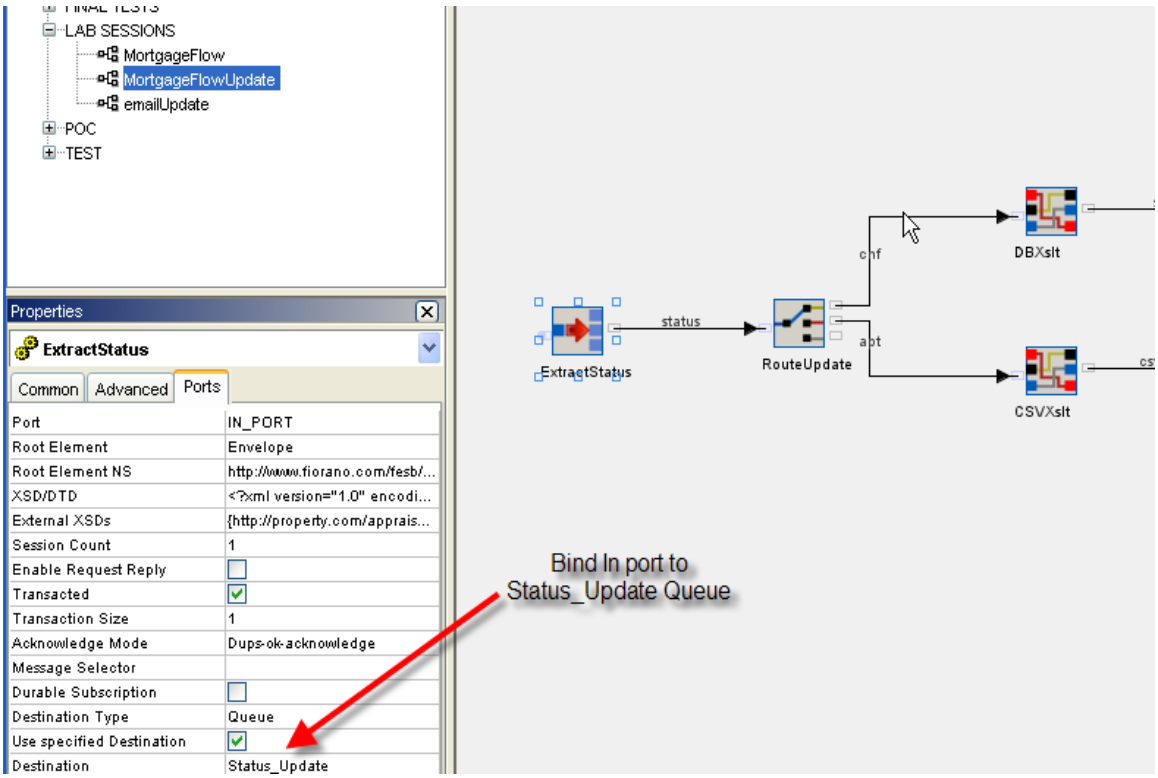


Figure 13.1.2: Event Process 2 (MortgageFlowUpdate)

Now the event processes like MortgageFlow and MortgageFlowUpdate can communicate with each other.

This approach enables developers to:

- Create and test their flows (EP's) individually.
- Handle large number of components in a modular fashion by splitting them up into EP's instead of using one large monolithic process. Later the EP's can be integrated together by simply modifying the port bindings for the components.

Note: The current restriction with the port bindings approach is that the components using port bindings would have to be on the same peer. In case, the components are on different peers, use JMS In/Out adapters to achieve the same effect.

13.1.2 Service Component Development

The new service development kit enables service development inside Eclipse, JBuilder IntelliJIdea and so on, and all the tasks like compiling, building and deploying services into Enterprise Server is done through ANT scripts.ANT scripts come handy when it comes to automating (without manual intervention) the process of building and deploying services into any of QA/Staging and production environments.

13.1.3 Error Handling

A common error process can be defined to handle errors across all business processes. This again could be handled by using the port binding functionality as explained in the "**Development Model**" section.

The service component customization sheet includes an **Error Panel** which gives various options like send on exception port, log the message, and stop the service and so on to handle errors/exceptions. For e.g. connection lost, invalid request and so on, occur within the component. The default Error Panel configuration is to log the errors and send out the same on ON_Exception port. This Error Panel needs to be configured to meet your requirements. For example: To stop the service on first error or to configure the component retry connections and so on.

13.1.4 Explicit Transformations

For easier management of processes and data it is recommended to use the XSLT business component.

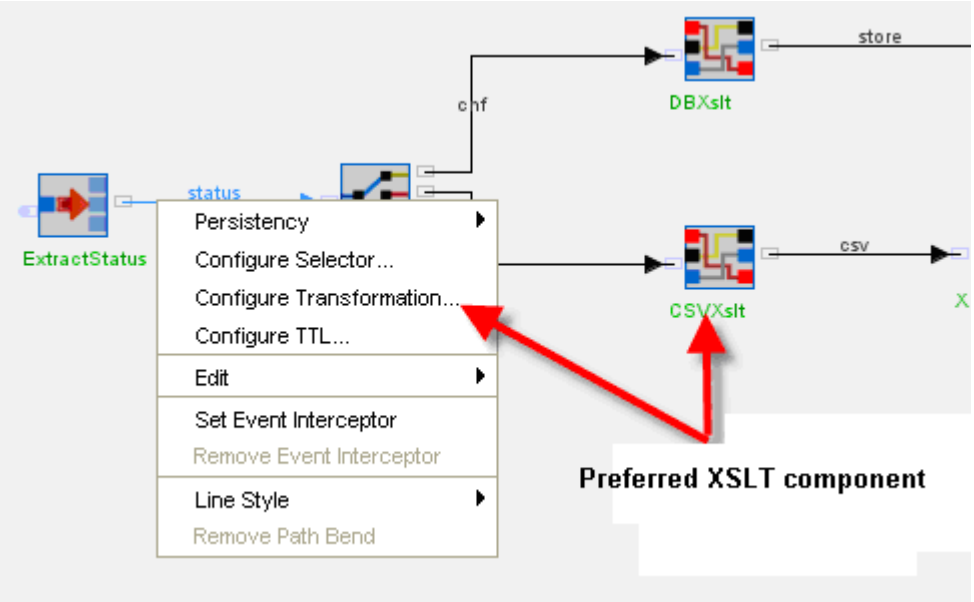


Figure 13.1.3: Configure Transformation

This recommendation also holds good for setting Application Context in a process. Instead of right clicking on a component and setting the App Context, it is better to do this via an XSLT component.

13.1.5 Version Control Integration

As with any other development best practice, it is recommended to keep the files you create/ modify in Fiorano ESB/SOA version/source control systems.

The files typically include:

1. Peer server and enterprise server profiles
2. Event processes
3. Custom service components
4. Custom mapper functions
5. Destinations (that is, queues/topics)

13.2 Testing

13.2.1 Component Level Testing

Fiorano Service Component Development Kit allows you to setup JUnit test cases. It is recommended to start building the test cases along with service component development.

13.2.2 Process Level Testing

It is recommended to develop test cases for each process using JUnit or a unit testing tool.

13.3 Deployment Model

13.3.1 Server Deployment

It's recommended to dedicate peer servers per business process.

Note: A business process consists of one or more event processes. If there is a need to run two business processes on one single machine then run two instances of the peer server, one dedicated for running event processes of first business process and the other peer server instance for running second business process flows. This helps you manage individual business processes without impacting other business processes.

You can run as many as peer servers as you need on a single machine, as long as there's free CPU and Physical RAM.

13.3.2 Event Process and Component Deployment

An event process consists of one or more components that combine to create a business process. The Fiorano SOA deployment architecture allows the components to be deployed on any available peer server across the network.

The precise deployment architecture for each solution needs to be determined experimentally.

The simplest organization is to run all components of an event process on a single peer server. However, as the number of components increase, one has to do the following:

1. Create multiple smaller event processes rather than a single very large event-process; a good rule of thumb is to limit the number of component instances per process to about 13.
2. Run components on different peers (installed on different machines over the network). Fiorano SOA allows you to deploy components dynamically at the click of a button, so you can experiment with running different components on different machines. This becomes particularly important when you have heavy memory processes such as large XML transformations, some of which can bring down an entire machine if there are too many concurrent instances. In such cases, it is always better to run different components on different machines to spread the load and ease memory utilization.

13.4 Performance Tuning and Memory Optimization

13.4.1 Servers

Enabling optimized memory settings:

Server JVM Parameters are located in fiorano_vars.bat. The servers start off with default parameters. The following is an example of setting a box with 1GB RAM:

SET ESB_JVM_SERVER_ARGS=-server -Xms256m -Xmx512m

Log Levels

In dev and QA environments, it is recommended to set the log level to Config that is, the logs contain all the configuration settings. This applies to service components as well. The “Config” level is one level more verbose than “error” (which is the default setting). By setting the log level to “Config” during development and QA, it becomes a lot easier to debug and test flows. Once a flow is ready for production deployment, the log level can be set back to “Errors” (making the logs less verbose). Note that the default log level configuration in Fiorano products is Errors.

13.4.2 Service Components

Service threading options that is, max/min # of sessions and so on

By default, a service component is single threaded i.e. there is only one document being processed by a service component. If you observe that CPU is not at 100% utilization during a load test, then it is highly recommended to increase the # of sessions (i.e. threads) for a component. The number of threads per component can be set within the Properties Window in the STUDIO (as a Main property, in the LHS of the properties window). Always start off with the bottleneck component. There would be only one bottleneck at a time. If you fix one bottleneck component, the bottleneck moves to another component in the flow. Using this technique, one can optimize the number of sessions/threads for each component based on the ability of the hardware to process the flow.

JVM Parameters

The default max JVM heap size is 64MB. STUDIO tool leaves the JVM parameters as default, i.e. 64 MB heap memory for each of the components.
This parameter can be fine tuned to reduce the memory footprint of service components. The amount of memory allocated per JVM can (and should) be reduced for smaller components (such as flow-control components) or increased for memory-heavy components (such as XSLT, Database Adapter and so on)

Large Messages

Try to split the message into smaller XML messages using XMLSplitter component. Any message over 1 MB is considered large message with the default service memory settings. The XML splitter allows a message to be split into multiple messages so take the advantage of this component.

Note:

- Once a message is split, it has to be aggregated back at the target note by using Aggregator component.
- All messages cannot be split. Splitting works only when each of the final messages after the split becomes standalone documents and that's why the XML Splitter cannot be used for all large messages.

Message Attachments (Binary)

Instead of carrying the binary attachment (for example: PDF, XML, Images and so on.) with Tifosi Document/JMS Message or ESBRecord, all across the flow, make use of SAN/NAS to store the attachment and carry forward only the reference.

Note: The above technique works even when the flow executes across distributed machines. The technique is to store large binary attachments in one location (essentially a file somewhere on the network), pass just a reference around the flow and then on the final step to access the (large) binary as needed. Please note that the binary attachment in most cases is not parsed at every step of the event process. As such, it is not needed at each step of the process and a simple reference would do.

Log Levels

In Dev and QA environments, it is recommended to set the log level to Config, that is, the logs contain all the configuration settings. This applies to Peer server and Enterprise Server as well.

The "Config" level is one level more verbose than error (which is the default setting). By setting the log level to "Config" during development and QA, it becomes a lot easier to debug and test flows. Once a flow is ready for production deployment, the log level can be set back to Errors (making the logs less verbose).

Note: The default log level configuration in Fiorano products is Errors.

13.5 Troubleshooting

a. To ensure successful server startup

   It is always recommended to make sure the servers start off fine without ANY errors. If you find an entry in the error log files during the server startup, take a moment to resolve the error. Most of the times, it is the incorrect server configuration or configured paths that do not exist.

   Feel free to contact Fiorano tech support for further assistance. If you choose to ignore the errors and the log files roll over, it might become difficult to debug the errors/exception that may occur later on.

b. To ensure successful Process or Service startup

   Similar to successful server startup, we need to ensure successful start of event processes before using the event process for processing the incoming message load.

Chapter 14: Named Configuration

Named configurations are predefined configurations that are assigned a name and stored for later reuse. These named configurations ease the process of Event Process Orchestration and Change Management within the Fiorano Event Processes. For example, if a particular connection configuration for SMTP or JMS components is reused in multiple Event Processes, each such Service Instance will have its own copy of the configuration. If a change in configuration is required at a later point of time, then all such Service Instances have to be reconfigured. Using named configuration support, configurations can be predefined and the name of the predefined configuration can be linked/transferred to all Service Instances. Since the actual configuration has only one location but is referred to by multiple Service Instances, making changes to the named configuration will affect all Service Instances automatically (without the need to reconfigure the Service Instances again). The route messaging/selector/transformation configuration, port messaging/workflow configuration and runtime arguments/connection factory properties of service instances follow the same principle stated above.

Terminology

Below is a glossary of terms used in this Chapter.

Term Used	Meaning
Named Configuration / Named Object	A name-value pair that stores the configuration given against the name specified
Registry	A location within which all named configurations are stored
Artifact	An artifact can refer to any entity in an Event Process that requires configuration. For example, service instances, routes, ports are some artifacts.

14.1 Advantages

14.1.1 Faster Event Process Composition

Once named configurations exist, then all artifacts using the same configuration can be configured by providing only the name of the configuration. Configuration details do not have to be provided for each artifact separately.

For example, multiple database instances might be configured to use the same database connection properties. Instead of typing these properties again for each such database instances, a User can save this configuration as a named object and reuse it in all database instances.

14.1.2 Easy Change Management

When artifacts like Service Instance or Port Instance are configured using a named configuration, any changes to the configuration at a later point should be applied only to the predefined configuration instead of reconfiguring each artifact.

For example, if the IP address of a database server changes, all database components should use the new IP address to connect to the database server. Once named configurations exist, instead of re-configuring each database instance using the previous IP address, changes can be made just to the saved named object and this will automatically be reflected across all Event Processes.

14.1.3 Easy Resource Management

Easy Resource Management indicates that if Users wish to see details of all FTP Servers used across all Event Processes, they just need to query the registry for all saved FTP connection configurations.

14.1.4 Smaller Event Process Size

Given that configuration data is no longer part of the Event Process (since the Event Process only contains references to the named objects), the size of the Event Process reduces considerably.

14.2 Configuration Types

Configurations can be of the **types** listed below. An explanation for each of these configuration types is given in this section.

- Service Instance Configuration
- Resource Configuration
- Route Configuration
- Selectors Configuration
- Transformation Configuration (Route / App-Context / XSLT Transformations)
- Port Configuration
- Workflow Configuration
- Runtime argument Configuration
- Connection Factory Configuration

14.2.1 Service Instance Configuration

Service Instance Configurations refer to the configuration of component instances in an Event Process. Various sections of the configuration, using the Custom Property Sheet (CPS) of a component [Semantic check please], can be saved and used as named configurations. Examples of component configurations that can be saved as named configurations are:

- Component Connection Configuration
- Business logic configuration / Interaction Configuration
- SSL Configuration
- Proxy Configuration

14.2.2 Resource Configuration

Resource Configurations refer to resources such as files, URLs etc. Various types of resources such as XML, XSD, XSL, TFL, HTML, Script, SSL Configurations and so on can be saved as resource configurations.

14.2.3 Route Configuration

Route Configurations refer to the messaging configurations used by routes. These configurations define the following properties of the routes created within an Event Process:

- Compress Messages
- Encrypt Messages
- Durability of route

14.2.4 Selectors Configuration

Selectors Configurations refer to the selector configurations used by routes. A single selector configuration can be used to define the four types of message selectors available on the routes. Selector configurations store within them the four message selector configurations given below:

- Sender Selector
- JMS Selector
- Message Body XPath Selector
- Application Context XPath Selector

14.2.5 Transformation Configuration (Route / App-Context / XSLT Transformations)

Transformation configurations consist of the three types of configurations given below:

- Message transformation configurations used by routes
- Application context transformation at output ports
- XSLT component transformation

A transformation configuration once defined can be used for any of the three types of transformations given above.

14.2.6 Port Configuration

Port Configurations point to messaging configurations that are used by the input and the output ports of components within Event Processes.

An input port configuration stores the properties given below:

- Is Transacted
- Transaction Size
- Number Of Sessions
- Acknowledge Mode
- Message Selector
- Use Durable Subscription
- Subscription Name

An output port configuration stores the properties given below:

- Time To Live
- Message Priority
- Persistent Messages

14.2.7 Workflow Configuration

Workflow Configurations point to workflow properties used by ports of component instances within Event Processes. A workflow configuration can define the type of workflow to be tracked at a port (such as None, Workflow Item or Workflow End). Workflow configurations also define the **workflow data type** to be used for the workflow which, in turn, is used to define the **type of data tracked**. Workflow data type can be a combination of one or more of the data types given below:

- Message Header
- Message Body
- Attachments
- Application Context

14.2.8 Runtime Argument Configuration

Runtime Argument Configurations refer to the runtime arguments used by component instances within Event Processes. A runtime argument configuration defines a set of name-value pairs that are used at the time of launching the service/component instances. The set of allowed values for names used within these runtime argument configurations depend on the service for which the configuration is being created. For example, a runtime argument configuration created for the **Feeder** component has only two runtime arguments, JVM_PARAMS and JAVA_HOME, where the User can define the values. On the other hand, a runtime argument configuration created for the **Sender** component will have several additional runtime arguments such as totalMessageCount, isTransacted, transactionSize, msgSize and so on.

14.2.9 Connection Factory Configuration

Connection Factory Configurations point to connection factory properties used by component instances to create a connection to Peer Servers. A connection factory configuration defines a set of name-value properties that are used by the component at the time of creating a connection to the Peer Server.

14.3 Defining & Using Configurations

This section explains how Users can create new configurations, modify existing ones and use the configurations already defined at the time of orchestration:

14.3.1 Creating / Modifying Configurations

Configurations can be added/modified to the named configuration repository by using the (two) methods described below:

14.3.1.1 Configuration Repository view in eStudio

eStudio provides a new view named 'Configuration Repository' thorough which Users can create new named objects. To create a new named object, right-click on the **type** of named object to be created and click on **Add Configuration**, as shown in **Figure 1**. A window is displayed where the named object can be created after specifying the required values. In order to modify an existing named object, right click on the object and choose **Edit...** .

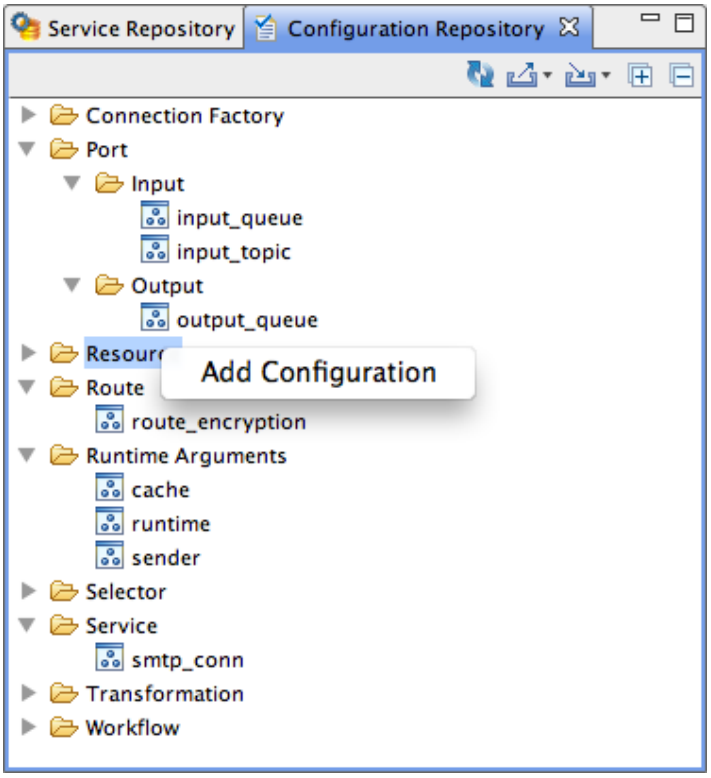


Figure 1: Add Configuration

14.3.1.2 'Save As Named Configuration' in component CPS

At various locations in the Custom Property Sheet (CPS) of components, an option is provided to save the configurations defined in the CPS as a named object, as shown in **Figure 2**. When this option is chosen, the configuration defined in the CPS of the component will be saved against the given named object at the time of closing/finishing the CPS wizard.

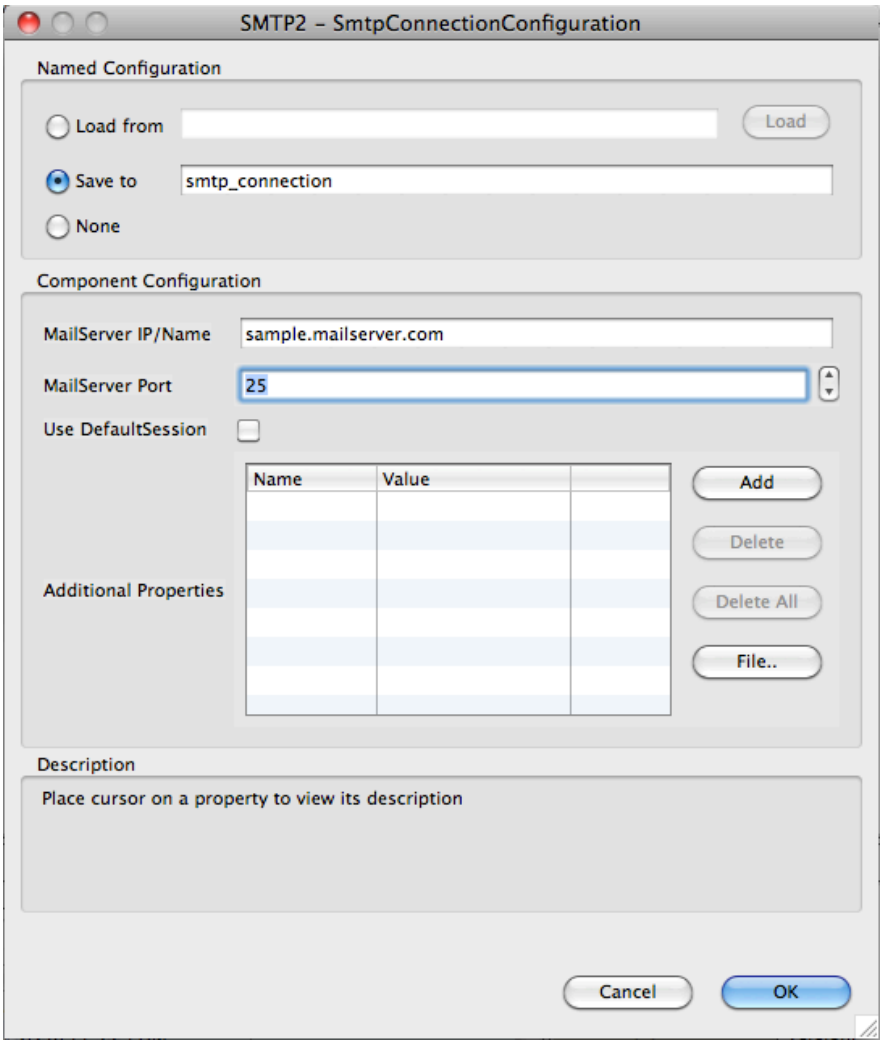


Figure 2: SMTP connection Configuration

14.3.2 Deleting / Renaming Configurations

Named objects can be deleted/renamed using the Configuration Repository view in eStudio. Right click on the desired named object in the Configuration Repository view and choose the appropriate option from the pop-up menu, as shown in **Figure 3**.

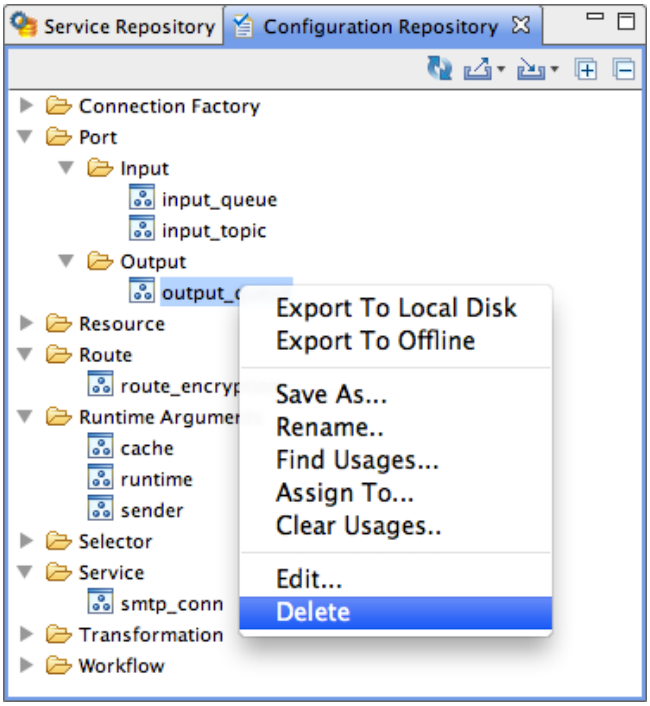


Figure 3: Saving/Deleting/Renaming Configurations

14.3.3 Creating New Configurations from Existing Configurations

To create new named objects as duplicates of existing named object the **Save As**... can be used by right clicking the existing named object, as seen in **Figure 3**.

14.3.4 Using Configurations

Named objects can be used at different places in an Event Process such as routes, ports and service instances. In order to configure these artifacts to use named objects, different UI options are provided in eStudio as explained below:

14.3.4.1 Route via the Properties Panel

When clicked on a route or a port, the properties panel for that artifact is shown. This properties panel shows a Configuration Selector drop-down list that shows all available named configurations available for that artifact. A User can choose one of these configurations for an artifact. The Figure 4 below shows the messaging properties panel for a route, listing the available route messaging configurations. Selector configurations for routes and workflow/messaging configurations for ports can be defined using the properties panel.

Note: Transformation configurations cannot be assigned to routes and output ports using the properties panel. See point #3 below.

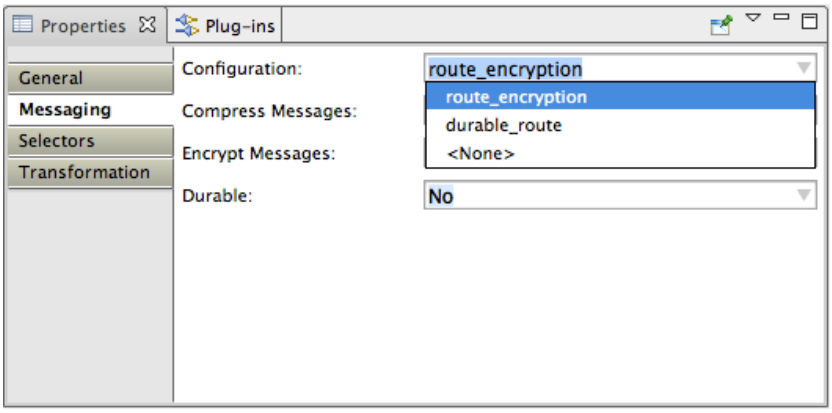


Figure 4: Messaging Properties Panel

14.3.4.2 Route via the Configuration Repository View

The configuration repository view in eStudio provides an **Assign To...** command, which is available by right clicking on a named object, using which users can assign the selected named configuration to specific artifacts shown in a selection list. For example, when a User executes the **Assign To...** command for a route configuration created for the development environment, a list of all routes belonging to the Event Process within the development environment as shown in Figure 5. Users can select the routes to which this configuration is to be applied and click the **OK** button.

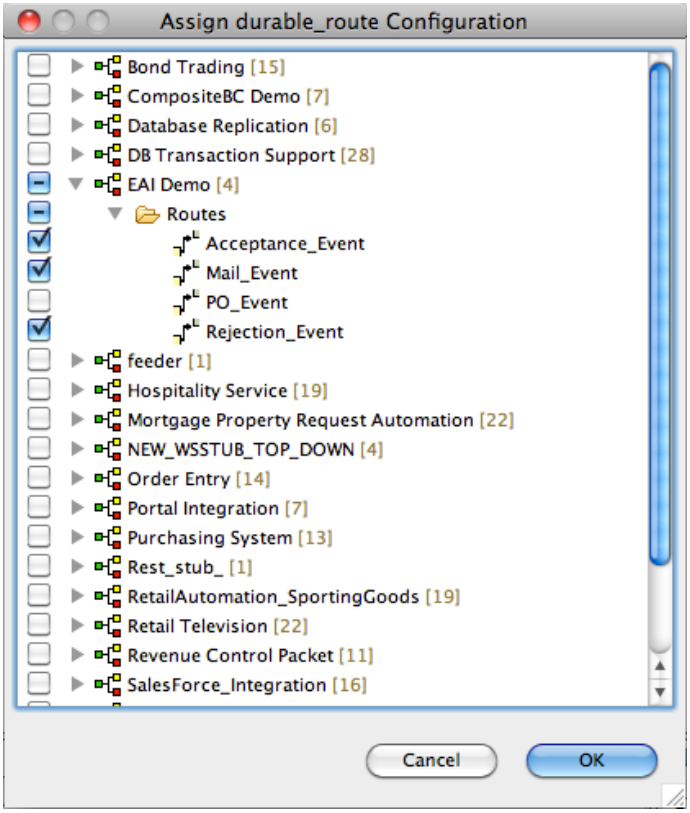


Figure 5: Routes belonging to Event Processes

14.3.4.3 Route using Transformation Configurations

Message transformation configuration on routes and application context transformation on output ports cannot be defined using the properties panel. To perform these configurations use the routes given below:

1. Defining Message Transformation for Routes

   In order to define message transformation on route, right click on the route and choose **Configure Transformation -> Use Named Configuration**. This displays a drop down list with available transformation configurations. Choose a transformation from this list.

2. Defining Application Context Transformation for Ports

   To define a application context transformation on a port, right click on the port and choose **Use Named Configuration.** This displays a drop down list with available transformation configurations. Choose a transformation from this list.

14.4 Finding and Clearing Usages

To find the list of artifacts (routes, ports, services) using a particular named object, a User should right click on the named object in the Configuration Repository view and select the **Find Usages...** option. This action will show a list of all artifacts using the named object.

To clear the usages of this named object, select **Clear Usages...** option and select the artifacts from which the usage should be cleared and click **OK** . The configuration name for these artifacts will be set to **None** and the effected Event Processes will be automatically saved/deployed to the server.

14.5 Validating Configurations

At various stages during the creation/running of an application, a check is made to confirm the presence of all named objects used in the Event Process. If any of the named object is found missing from the registry, an appropriate error message is thrown and the operation is not allowed to execute unless this problem is resolved. The various stages at which this check is made are while:

- Saving the Event Process
- Connectivity Resource Check (CRC)
- Launching the Event Process
- Synchronizing the Event Process

In addition to the above, a check is made to confirm the presence of all named objects used by a service instance at the time of starting the service instance. This check confirms the presence of all named configurations used within the Custom Property Sheet [CPS] of that particular service instance and the named configurations used at the ports of that service instance.

14.5.1 Error Handling

Though the CRC check may be fine, there might be some runtime errors that may crop up with regard to named configurations. For example, a named object might be incorrectly referenced. An example of this is when a named object representing the SSL configuration for a service instance might be specified at a place where the Proxy configuration is expected. Such issues will be resolved at run-time when the Event Process/Service Instance is launched. An exception message will be shown/logged, wherever possible.

14.6 Resolving Named Configurations

Since Event Processes are configured to use named configuration features, they may refer to many named objects on ports, routes or service instances. All these named objects have to be resolved using their actual configuration [Semantic check please] before launching the Event Process. These named objects are resolved during runtime of the respective artifacts as described below.

14.6.1 Service Instances

The named objects referred from the Custom Property Sheet (CPS) of components are resolved at the time of launching/synchronizing the Event Process (and not at the time of CRC). Once these service instances have been launched, they continue to use the configuration data from the registry at the time they were launched. For the changes made to named objects used by service instances to take effect, these service instances have to be restarted either by stopping all the service instances and starting them again or by restarting the Event Process. To summarize, service instance CPS configurations are resolved during:

- Launching the Event Process
- Synchronizing the Event Process
- Starting a stopped service instance

Note: During the synchronization of Event Processes, changes made to named objects used by running the service instances do take effect. Only those changes made to named objects used when the service instances is stopped take effect.

14.6.2 Port Instances

As ports are bound to service instances, the named objects referred from port instances of services follow exactly the same rules for getting resolved as those followed by the named objects referred from the CPS of components. Please refer to the Service Instances section above.

14.6.3 Routes

The named objects that are referred from routes are resolved at the time of launching/synchronizing Event Processes. **Note:** During synchronization of an Event Process, changes made to named objects used by routes that already exist will not take effect. Only those changes made to named objects used by newly created routes take effect.

14.7 Security

The permissions given below are available at the Enterprise Server Level to govern the usage of named configurations within Event Processes. By default, only the administrators group is allowed to perform the actions protected by these permissions. Administrators can also assign permissions to other Users to perform one or more of these actions. Navigate to the **Security -> Global Permissions** page in the dashboard to view and modify these permissions.

1. Permission to view and use named objects

   This permission governs whether a particular User has permission to view the list of named configurations present in the Server. The permission also governs whether those named objects can be used within Event Processes.

   When a User who does not have this permission tries to save/launch/synchronize an Event Process that uses named configurations, an exception will be thrown stating that the User is not allowed to use the named objects from the configuration repository.

2. Permission to add and/or delete named objects

   This permission governs whether a particular User has the permission to add and/or delete named objects from a configuration repository.

3. Permission to modify existing named objects

   This permission governs whether a particular User has the permission to edit/modify the data stored within a named object.

Chapter 15: Fiorano BPM Tool

Fiorano is integrated with a BPM tool called as YAWL (Yet Another Workflow Language). YAWL is a BPM/Workflow system based on a concise and powerful modeling language that handles complex data transformations, and full integration with organizational resources and external Web Services.

15.1 Enabling BPM Tool

By default the BPM tool is disabled in Fiorano SOA. To enable the BPM tool, perform the steps describe below:

1. Open the Enterprise Server profiles from eStudio and check the enableStart property as shown in figure 15.1 and save the profile.
2. Also provide the full path of Worklet service repository in FIORNAO_HOME/esb/server/jetty/yawl/webapps/workletservice/WEB-INF/web.xml.
3. Start the Enterprise Server. BPM tool uses jetty server to launch the BPM engine and listens to port 1990.

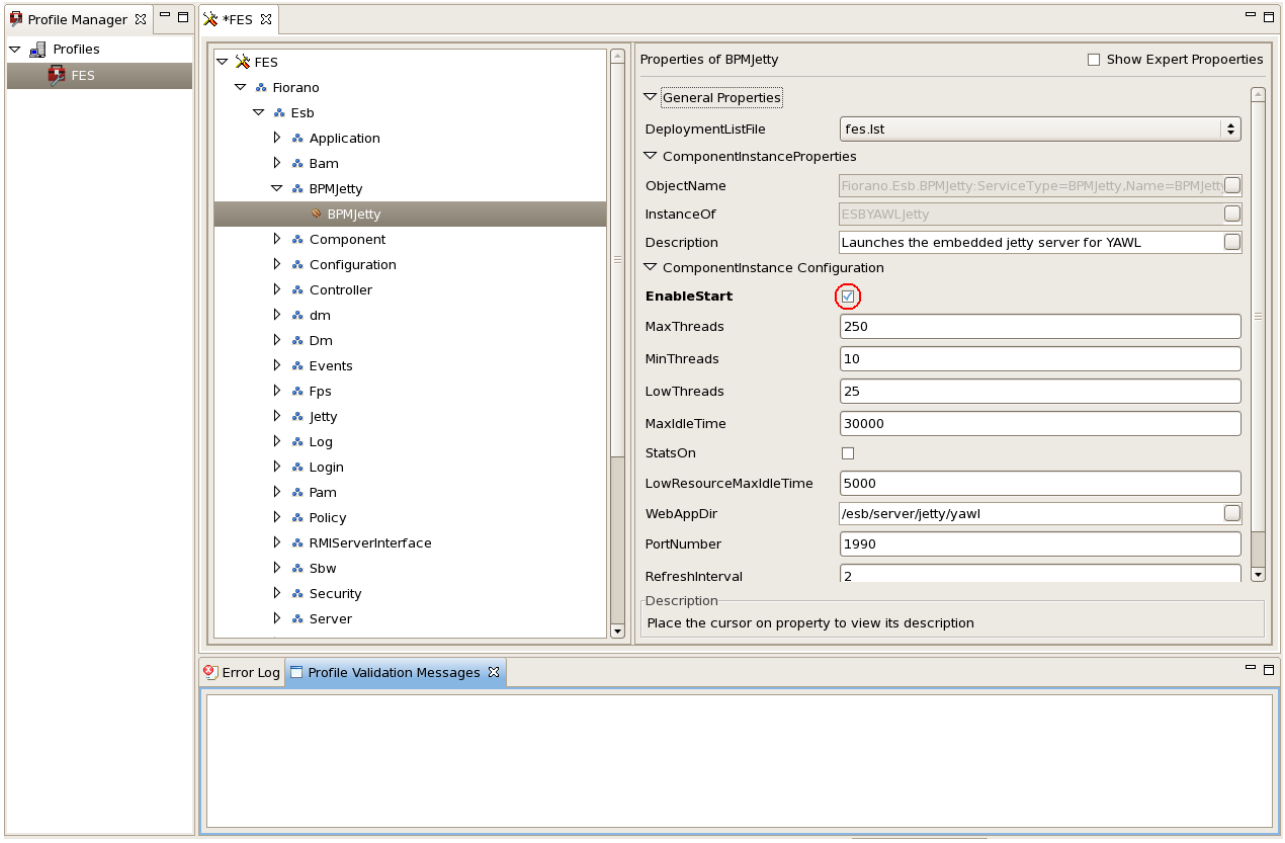


Figure 15.1: Enable BPM

Note: In order to start the BPM tool, you should run the enterprise server with JAVA 1.6 or above.

The BPM log files are present in ${FIORANO_HOME}/runtimedata/EnterpriseServers/${profile}/FES/run/logs/BPM folder.

15.2 Configuring DB

The BPM tool by default uses H2 database. H2 database files are created in ${FIORANO_HOME}/runtimedata/EnterpriseServers/${profile}/FES/yawl_db. You can change the database by changing the database configurations in FIORNAO_HOME/esb/server/jetty/yawl/webapps/*/WEB-INF/classes/hibernate.properties.

15.3 Resource Service

The resource perspective is responsible for modeling an organizational structure, and the people who populate it in a computational form, so that a person may be coupled with tasks and data emanating from the control-flow and data perspectives. The basic role of resource service is to allocate workitems to resources for processing.

To open the Resource service, open a web browser and provide the url http://localhost:1990/resourceService/faces/Login.jsp.

To login with admin privilege, use the following credentials: UserName: admin Password: YAWL.

15.4 BPM Editor

BPM editor is used to design the specifications and validate them.

To launch the BPM editor:

In Unix

Go to esb/tools/yawleditor/ and run ./yawleditor.sh

In windows

Click start menu-->Fiorano-->FioranoSOAX.X.X-->Fiorano Tools and click on BPMEditor

Or

Open command prompt, goto %FIORANO_HOME%/esb/tools/yawleditor and run yawleditor.bat.

After Launching the editor, test the Engine Connection and Resource Service Connection.

To test the Engine Connection, go to settings-->Engine Connection, provide YAWL Engine URL as http://localhost:1990/yawl/ia, User Name as admin and Password as YAWL.

To test the Resource Service Connection go to settings-->Resource Service Connection, provide Resource Service Connection as http://localhost:1990/yawl/ia, User Name as editor and Password as yEditor.

Chapter 16: Frequently Asked Questions

Question 1: In what scenarios can we best use these two components in a process?

Answer: LDAP authentication is used as a secure gateway into a flow. Once the incoming request is authenticated, using LDAP component, the security context can be passed on to other components of the flow, using application context.

Question 2: What is the need for the LDAP Authenticator component if, the LDAP lookup component also can perform authentication functionality? What is the purpose of this component?

Answer: There is a duplication of functionality that's why there is a need for the LDAP Authenticator component. The LDAP Authenticator code was created to demonstrate creation of a custom component. The source code for this component can be found here:**install_path>\esb\samples\components\EDBC**

Question 3: Can the LDAP Authenticator component be used in conjunction with the LDAP lookup?

Answer: Yes, LDAP Authenticator component can be used in conjunction with the LDAP lookup If, it is required.

Question 4: In the drop-down menu of the LDAP Lookup component Configuration, there are 3 options to select, "Authenticate", "Lookup" and "Bind". What is the difference between Authenticate and Bind?

Answer: Bind is used to add named objects (new users) into the LDAP repository. This may be used initially while populating the LDAP store with the user information.

Question 5: Once we authenticate a user, will that user's credentials roll over to other components in the process, in some global variable? Also if one of the Web Services Consumer components requires security Credentials, can the successfully authenticated credentials from the Authenticator be rolled over to the WS consumer?

Answer: Yes, this can be done by populating the security context in the application context.

Question 6: It was observed that every time the flow/servers were restarted, the size of admin.dat file, which was under FioranoSOA Install Dir\runtimedata\EnterpriseServers\FES\run, increased in size. After FES was running for a long period, the size may be in GB. Do we have any way to clear the admin.dat except restarting fes? What is the admin.dat for?

Answer: The admin.dat file stores runtime status information for the Event Processes and their service components, that is, whether the Event Process and its service components are running or not. Moreover, this file also stores other useful information, for example, the information about JMS destinations, connection factories, users and groups created in the server. The runtime status for Event Processes and their service components gets stored in this file whenever there is a change in state for an Event Process or its service components. This information is used by Enterprise server to restore the states of all the Event Processes whenever fes/fps server is restarted. Now, the size of admin.dat file increases because whenever new information is to be persisted into this file, old information is not deleted, instead it is marked for deletion for the next time when the Enterprise server restarts. This process of deleting the entries which have been marked for deletion is called as defragmentation process. This is done for minimizing disk-write operations and hence improves performance. Therefore, you can observe a decrease in the size of this file everytime Enterprise server restarts. From Fiorano SOA 2007 SP8 release, the users will be able to configure the size of this file (in bytes) at which defragmentation process will be triggered even while servers are running. The default file size is kept at 512 MB and is configurable by logging into FES-JMX through Studio and navigating to JMX Connection -> Fiorano -> jndi -> NamingManager-> FILE -> NativeFileNamingManager. In the MBean Explorer window, change the AdminObjectFileSize property to reflect the new value. The changes will be applied to the server. To persist the changes permanently for the running profile of the server, choose Save Configurations by right-clicking on FES-JMX node in Studio.

Question 7: User cannot connect via JMX to the server, when the server is behind a firewall.

Answer: While creating a JMX connection to servers, the JMX RMI connector opens two ports. One is for the RMI registry, which is fixed and known and other port is used to export JMX RMI connection objects. This second port is usually dynamically allocated at random. This causes problem if server is behind a firewall as one of the connection ports is random and it cannot be unblocked.

SOA 9.1.0 release provides a way to configure the value for this second port. To configure this property, open the profile and navigate to Fiorano->jmx->connector->RMIBasedJMXConnector and specify the desired value for 'RmiPortExportObjects' property.

Question 8: Server does not start up and I get the LifeCycleManager Error for a particular component on console. How do i get the stacktrace of the exception thrown to debug the issue?

Answer: Shutdown the server. Open the log4j.properties file located under $FIORANO_HOME/esb/server/profiles and change the level of fiorano.container Logger from ERROR to DEBUG, that is, log4j.logger.fiorano.container=ERROR, Console to log4j.logger.fiorano.container=DEBUG, Console and start the server up. The exception stacktrace will now be printed on console.

Question 9: When running multiple Publisher/Subscriber applications, while closing Subscribers we get the following exception:

"Unable to stop the consumer's delivery of messages to listener within specified timeout. :: Message delivery took longer then 30,000."

Answer: This is a harmless exception. This usually occurs when subscriber.close() is called and subscriber is not able to deliver the message within 30 seconds to the message listeners. Further there will not be any message loss because of this exception.

Chapter 17: Trouble Shooting

Hostname Resolution Issue

If the user gets the following exceptions while adding Breakpoint from eStudio:

javax.naming.NamingException: Lookup failed for object named :: PRIMARYQCF. Reason : fiorano.jms.common.ConnectionFailedException: Kernel I/O error occurred :: Failed_to_establish_connection_with_cause::_ {http://localhost.localdomain:1848;http://192.168.xxx.xxx:1847};

Check whether 'hostname -i' gives the correct in-use IP address of the host, if it does not, modify /etc/hosts file to correct it.

Peer on Multi-homed machine

If the Peer is started on a multi-homed machine, and launching EventProcesses is taking a lot of time, throwing timeout exceptions at times, then configure MQ address(working) for Peer using the following address:

Open FPS profile, goto Fiorano->socketAcceptors->ConnectionManager and set the ServerAddress property to the desired IP Address.

Firewall Issues

If a host running the server has a firewall, which only allows connections on some specific ports, the firewall has to be modified to allow connections from other ports.

If the host running the server is a multi-homed host and creating routes or breakpoints is causing problems, check if the connect URL of the server connection factories point to the IP address which is firewalled/barred from accepting connections.

- If creating routes causing issues,
    - Login to eStudio->ConnectionManagement->FES
    - Change connectURL of peer connection factories to the correct one if they point to a firewalled IP
- If creating breakpoints is causing issues,
    - Login to eStudio->ConnectionManagement->FES
    - Change connectURL of PRIMARYQCF, SERVICEPROVIDERQCF,CF,TCF to the correct one if they point to a firewalled IP

eStudio login issue

If login to a server on remote machine is extremely slow, seeming like it is hung, check if Windows firewall is on in the eStudio machine and disable it

File encoding issue

When working with locales other than English, it is recommended to set encoding property when working with mappings.

- Add -Dfile.encoding=UTF-8 in %FIORANO_HOME%/eStudio/eStudio.ini.
- eStudio->Window->Preferences->Fiorano->SOA Orchestration->CPS Launch (Tab)->System Properties, Add file.encoding as name and UTF-8 as Value.
- Add file.encoding=UTF-8 at <java.system.props> in %FIORANO_HOME%/esb/fes/bin/fes.conf. In case of multiple esb/fes servers, repeat this step for all esb/fes servers.
- Add file.encoding=UTF-8 at <java.system.props> in in %FIORANO_HOME%/esb/fps/bin/fps.conf. In case of multiple peer/fps servers, repeat this step for all peer/fps servers.

Note:

- If required, other encodings can also be used in place of UTF-8.
- If already multi-byte characters are used in mapper funclets (as constants), existing mappings or characters may get deleted or corrupted because of differences in encoding. Redrawing mappings is recommended in this scenario.

Client Connections in Peer

If the error "Cannot perform this operation, limit reached. :: Cannot create more connections as the connection count for this Connection manager has reached the upper limit of X connections", then the user needs to increase Client Connections in Peer.

Open FPS profile, goto Fiorano->socketAcceptors->ConnectionManager and set the MaxClientConnectionsCount property to a value higher than 1024.

File Handles in Linux

If the error "Too many files open" is encountered, the user needs to increase the system file handles.

Goto /etc/security/limits.conf, at the end, add these lines:

- soft nofile <noOfFileHandles>
- hard nofile <noOfFileHandles>

Problems When Non-ascii Chars Used in build.properties

If user is running CLI tools on Linux and the Applications/Services names contain non-ascii characters, the process can fail.

To resolve, we need to either convert the file into unicode,

```
native2ascii build.properties > newbuild.properties
```

or reconvert into ISO8859-1,

```
iconv -f UTF-8 -t ISO8859-1 build.properties > newbuild.properties.
```

and then use the new build.properties file for CLI tasks.

JRE Problems

If the following error

Error: no 'server' JVM at '<JRE PATH>\bin\server\jvm.dll'

is observed on running the Fiorano servers, if a JRE(at <JRE PATH>) other than one shipped, the user can either use corresponding JDK or copy the JDK's jre\bin\server folder to <JRE PATH>\bin\server directory.

For more information, please refer to the following link:

http://www.oracle.com/technetwork/java/javase/jrereadme-182762.html

HA Troubleshooting

1. SocketBindException saying that the HA Port is already bound.

   This exception means that some other program running on the HA port or the last instance of the server is not properly killed. You can choose to stop or kill the application which is holding up the port and start the server again or choose a different HA port. But changing this means that there needs to be a change in the Backup Servers' configuration for its Backup Server port.

2. None of the servers starting up. Both the servers are in WAITING state and the Primary Server is trying to connect to its Backup Server.

   This exception means that the Backup Server IP and port numbers are wrong for both the server configurations.

   Example: Server console when it cannot connect to the Backup Server.

Figure 1 illustrates a situation where the server is not able to connect to the Backup Server. If it is already connected, then there is a problem with the configuration. The message prints the IP address and the port to which it is trying to connect to establish the HA channel. You can check if the Backup Server is running in the printed IP address and port.

```
C:\WINDOWS\system32\cmd.exe - server -profile haprofile1/primary

C:\PROGRA~1\Fiorano\FMP90~2.0-B\esb\server\bin>server -profile haprofile1/primary

Fiorano Home      : C:\PROGRA~1\Fiorano\FMP90~2.0-B
Java Home         : C:\PROGRA~1\Fiorano\FMP90~2.0-B\jre1.5.0_16
Profile           : C:\PROGRA~1\Fiorano\FMP90~2.0-B\esb\server\profiles\haprofile1\primary

Server's Process ID : 2512
[07/Feb/2009 13:25:52]    license      INFO       The fiorano-soa9.lic license for the product FPS ver 9x is
RMIConnectorServer started at: service:jmx:rmi://localhost/jndi/fmq
RmiConnectorServer Listening Port: 2067
[Sat Feb 07 13:26:00 IST 2009]  Trying to establish connection with Unknown IP:1768 ...
[Sat Feb 07 13:26:00 IST 2009]  Primary Server switched to WAITING
Backup Server not Connected. Check configuration or Start backup Server ...
Profile ..\profiles\haprofile1/primary\FPS successfully deployed on Sat Feb 07 13:26:00 IST 2009
[Sat Feb 07 13:26:00 IST 2009]  Trying to establish connection with Unknown IP:1768 ...
[Sat Feb 07 13:26:18 IST 2009]  Trying to establish connection with Unknown IP:1768 ...
[Sat Feb 07 13:26:35 IST 2009]  Trying to establish connection with Unknown IP:1768 ...
[Sat Feb 07 13:26:52 IST 2009]  Trying to establish connection with Unknown IP:1768 ...
[Sat Feb 07 13:27:09 IST 2009]  Trying to establish connection with Unknown IP:1768 ...
[Sat Feb 07 13:27:27 IST 2009]  Trying to establish connection with Unknown IP:1768 ...
[Sat Feb 07 13:27:44 IST 2009]  Trying to establish connection with Unknown IP:1768 ...
```

Figure 1: Server unable to connect to the Backup Server

3. One of the HA Servers switched into Active or Passive Sync and it hangs there, but the other seems to be in WAITING state for a long time trying to connect to the Backup Server.

   This exception means that your configuration for the Backup Servers does not match at the end where the server is still in WAITING state, but the Backup Server is still able to connect. This will cause the Backup Server to hand indefinitely as it expects a Synchronization Complete Notification which is never going to be delivered.

```
C:\WINDOWS\system32\cmd.exe - server -profile haprofile1/secondary

C:\PROGRA~1\Fiorano\FMP90~2.0-B\esb\server\bin>server -profile haprofile1/secondary

Fiorano Home      : C:\PROGRA~1\Fiorano\FMP90~2.0-B
Java Home         : C:\PROGRA~1\Fiorano\FMP90~2.0-B\jre1.5.0_16
Profile           : C:\PROGRA~1\Fiorano\FMP90~2.0-B\esb\server\profiles\haprofile1\secondary

Server's Process ID : 3916
[07/Feb/2009 13:37:46]    license      INFO       The fiorano-soa9.lic license for the product FPS ver 9
 false, Node locked = false
RMIConnectorServer started at: service:jmx:rmi://localhost/jndi/fmq
RmiConnectorServer Listening Port: 2068
[Sat Feb 07 13:37:52 IST 2009]  Secondary Server switched to WAITING
Profile ..\profiles\haprofile1/secondary\FPS successfully deployed on Sat Feb 07 13:37:52 IST 2009
[Sat Feb 07 13:37:52 IST 2009]  Old status of remote server = DEAD
[Sat Feb 07 13:37:52 IST 2009]  New status of remote server = WAITING
[Sat Feb 07 13:37:52 IST 2009]  Old status of remote server = DEAD
[Sat Feb 07 13:37:52 IST 2009]  New status of remote server = WAITING
[Sat Feb 07 13:37:52 IST 2009]  Secondary Server switched to PASSIVE TRANSITION STATE
```

Figure 2: The server hanging in one of the synchronization states

4. Both servers go to Standalone/Active state in replicated/shared mode respectively if the network link between them is broken. This can happen if the servers do not refer to the same LockFile.
5. The server in replicated mode shuts down on boot up.

   This can happen if the LockFile specified is not valid or the machine hosting the LockFile is not allowing the server to acquire a lock.

   Figure 3 illustrates the server shutting down on boot up.

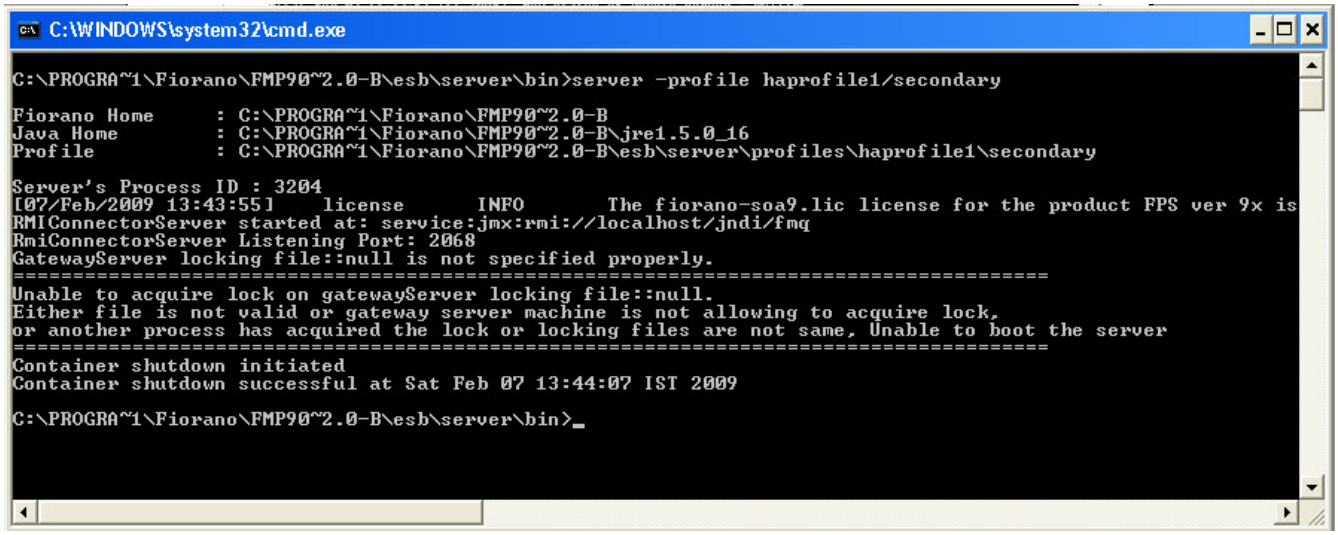


Figure 3: The server shutting down on boot up

Index

ACL, 1407

Add
- breakpoint, 56
- JMS Property, 1259
- log Modules for the component, 241
- new parameters to the component, 244
- new resource/dependency, 221
- new system library, 222, 229
- Node Name to a Component Instance, 246
- Peer Server, 109
- Ports for the component, 239
- resource to class path, 225, 229
- Runtime Arguments for the component, 242
- User to a Group, 88
- User XSLT, 1503

Additional Component Administration Features in the Fiorano Studio, 258

Aggregator, 725
- aggregator condition, 728
- completeness condition, 726
- completeness criteria source, 728
- correlation id, 731
- functional demonstration, 734
- message persistence, 732
- override message, 734
- timeout with override, 726
- xpath, 730, 732

alert
- create, 1574
- edit, 1577, 1585
- jms, 1575
- managing, 1573
- remove, 1577
- smtp, 1573

Application Context
- Configuring, 1238

application level
- changing permission, 1413
- permission, 1407

Assign Rights to Users and Groups, 93

Assigning Rights to Users and Groups, 91

Asynchronous Components, 208

Authentication, 1384

authentication mode
- HTTPAdapter, 1019

Authorization, 1385

Backlog policies
- managing, 1578, 1581

Basic Authentication
- Jetty Server, 151

Cache, 769
- add operation, 775
- configuration, 770
- delete operation, 779
- functional demonstration, 774
- input and output, 773
- lookup operation, 778
- output, 774
- output and input, 773
- remove entry, 772
- threshold, 771
- update operation, 776

CBR, 736
- configuration, 737
- functional demonstration, 741
- namespaces, 738
- processor, 739
- routing rules, 738
- schema, 737
- use case, 742
- xpath, 739
- xslt, 740

Clear
- All Mappings, 1510
- Data, 1510
- Input Structure, 1432
- Output Structure, 1438

Clearing ESB Server Database, 93

Clearing Peer Server Database, 114

Comments Tab, 1424

common component properties, 1525

common properties
- cache component, 1528
- connection error, 1550
- connection pool params, 1539

defined jms, 1538
deployment, 1527
error handling, 1549
execution, 1529
debug mode, 1530
debug port, 1531
input port properties, 1532
interaction configuration, 1544
invalid request, 1555
jms destination, 1532, 1537
destination name, 1533
destination type, 1532
use specified destination, 1533
log module instance, 1531
managed connection factory, 1539
messaging, 1534, 1537
acknowledgement mode, 1535
durable subscription, 1536
transacted, 1535
monitoring configuration, 1545
outport transport, 1549
output port properties, 1537
port properties, 1532
proxy server, 1540
proxy settings, 1540
runtime arguments, 1531
scheduler configuration, 1546
schema editor, 1557
schema repository, 1562
ssl security, 1542
transport configuration, 1548
version locked, 1528
xpath editor, 1564

Communicating with ESB Server, 64

Component
Configuration, 213
Creation, 266
Creation from the Fiorano Studio, 289
Dependencies and System Libraries, 218
Deployment, 235
Launch Semantics, 209

Component and Component Instances, 207

Compose
Event Process, 50

configuration
XMLSplitter, 758

configuration
Exception Listener, **677**
File Writer, 699
Join, 745

configuration
MQSeriesIn, 848

configuration
MQSeriesOut, 878

configuration
XSLT, 969

configuration
HTTPAdapter, 1018

configuration
HTTP Stub, 1064

configuration
HTTP Stub
fes connection, 1064

configuration
Simple HTTP, 1075

Configure and run Business Components, 251

Configuring
Application Context, 1238
Document Tracking, 1231
event tracking, 184
Logging Parameters, 232
Servers, 143
specific database, 186
Specific Database, 1231
Tools, 143
transport, 1570

Configuring Jetty Server, 151

Configuring Mapper Settings, 1511

Configuring SSL support, 148

connection error
common properties, 1550

connection mode
HTTP Receive, 1044

Copying functions in a Mapping, 1510

Create
a New Group, 88
a New User Account, 85
Mappings, 1488
New system libraries, 228

Creating
alerts, 1574
jms alert, 1577
policy, 1580, 1583

creating queues, 848, 878

Customizing the Mapper User Interface, 1513

Database
configuring, 186

database configuration
DBQueryOnInput, **635**

DB Adapter Tuning, 1343

DBProc, 620
advanced settings, **624**
connection properties, **623**
database configuration, **621**
functional demonstration, **633**
input schema, **631**
interaction configuration, **624**
JDBC, **622**, **624**
output schema, **632**
sp configuration, **625**
wrap DB object, **624**

DBQueryOnInput, **633**
advanced settings, **637**
database configuration, **635**
functional demonstration, **642**
input, **641**
interaction configuration, **638**
JDBC, **636**
JDBC driver, **637**
output, **642**
single batch mode, **641**
wrap db object, **637**

Default Users, Groups and ACLs
LDAP Security Realm, 1399
RDBMS Realm, 1396
XML Security Realm, 1404

Define Mappings
Using the Automatic Mapping option, 1495
Using the Visual Expression Builder, 1496

Defining Components, 271

defining jms message xsl, 975

Delete
Group, 90
User from a Group, 89

Deploying
component, 288
Event Process, 55
JCA Component in an External JCA container, 255

deployment
common properties, 1527

Deployment Manager, 50, 1389

Details Pane, 1424

Developing and Adding New Components, 59

Different Topologies, 64

Displaying All Mappings, 1509

Distribute components, 1341

Document Tracking, 1230
Configuring, 1231

Duplicating For-Each Mapping, 1493

Dynamic Change Support, 1339

Enabling Basic Authentication
bswsgateway, 152
wsstub, 152

error action map
HTTP Receive, 1053

error handling
common properties, 1549

Error Messages Panel, 1427

ESB Component and Process Repository, 63

ESB Peer
Installation Steps, 66
System Requirements, 66

ESB Server
Installation Steps, 66
System Requirement, 65

ESB Server to Peer Server communication, 63

Event
tracking, 184

Event Ports Information Panel, 278

Event Process
Debugging, 56
Deploying, 55
security, 1407

Event Processes, 1223

Event tracking
configuring, 184

Exception Handling, 58

Exception Listener, **676**
backup server url, **679**
configuration, **677**
connection to enterprise server, **681**
functional demonstration, **681**
use case, **683**

execution
common properties, 1529

execution configuration
HTTP Stub, 1065

Existing XML Input Structure, 1428

Export a Component, 264

Export and Import Components, 264

Exporting Mappings to a File, 1508

Figure
Adding Breaking Point, 56
Adding Components to Palette, 60
CRM Logs, 59
Exception Handling, 58
Fiorano Network, 68
Fiorano Services and Security Manager, 61
Fiorano System Architecture, 62
HA on ES, 60
HA on Peer, 61
Inport Properties, 52
Installation Topology, 65
Log Manager, 58
Out Exception Properties, 52
Server Explorer, 55
Service Component, 51
Shutting Down FES, 72
Tracking Document, 57

file encoding
File Writer, 706

File Reader, **684**
chunk size, **685**
file encoding, **688**
functional demonstration, 697
input, 696
output, 697
postprocessing actions, 690
preprocessing command, **688**
testing, 697
use case, 698

file timeout
File Writer, 705

File Writer, 698
configuration, 699
file encoding, 706
file timeout, 705
functional demonstration, 713
input, 711
output, 711
output mode, 704
post processing command, 707
sample scenario, 709
target directory, 702
testing, 712
timestamp format, 703
use case, 714

Fiorano
ESB Server, 63
Peer Server, 63

Fiorano Environment, 62

Fiorano ESB Server
Configuration, 73
Configuration Steps, 73
External Ports, 73
Functionality, 69
Internal Ports, 78
Offline Mode, 73
Online Mode, 76
Server Port Configuration, 73

Fiorano Event Manager, 50

Fiorano Peer Server
Configuration, 99
Functionality, 95

Fiorano Processes
Managing, 55

Fiorano Server, 49
Enterprise Server, 49
Peer Server, 49

Fiorano Service and Security Manager, 50

Fiorano Studio, 50

Fiorano System Architecture, 62

Fiorano Tools, 50

For-Each Mapping, 1492

FTPGet
advanced settings, 425
client authentication type, 421
connect mode, 425
connection pool params, 422
debug responses, 427
functional demonstration, 445

input schema, 443
inteaction configuration panel, 429
key type, 421
Managed Coonection Factory, 419
monitor progress interval, 433
Monitor settings, 433
output schema, 444
port, 421
private key file, 421
protocol, 420
proxy settings, 423
renote host, 421
site command, 427
source settings, 429
ssl security, 424
target settings, 436
transfer type, 426
use case scenario, 448
validate input, 433

FTPPut
target settings, 465

FTPPut, 449
advanced settings, 455
client autentication type, 451
connect mode, 455
connection pool params, 452
interaction configurations, 460
key type, 452
managed connection factory, 450
monitor settings, 462
private key file, 452
protocol, 451
proxy settings, 453
resume type, 456
site command, 457
source settings, 460
ssl security, 454
testing connection, 459
transfer type, 456

FTPPut
input and output, 466

FTPPut
functional demonstration, 471

FTPPut
use case, 474

Funclet Easel, 1484

Funclet Tab, 1425

Function Palette, 1439

functional demonstration
XMLSplitter, 764
XSLT, 980

functional demonstration
DBProc, **633**
DBQueryOnInput, **642**
Exception Listener, **681**
File Reader, 697
File Writer, 713
Join, 751
POP3, 490
SMTP, 510

functional demonstration
XMLVerification, 768

functional demonstration
Cache, 774

functional demonstration
MQSeriesIn, 874

functional demonstration
MQSeriesOut, 903

functional demonstration
HTTPAdapter, 1040

functional demonstration
HTTP Receive, 1061

functional demonstration
HTTP Stub, 1072

functional demonstration
Simple HTTP, 1077

functional demonstration
WS Stub, 1165

Functions
Advanced, 1450
Arithmetic, 1439
Boolean, 1467
Control, 1447
Conversion, 1448
Date-Time, 1452
JMS Message Functions, 1481
Lookup, 1479
Math, 1442
NodeSet, 1463
SQL, 1459
String, 1444
User Defined, 1482

Generate
Code for the Defined Component, 287
manual launch script from studio, 248

global level

changing permission, 1411
permission, 1408

High Availability, 60

HL7
Receiver, 512
Sender, 516

HTTP Receive, 1043
advanced properties, 1050
connection mode, 1044
error action map, 1053
functional demonstration, 1061
input, 1056
output, 1058
request parsing, 1045
headers, 1048
parameters, 1045
post data, 1047
response generation, 1049
use case, 1062

HTTP Stub, 1063
configuration, 1064
error details, 1069
execution configuration, 1065
fes connection configuration, 1064
functional demonstration, 1072
input, 1070
output, 1070
request details, 1066
parameters, 1066
post data, 1066
response details, 1068
use case, 1074

HTTPAdapter, 1017
authentication mode, 1019
configuration, 1018
content type, 1024, 1027
functional demonstration, 1040
http response property, 1026
input, 1029
interaction configuration, 1020
output, 1036
parameter details, 1024
post data details, 1024
proxy settings, 1019
response code, 1026
use case, 1042

IBM WebSphere, 848, 878

Import
Component, 265
Project from the File, 1509

input
DBQueryOnInput, **641**
File Reader, 696
File Writer, 711
HTTP Receive, 1056
HTTP Stub, 1070
HTTPAdapter, 1029
MQSeriesIn, 868
MQSeriesOut, 897
SMTP, 508

input port properties
common properties, 1532

Input Structure Panel, 1423

Installation, 64
Enterprise Edition, 64
Workstation Edition, 64

interaction configuration
HTTPAdapter, 1020

Inter-Connect flows, 1342

invalid request
common properties, 1555

JDBC
DBQueryOnInput, **636**

Jetty
basic authentication, 151
configuring server, 151
SSL configuration, 147

jms, 1532
alert, 1575
create alert, 1577
transport, 1572

jms message xsl
define, 975

JMS property
adding, 1259

Join, 744
configuration, 745
functional demonstration, 751
saxon, 746
xalan, 746

JVM Parameters, 1340

Kill a running component instance, 211

Labels, 1389

Launch
Component in a running application, 211

Components Using the Fiorano Studio, 210
Fiorano Peer Server, 95
Fiorano Peer Server from Fiorano Studio, 95
Fiorano Text Schema Editor, 374
The CPS, 213

Launching Fiorano ESB Server, 70
from Fiorano Studio, 70
from Script Files, 70

Lines Panel, 1423

Linking Nodes to Define Mappings, 1495

List of pre-built components, 408

Load Balancing, 1336

Loading
CSV Output Structure, 1435
Existing XML Input Structure, 1428
Input Structure, 1428
New Input XML Structure, 1431
Output Structure, 1433
XML Output Structure, 1434

log level
offline server setting, 171

log levels
changing running server, 173

log module instance
common properties, 1531

Logging, 58

Logging and Document Tracking Panel, 284

manage connection
MQSeriesOut, 883

manage connection:, 853

Managing
alerts, 1573
backlog policy, 1578
Fiorano Processes, 55
Mappings, 1508
policy execution, 1584
Users, 84
XSLT Properties, 1512

Mapper, 1417
autoMap menu, 1420
edit menu, 1419
environment, 1417
file menu, 1419
help menu, 1420
map view, 1422
menu bar, 1419
structure menu, 1419
toolbar, 1420
tools menu, 1420
view menu, 1419

mapping
XSLT, 972

Mapping types, 1491

Mappings
Tab, 1425
XML Formats, 1499
XML Formats to CSV Files, 1499
XML Formats to RDBMS Queries, 1499
XML Formats to RDBMS-Delete Queries, 1502
XML Formats to RDBMS-Insert Queries, 1499
XML Formats to RDBMS-Update Queries, 1500

Memory Configurations, 83

Memory Optimization, 1340

Messages Tab, 1426

messaging
common properties, 1534

MetaData View, 1427

Modify a saved configuration XML file, 254

Modifying the RDBMS Output Structure Settings, 1511

monitoring configuration
common properties, 1545

MQSeriesIn, 848
CCSID, 866
configuration, 848
defining MQMD headers, 861
destination queue, 855
functional demonstration, 874
host address, 853
input, 868
interaction configuration, 855
managed connection, 853
MQMD headers, 862
output, 873
port for MQSeries server, 854
receiveexitclass, 855
RFH2 header, 864
securityexitclass, 855
sendexitclass, 854

MQSeriesOut
CCSID, 894
configuration, 878
defining MQMD headers, 889
functional demonstration, 903
input, 897
interaction configuration, 886
managed connection, 883
message selection properties, 893
monitored queue, 886
MQMD headers, 891
output, 897
output mode, 887
port for MQSeries server, 884
queue name, 887
receive exit class, 885
security exit class, 885
send exit class, 885
sync point control, 892

MQSeriesOut, 878

Name-to-Name Mapping, 1491

Network Administrator, 50

New
Input XML Structure, 1431

New Group
Configuration, 88

Node
Destination Node, 1484

Node
Source Node, 1484

Node types, 1488

NodeInfo Tab, 1424

output
DBQueryOnInput, **642**
File Writer, 711
HTTP Receive, 1058
HTTP Stub, 1070
HTTPAdapter, 1036
MQSeriesIn, 873
MQSeriesOut
SMTP, 510

output mode
File Writer, 704

output port properties
common properties, 1537

Output Structure Panel, 1424

Parallel Data Flow, 1339

peer server
adding, 109

permission
application level, 1407
changing application level, 1413
changing defaults, 1411
changing global, 1411
global level, 1408
system level, 1408

policy
create, 1580

policy execution, 1584

POP3, 479
functional demonstration, 490
input schema, 488
interaction configuration, 483
output schema, 488
sample
input and output, 487

Port
Adding a Port, 279
Modifying details of Port, 279
Removing a Port, 279

port properties
common properties, 1532

post processing command
File Writer, 707

Pre-built Components, 408

Precedence, 1390

properties
common, 1525
common component, 1525

proxy settings
HTTPAdapter, 1019

RDBMS Realm, 1396
User Management, 1396

Removing Mappings for a Node, 1509

Removing Network Rights for Users and Groups, 93

Repository Location, 183

RMI Server Ports, 104

RMI Server Ports Configuration
Offline Mode, 104
Online Mode, 105

Rule syntax, 1391

Rules, 1390

Running Components from Outside the Studio, 247

running server
    changing log levels, 173

runtime arguments
    common properties, 1531

Sample Rule, 1391

sample scenario
    File Writer, 709

saxon
    XSLT, 978

Scalability, 1339

Scalability and Load Balancing, 1336

scheduler configuration
    common properties, 1546

schema editor
    common properties, 1557

schema repository
    common properties, 1562

security
    event process, 1407

Security, 61, 1384

Separate machines for Servers, 1341

Server Ports Configuration, 99
    Offline Mode, 99
    Online Mode, 102

Service Component Characteristics, Configuration, and Deployment, 209

Service Components Characteristics, 207

setting
    offline server log level, 171

Setting Access Control to Users and Groups, 91

Setting Component Launch Type in the Studio, 210

Setting Up Users and Groups, 84

Settings
    mapping, 1511

Simple HTTP, 1075
    configuration, 1075
    functional demonstration, 1077

Size of Event Flows, 1342

Size of Messages, 1342

smtp
    alert, 1573
    transport, 1571

SMTP, 502
    authentication details, 504
    functional demonstration, 510
    input, 508
    interaction configuration, 507
    output, 510
    sender information, 507
    timeout settings, 504
    use case, 511

SSL configuration
    Jetty, 147

ssl security
    common properties, 1542

Stopping
    Fiorano ESB Server, 71
        from Fiorano Studio, 71
        from Script Files, 72
    Fiorano Peer Server, 97
    Fiorano Peer Server from Fiorano Studio, 97

Synchronous Components, 207

system level
    permission, 1408

Template Engine, 267

testing
    File Writer, 712

Testing Transformation, 1505

Testing Web Service, 149, 152, 154

Thread Count of Components, 1338

timestamp format
    File Writer, 703

tracking
    events, 184

Tracking Document, 57

Transparent Resource Addition, 1339

transport
    configure, 1570
    jms, 1572
    smtp, 1571

transport configuration
common properties, 1548

use case
XMLSplitter, 766

use case
Exception Listener, **683**
File Reader, 698
File Writer, 714
SMTP, 511

use case
XSLT, 982

use case
HTTPAdapter, 1042

use case
HTTP Receive, 1062

use case
HTTP Stub, 1074

Use Case, 1337

User Management
LDAP Security Realm, 1399
RDBMS Realm, 1396

Using Script Files, 96

Using the Variables in Template, 270

Validating All Mappings, 1509

Variables Defined in the fiorano Setting, 269

View the resources of a component, 219

Viewing
Output Structure Source, 1438
Source of Input Structure, 1432

Web Service Consumer 4.0, 1169

Web Service Consumer 5.0, 1188

working with ACL, 1407

Working with the Visual Expression Builder, 1439

WS Stub
functional demonstration, 1165
realms, 1096, 1158

xalan
XSLT, 978

XMLSplitter
configuration, 758
functional demonstration, 764
namespaces, 760
schema, 759
use case, 766
xpath, 761

XMLSplitter, 758

XMLVerification
functional demonstration, 768
xsd structures, 767

XMLVerification, 767

Xms, 1340

Xmx, 1340

xpath editor
common properties, 1564

xsl
XSLT, 974

XSLT, 969
configuration, 969
engine, 977
fail transformation error, 979
functional demonstration, 980
jms message xsl, 974
mapping, 972
optimization, 980
saxon, 978
strip white spaces, 978
transformer factory class, 978
use case, 982
xalan, 978
xsl, 974

Xss, 1340