



Fiorano
Peer-to-Peer Dataflow Pipelines™

www.fiorano.com

Fiorano eStudio®

User Guide

AMERICA'S

Fiorano Software, Inc.
718 University Avenue Suite
212, Los Gatos,
CA 95032 USA
Tel: +1 408 354 3210
Fax: +1 408 354 0846
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

Fiorano

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 sublicense, 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% support.

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) 2008-2011, Fiorano Software Pte Ltd. and Affiliates

Contents

Chapter 1: Introduction to Fiorano eStudio..... 24

1.1 Key Features.....	24
1.2 Getting started with Fiorano eStudio.....	25
1.2.1 Launching eStudio using 64-bit JVM	26
1.2.1.1 To install 64bit jars.....	26
1.2.1.2 To uninstall 64bit jars	27

Chapter 2: Offline Event Process Development 28

2.1 Fiorano Views.....	29
2.1.1 Event Process Repository View	29
2.1.2 Fiorano Orchestration.....	29
2.1.3 Service Palette	30
2.1.4 Properties.....	31
2.1.5 Problems.....	31
2.1.6 Error Log.....	32
2.1.7 Service Repository (Offline)	32
2.1.8 Project Explorer	33
2.1.9 Service Descriptor Editor	34
2.2 Event Processes	35
2.2.1 Creating New Event Process.....	35
2.2.2 Opening Sample Event Process.....	36
2.2.3 Import and Export Event Processes	38
2.2.3.1 Exporting an Event Process.....	38
2.2.3.2 Importing an Event Process	41
2.2.4 Importing nStudio Event Processes	43
2.3 Service Repository (Offline Event Process Development)	45
2.3.1 Deploying Services to Server.....	45
2.3.2 Fetching Services from Server	46
2.3.3 Exporting Services to Local Disk	47
2.3.4 Importing Services from Local disk.....	48

Chapter 3: Online Event Process Development Perspective 50

3.1 Fiorano Views.....	52
3.1.1 Server Explorer	52
3.1.2 Fiorano Debugger View	53
3.2 Service Repository (Online Event Process Development)	53

3.2.1 Exporting Services to Local Disk	54
3.2.2 Importing Services from Local disk.....	55

Chapter 4: Mapper Perspective 56

Chapter 5: Composing Event Processes 57

5.1 Adding Components	57
5.2 Connecting Routes	58
5.3 Configuring Components.....	58
5.4 Configuring Component Properties	60
5.5 Adding Remote Service Instance.....	63
5.6 Adding External Event Process (Subflow)	65
5.7 Document Tracking.....	67
5.8 Defining Route Transformations	69
5.9 Configuring Selectors on Routes.....	72
5.10 Configuring Application Context	73
5.11 Check Resource and Connectivity	75
5.12 Running Event Process.....	76
5.13 Stopping an Event Process	76
5.14 Synchronizing an Event Process	77

Chapter 6: Event Process Life Cycle Management 78

6.1 Setting Properties of Service Instances for Different Environments.....	78
6.2 Running Event Process on an Environment	79

Chapter 7: Debugging Event Process 80

7.1 Adding Breakpoint	80
7.1.1 Context Menu option.....	80
7.1.2 Debugger View.....	81
7.2 Viewing Messages at Breakpoint	82
7.3 Editing Messages at Breakpoint.....	82
7.4 Inserting Messages into Breakpoint	83
7.5 Releasing Messages from Breakpoint	84
7.6 Discard Messages from Breakpoint	85
7.7 Remove Breakpoint.....	86

Chapter 8: Services 87

8.1 Service Descriptor Editor	87
8.1.1 Overview Section.....	89

8.1.2 Execution Section	90
8.1.2.1 Port Information.....	90
8.1.2.2 Support	91
8.1.2.3 Launch Configuration	92
8.1.2.4 Log Modules	92
8.1.2.4 Runtime.....	92
8.1.3 Deployment Section.....	93
8.1.3.1 Resource.....	94
8.1.3.2 Service Dependencies	94

Chapter 9: Service Creation..... 95

9.1 Service Generation	95
9.1.1 Service Location	95
9.1.2 Basic Details	96
9.1.3 Ports Information	97
9.1.4 Resources	98
9.1.5 Dependencies	99
9.2 Building and Deploying Services.....	99

Chapter 10: eMapper 101

10.1 Key Features of Fiorano eMapper	101
10.2 Fiorano eMapper Environment.....	101
10.2.1 eMapper Projects	102
10.2.2 eMapper Editor	103
10.2.2.1 Map View	103
10.2.2.2 MetaData tab	104
10.2.3 Funclet View	104
10.2.4 eMapper Console	105
10.2.5 MetaData Messages View	105
10.2.6 Node Info View.....	105
10.3 Working with Input and Output Structures.....	106
10.3.1 Loading Input/Output Structure	106
10.3.1.1 Load Input/Output Structure From an XSD document	106
10.3.1.2 Load Input/Output Structure from a DTD document.....	109
10.3.1.3 Load Input/Output Structure from an XML document	109
10.3.1.3 Load a FIX message as an Input/Output Structure.....	110
10.3.2 Delete Structure	111
10.3.3 Edit Structure.....	111
10.4 Working with the Visual Expression Builder	112
10.4.1 Function Palette.....	113
10.4.1.2 Math Functions.....	115
10.4.1.3 String Functions	118
10.4.1.4 Control Function	120

10.4.1.5 Conversion Functions	121
10.4.1.6 Advanced Functions	123
10.4.1.7 Date-Time Functions	125
10.4.1.8 NodeSet Functions	131
10.4.1.9 Boolean functions	133
10.4.1.10 Lookup functions	144
10.4.1.11 JMS Message Functions	146
10.4.1.12 User Defined functions	147
10.4.2 Funclet Easel	148
10.4.2.1 Source Node	149
10.4.2.2 Destination Node	149
10.5 Creating Mappings	152
10.5.1 Understanding Types of Nodes	152
10.5.2 Types of Mappings	154
10.5.2.1 Name-to-Name Mapping	154
10.5.2.2 For-Each Mapping	155
10.5.3 Duplicating a For-Each Mapping	156
10.5.4 Linking Nodes to Define Mappings	158
10.5.4.1 Using the Automatic Mapping option to Define Mappings	158
10.5.4.2 Using the Visual Expression Builder to Define Mappings	159
10.5.5 Mapping XML Formats	162
10.6 Adding User XSLT	162
10.7 Working with derived types	165
10.8 Create/Edit User Defined Function(s)	167
10.9 Testing the Transformation	170
10.10 Managing Mappings	175
10.10.1 Exporting eMapper Project	175
10.10.2 Importing Project from the File	175
10.10.3 Copying functions in a Mapping	176
10.10.4 Clearing All Mappings	176
10.10.5 Managing XSLT Properties	176
10.11 FIX-XML Transformations	178
10.11.1 Loading FIX message in Mapper	178
10.11.2 XSL generation	179
10.11.3 Test Mappings	179
10.11.4 Mapping XML to FIX	179
10.11.5 Defining FIX to XML or XML to FIX transformation on a Route	180

Chapter 11: Working With Multiple Servers And Perspectives..... 181

11.1 Active Server Node	181
11.2 Switching of Active Server	182
11.3 Switching Between Perspectives	185

Chapter 12: Fiorano Preferences 187

12.1 ESB Connection Preferences	187
12.2 SOA Orchestration	188
12.2.1 General Options.....	188
12.2.2 Workflow Options	188
12.2.3 Service Options	188
12.2.3.1 Default JVM Configurations	189
12.2.3.2 Connection Factory Preferences	190
12.2.4 CPS Options.....	190
12.3 SOA Orchestration Online.....	192
12.3.1 General Options.....	192
12.3.2 Application Options	193
12.3.3 Service Options	193
12.3.4 Peer Options	193
12.4 Key Board Short Cut Preferences.....	194

Chapter 13: Schema Repository 197

Chapter 14: SCM Integration..... 200

14.1 Downloading and integrating SVN into eStudio	200
14.2 Using SVN with eStudio	202
14.3 Checking out and Updating Event Processes.....	205

Chapter 15: Named Configurations 209

15.1 Connection Factory	210
15.2 Port	211
15.3 Resource	212
15.4 Route.....	215
15.5 Runtime Arguments	216
15.6 Selector.....	218
15.7 Service.....	218
15.8 Transformation	221
15.9 Workflow	223
15.10 Context Menu Options.....	224
15.11 Points to Note.....	229

Chapter 16: Connection Management..... 233

16.1 Admin Connection Management	234
16.2 JMX Connection Management.....	239

Chapter 17: Profile Management 245

17.1 Profile Management Perspective	245
17.1.1 Profile Manager View	245
17.1.2 Profile Editor	246
17.1.2.1 Component Tree	247
17.1.2.2 Properties UI	248
17.2 Using a Profile Editor to modify a Fiorano Server Profile	248
17.2.1 Modify Component Instance Properties	248
17.2.1.1 General Properties	248
17.2.1.2 Component Instance Properties	249
17.2.1.3 Component Instance Configuration	249
17.2.1.4 Component Instance Attributes	250
17.2.2 Add / Remove Component Instances	251
17.2.3 Resolving Component Dependencies	253
17.2.3.1 Properties of a Component dependency	253
17.2.3.2 Resolving Component Dependencies	254
17.2.4 Add a new Variable Dependency Instance	255
17.2.5 Change Implementation of a Component Instance	256
17.2.6 Adding attributes to a Component Instance	257
17.2.7 Remove custom attributes from a Component Instance	258
17.2.8 Edit Default Configuration for a Component Instance	259
17.2.9 Modify a domain	259
17.2.10 Edit Profile Metadata	260
17.2.11 Locate a Dependency Instance in the Component Tree	260
17.2.12 Find Usages of a Component Instance	260
17.3 Saving the Changes from Profile Editor	261
17.4 Saving a Profile to a Different Location	262
17.5 Managing Profiles from the Profile Manager View	262
17.5.1 Import / Export profiles	262
17.5.2 Validate Profile	262
17.5.3 Reload Profile	262
17.5.4 Close a Profile	262
17.5.5 Delete a Profile	263
17.5.6 Points to note	263

Chapter 18: Installing eStudio Dropins In Eclipse 264

18.1 Installation	264
18.2 Uninstallation	265

Chapter 19: Flat File Schema Editor..... 266

19.1 Text Format Layout Concepts	267
19.2 Creating Flat File Projects	268

19.3 Testing Flat File Schema	270
19.4 Generating Flat File Schema using sample data	271
19.5 Sample Schemas	274
19.6 Points to note	275
19.7 Flat File Element Properties	276
19.7.1 Schema Node Properties	276
19.7.2 Record Node Properties	279
19.7.3 Field Node Properties	281

Chapter 20: Fiorano Tools Perspective 283

20.1 Fiorano Tools Perspective	283
20.1.1 Fiorano Tools View	283
20.2 Deployment Manager	284
20.2.1 Using Fiorano Deployment Manager	284
20.2.1.1 Creating a Rule	285
20.2.1.2 Modifying a Rule	290
20.2.1.3 Creating a copy of a Rule	290
20.2.1.4 Deleting a Rule:	290
20.2.1.5 Testing the Rule[s]	290
20.2.1.6 Refreshing the display of Rules	291
20.2.1.7 Changing the precedence of Rules:	291
20.3 Event Manager	291
20.3.1 Managing Services	293
20.3.2 Managing Events	294
20.3.3 Managing Document Tracking	295
20.4 License Manager	298
20.4.1 Fiorano Licenses Overview	298
20.4.2 Tool Environment	299
20.4.3 License Configuration	300
20.4.4 Managing Fiorano Licenses	300

Chapter 1: Introduction to Fiorano eStudio

1.1 Key Features

This section outlines some of the key new features added to the Fiorano eStudio:

1. Offline Event Process Development

In Offline Event Process Development mode, Event Processes development is done without connecting to a server. The Offline perspective maintains its own repository of event processes and services. Event Processes can be developed in Offline mode and can be deployed to any Enterprise Server. A server connection is required only while deploying an Event Process.

2. EPLCM (Event Process Life Cycle Management)

EPLCM allows a user to move Event Processes in different labeled environments that is; Testing, Staging, QA, and Production, all at the click of a button. Pre-created profiles for each environment are automatically picked up by the Server at the deployment time. This allows the user to specify properties for service instances in an Event Process for multiple environments, rather than creating new event processes for each environment. With the new EPLCM functionality, migration from one environment to another is simple.

3. Sub-Flows

A powerful new Sub-flow concept has been added. Sub-flow allows the user to insert an event process into another event process, easing composition of large applications.

4. Improved Debugger Implementation

Message injection is added, together with a better set of views to simplify debugging.

5. Split File Development for Services and Application

The ServiceDescriptor.xml and Application.xml are changed to split files, thereby making them more readable and reducing the memory footprint of eStudio.

To reduce the memory footprints, internally the application object now contains just details of service instances while no longer holding any information of their configurations and schemas associated. Configurations and schemas are now picked up on demand.

6. Service Descriptor Editor

The editor edits the ServiceDescriptor.xml file, making the editing easier to perform than when using a Text/XML editor.

7. Quicker Custom Property Sheet (CPS) launch

The CPS, when associated with a given component now launches significantly faster than previous versions of the Studio.

The Save and Close options have been introduced in the CPS, allowing the user to save the CPS in the middle of configuration and revisit it at a later point of time.

8. Dynamic Validations while Editing and Creating Services and Applications

Dynamic Validations point out errors at development time, while Event Processes are being composed, or Services created; errors that had to previously wait until compilation or run-time can now be detected earlier in the development/composition cycle.

9. UI crafted for Rich User Experience

Significant user feedback has been incorporated within eStudio to provide a richer user-experience. Most common operations can now be performed with a single click and with much less navigation than in previous versions.

10. Support for Version Control Systems

Users can now store applications in any Version Control System (SVN, CVS, or VSS) using Fiorano eStudio.

11. The New Mapping Tool: eMapper

The eStudio incorporates a brand new mapping tool that is developed ground-up in Eclipse. This new version fixes many more bugs as compared to past versions and has several other enhancements.

12. Customization Possible as an Advantage of Eclipse Based Product

Since eStudio is developed over the Eclipse platform, users can now write their own plug-ins or use existing ones. Users are now able to customize the eStudio the way they want. For instance, a user can add a version control plug-in.

1.2 Getting started with Fiorano eStudio

To start Fiorano eStudio:

1. Navigate to **\$FIORANO_HOME/eStudio** and run the **eStudio** executable file.
2. Workspace Selection dialog is shown prompting for the workspace directory. Workspace is a directory where all the repositories (Event Processes, Services and other metadata) are stored.
3. The default workspace is set to **\$FIORANO_HOME/runtimedata/eStudio/workspace**. It is recommended to use the default workspace, but the user can change the workspace if required. The `Remember workspace` option can be selected to save the workspace used and not to show the dialog next time eStudio is launched.

Note: The workspace preferences are stored at `FIORANO_HOME/runtimedata/eStudio/WSprefs.properties`

The following preferences are stored in workspace preferences: `wsLastUsedWorkspaces`, `wsRemember` and `wsRootDir`.

If the user chooses a workspace and selects the `Remember workspace` option, and, if later, the Workspace Selection dialog has to be shown, then this can be done by changing the value of `wsRemember` to `false` in the workspace preferences.

When the Fiorano eStudio has completely launched, the user can switch between different workspaces. The option to switch the workspace is present at File -> Switch Workspace.

The current workspace selected is shown in Fiorano eStudio title bar.

4. By default, eStudio is launched in Offline Event Process Development Perspective mode and the offline repository is populated when eStudio is launched for the first time.
5. In Case, eStudio does not load properly, install XULRunner on your machine. Follow the guide lines from:
https://developer.mozilla.org/en/Getting_started_with_XULRunner to install and add the following:

```
-Dorg.eclipse.swt.browser.XULRunnerPath=$XULRunnerHome/xulrunner to  
$FIORANO_HOME/eStudio/eStudio.ini and restart eStudio.
```

Note: In Windows Server 2008, there are certain permissions settings that do not allow standard eclipse to function normally if eStudio is not running as an administrator. This will be resolved if eStudio is run as an administrator.

1.2.1 Launching eStudio using 64-bit JVM

By default eStudio is configured to launch using 32-bit JVM.

1.2.1.1 To install 64bit jars

1. Download eStudio 64bit JVM upgrade patch (javaupgradepatch.zip) from www.fiorano.com/latest/soa.

This patch can be used to upgrade eStudio from 32bit JDK to 64bit JDK and vice versa.

Note 1: This patch can be applied only in Windows and Linux platforms.

Note 2: Please do not rename any files in FIORANO_HOME/eStudio folder before executing these scripts. The patch takes care of files backup. If you need to backup in some external location then copy the files to the external location.

2. Extract javaupgradepatch.zip. This contains a file *Launching eStudio using 64.doc* and *eStudio-delta-pack* folder.

Copy the folder **eStudio-delta-pack** in **FIORANO_HOME/antscripts/patch** directory.

3. Open a console inside the folder **FIORANO_HOME/antscripts/patch/eStudio-delta-pack** and execute the command shown below:

```
ant patch
```

4. The patch will be applied and 64bit jar files are copied to the installer and backup is created in **eStudio-delta-pack/backup** and **eStudio-delta-pack/launchers/backup** directories.
5. Make the following change in **FIORANO_HOME/eStudio/eStudio.ini** file.

Add/Edit

```
-vm
```

```
{JAVA_HOME}\bin
```

The option `-vm` has to be added above the `"-vmargs"` line and `JDK_HOME` should point to a 64 bit JVM.

Note: `-vm` and `{JAVA_HOME}\bin` should be in two different lines.

6. In case of Linux, change the eStudio launcher executable permissions if required.
7. Now launch eStudio.
8. When eStudio is opened the JVM used can be checked from Help->About Fiorano eStudio->Installation Details->Configuration tab

1.2.1.2 To uninstall 64bit jars

1. Download the delta pack as mentioned in step1 in *"To install 64bit jars"* section.
2. Open a console inside the folder **FIORANO_HOME/antscripts/patch/eStudio-delta-pack** and execute the command shown below:
ant unpatch
3. The patch will be applied and 32bit jar files are copied to the installer and 64 bit jars backup is created in **eStudio-delta-pack/backup** directory.
4. Edit **FIORANO_HOME/eStudio/eStudio.ini** and change `-vm` option to point to 32bit JVM.
5. In case of Linux, change the eStudio launcher executable permissions if required.
6. Now launch eStudio.
7. When eStudio is opened the JVM used can be checked from Help->About Fiorano eStudio->Installation Details->Configuration tab

Chapter 2: Offline Event Process Development

The **Offline Event Process Development (OEPD)** perspective contains all the views and the editors required for the offline event process development. The OEPD perspective maintains its own repository of Event Processes and Services and no server connection is required to create Event processes. This offline repository is populated when the user launches the Fiorano eStudio for the first time. The default location of the Offline repository is \$FIORANO_HOME/runtimedata/eStudio/workspace/.repositories/Offline.

A Server connection is required only to export the developed Event Processes into the Server. Similarly, Event Processes present in the Server can also be imported into the eStudio. Figure 2.1 illustrates the OEPD perspective.

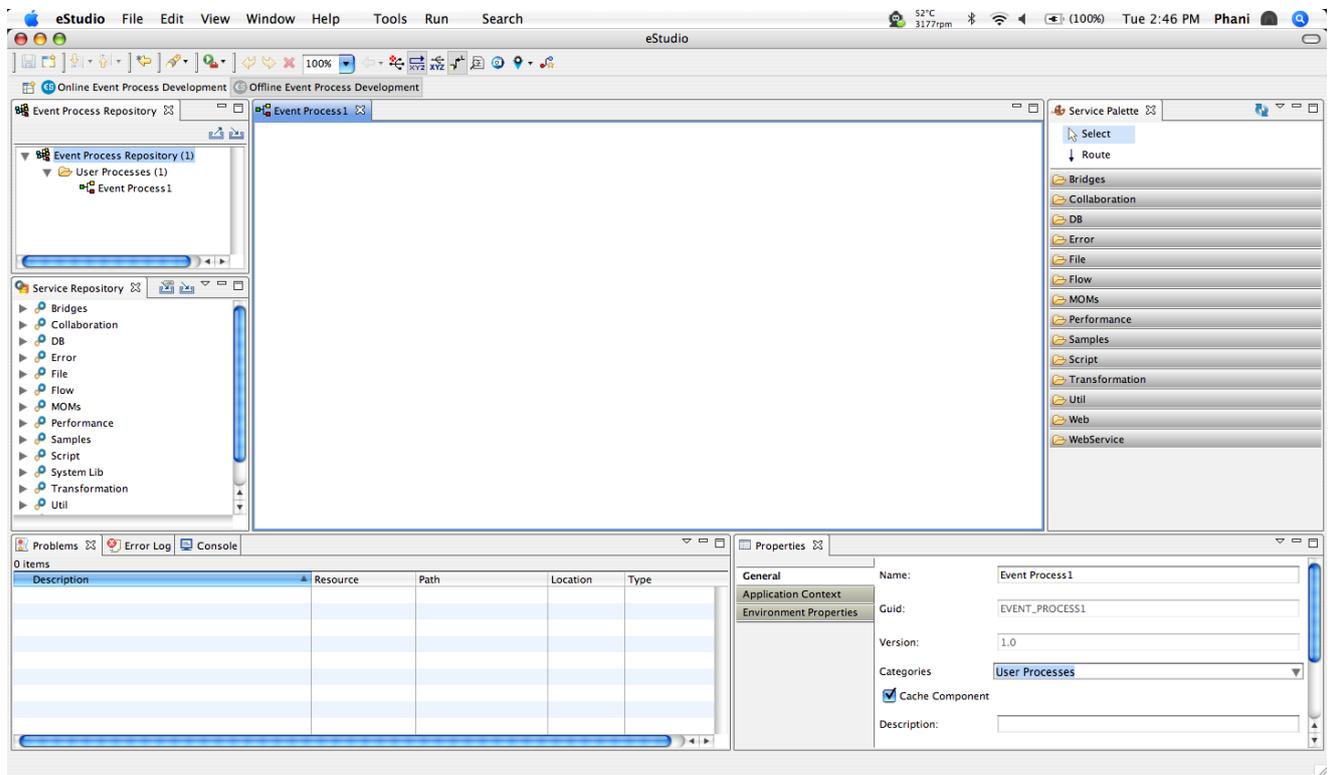


Figure 2.1: Offline Event Process Development perspective

The OEPD perspective comprises of various Views as explained in the following section.

2.1 Fiorano Views

2.1.1 Event Process Repository View

The Event Process Repository view is one of the views of the Offline Application Development Perspective, which is available under Window > Show View > Fiorano > Event Process Repository.

Event Process Repository view shows all the event processes created in the offline application development perspective, under various categories.

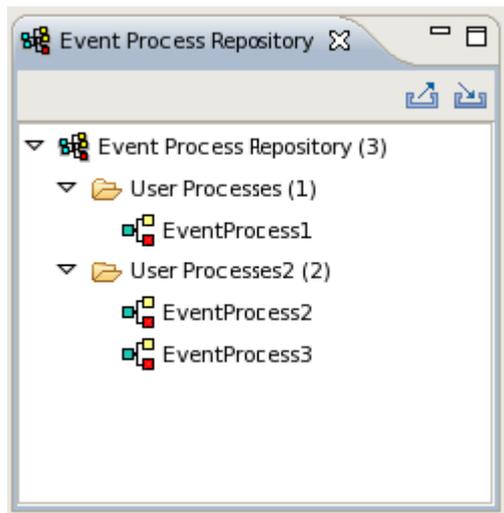


Figure 2.1.1: Event Process Repository

2.1.2 Fiorano Orchestration

Offline and Online Event Process Development perspectives are comprised of an editor area Fiorano Orchestrator.

When an Event Process is opened, the design of the event process is shown in the Fiorano Orchestrator.

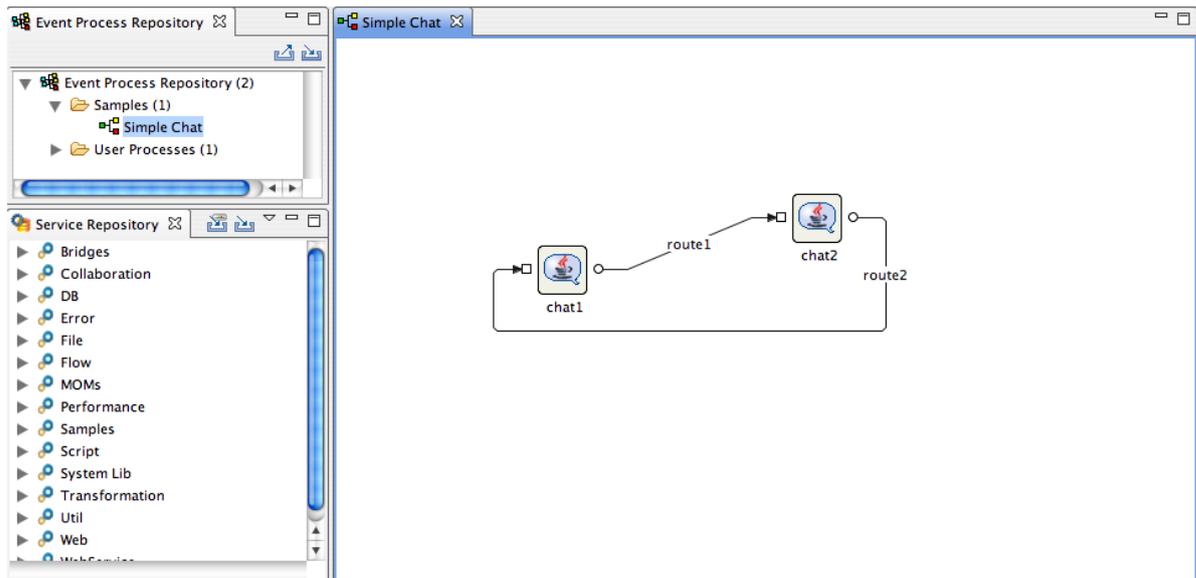


Figure 2.1.2: Orchestration Editor

2.1.3 Service Palette

The Service Palette shows the services that are present in the eStudio repository. The Service Palette contains all the Fiorano services grouped into various categories such as: Bridges, Collaboration, DB, Error, File, and so on as shown in figure 2.1.3.

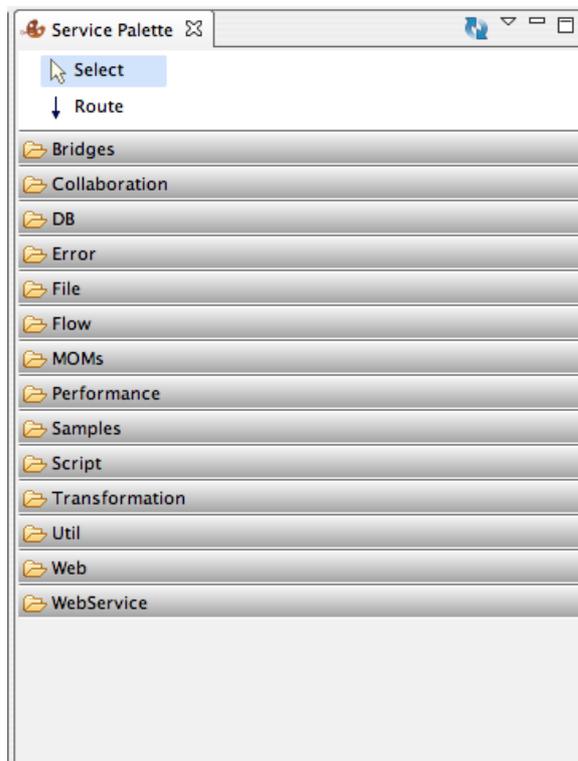


Figure 2.1.3: Fiorano Service Palette

2.1.4 Properties

The Properties view displays all the property names and values for any selected item such as: a service instance, route, port, and so on. The Properties view is available under Window > Show View > Other > General > Properties.

Placing the cursor on a property shows the property description.

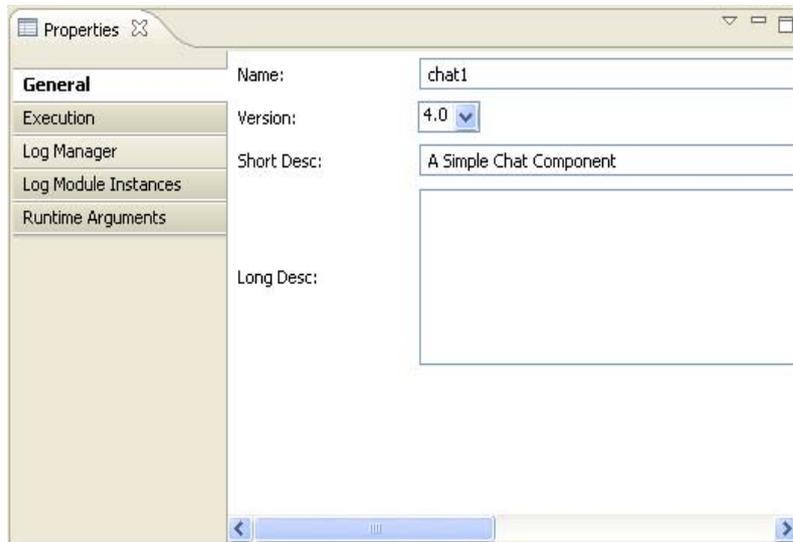


Figure 2.1.4: Properties view

2.1.5 Problems

When working in the Fiorano environment, the errors and warnings occurred are displayed in the Problems view. For example, when an Event Process containing errors is saved, the errors are displayed in the Problems view as shown in Figure 2.1.5.

The Problems view is available under Window > Show View > Other > General > Problems.

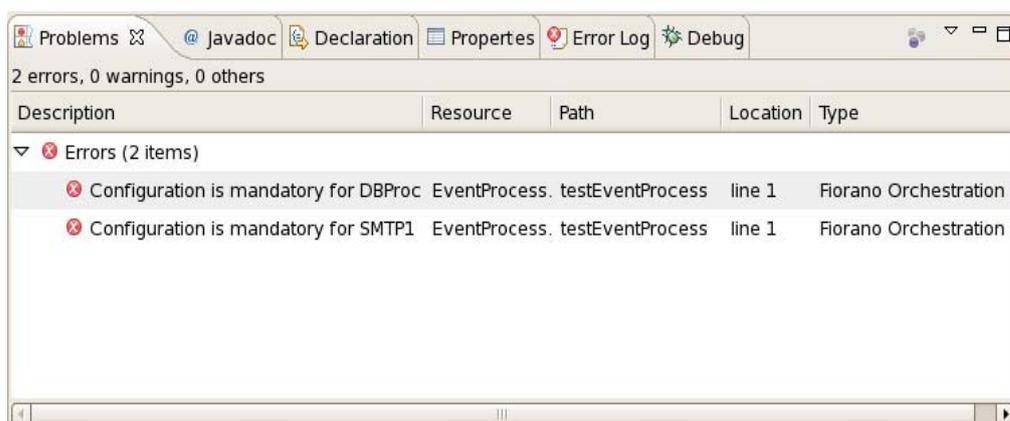


Figure 2.1.5: Problems view

By default the problems are grouped by severity level. The grouping can be selected using the Group By menu.

Problems view can also be configured to show the warnings and errors associated with a particular resource or group of resources. This is done using the Configure Contents option in the drop-down menu. Additionally, you can add multiple filters to the problems view and enable or disable them as required. Filters can either be additive (any problem that satisfies at least one of the enabled filters will be shown) or exclusive (only problems that satisfy all of the filters will be shown).

2.1.6 Error Log

The Error Log view captures all the warnings and errors logged in the Fiorano environment. The underlying log file (.log) is stored in the .metadata subdirectory of the workspace. The Error Log view is available under Window > Show View > Error Log.

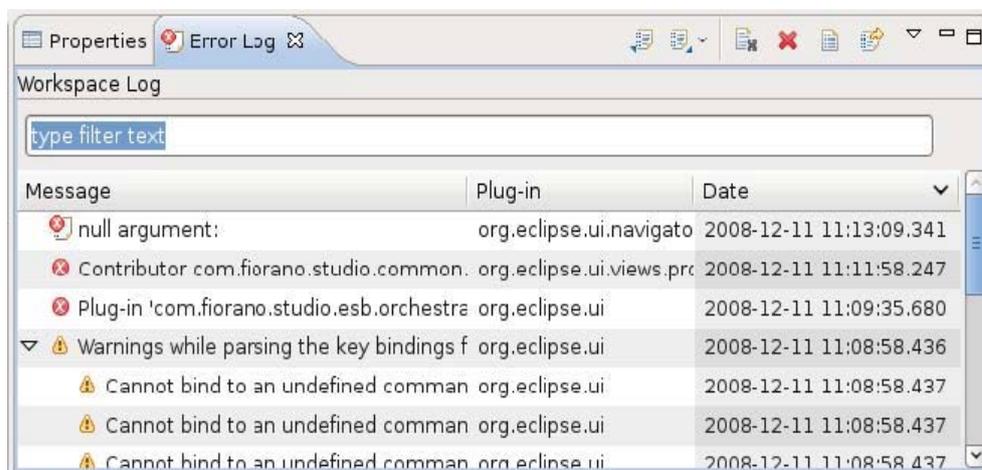


Figure 2.1.6: Error Log view

2.1.7 Service Repository (Offline)

Fiorano eStudio provides a Service Repository view which is available under Window > Show View > Fiorano > Service Repository. This shows a categorized list of all available services. When the Fiorano eStudio is launched for the first time, the offline repository will be loaded from the installer.

Services which are available only in the service repository can be used for composing event processes in eStudio. Services can be imported from or exported to a file system or a Fiorano ESB Server from the Service Repository.

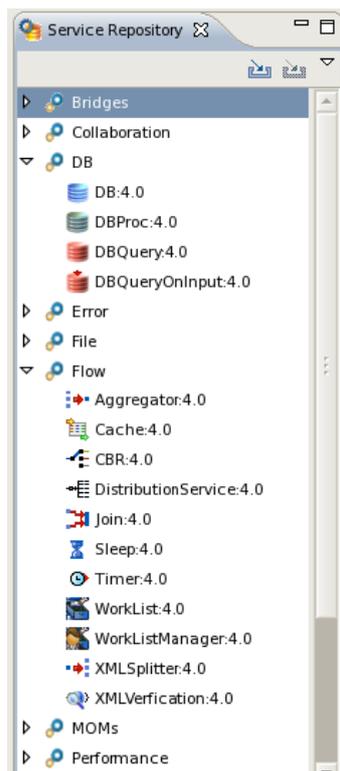


Figure 2.1.7: Service Repository

2.1.8 Project Explorer

The Project Explorer view lists all the projects in eStudio. The Project Explorer view is available under Window > Show View > Project Explorer.

All the Event Process, Service and Mapper projects are shown in Project Explorer view. Structure of the Event process is shown in Figure 2.1.8.

To use Version Control, corresponding plug-ins have to be added in drop-ins. If the drop-ins are added, then the version control options will be available in the context menu of a project in this view.

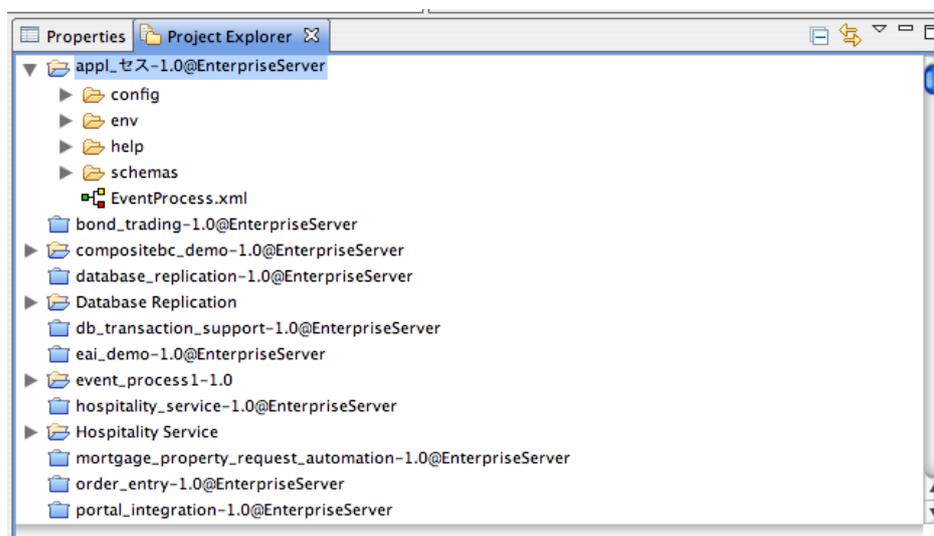


Figure 2.1.8: Project Explorer

The service projects are by default shown as closed projects. User can open a project by right-clicking on a project and by selecting the Open Project option. For performance reasons it is advised to close the service projects when they are not being used.

2.1.9 Service Descriptor Editor

Service can be edited using a Service Descriptor editor. To edit a service in the service descriptor editor, right-click on the desired service in Service Palette or in Service Repository and click the edit option from the context menu.

The properties of Service are divided into three categories:

- **Overview** – Contains general information about the Service like Name, GUID, version, icon etc.
- **Execution** – Contains information about service ports, runtime arguments, launch options and log configuration.
- **Deployment** – Contains information about service resources, dependencies and general deployment information.

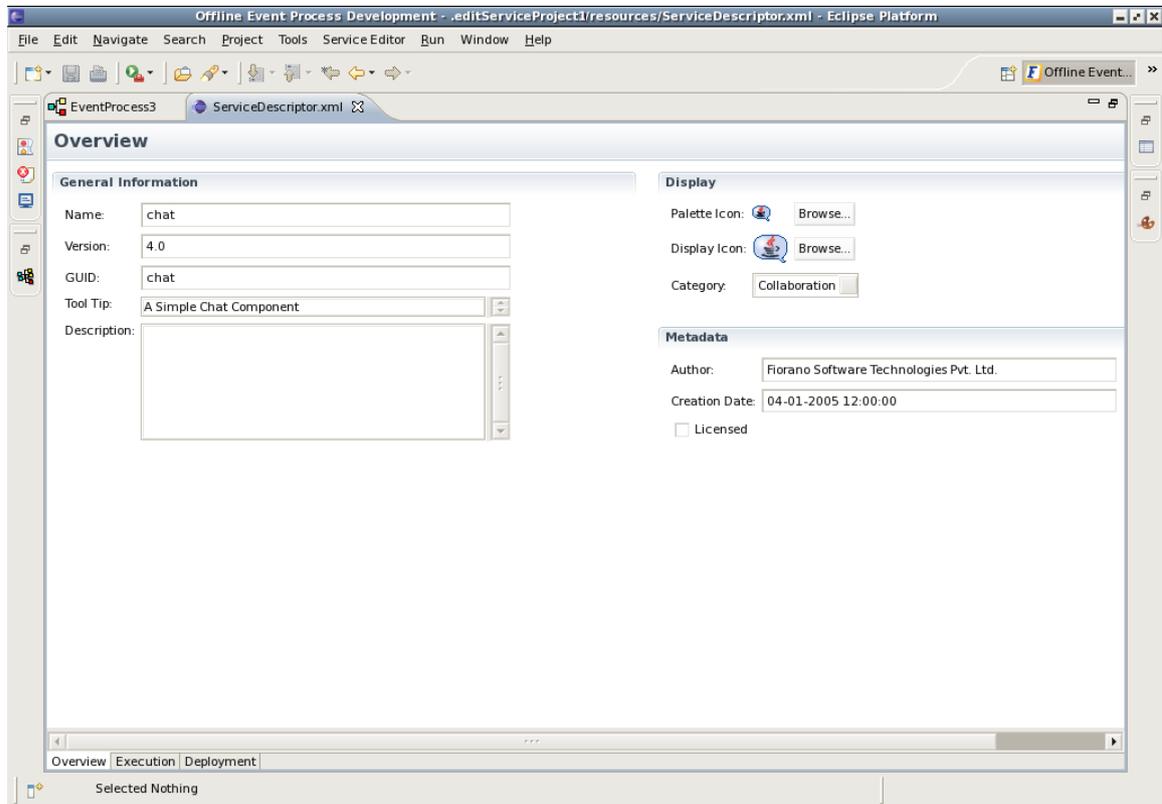


Figure 2.1.9: Service Descriptor Editor

Note: Changes made to the Service will be saved to the repository only after the editor associated with the Service is closed.

2.2 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. Event processes in Fiorano are designed to connect disparate applications in a heterogeneously distributed SOA environment.

Fiorano eStudio 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, profile information, and layout. The event process metadata contains all required information in XML format, which is stored in the repository.

2.2.1 Creating New Event Process

To create a new Event Process, perform the following steps:

1. Right-click on the Event Process Repository node and select **Add Event Process**. The Customize Event Process dialog box appears.

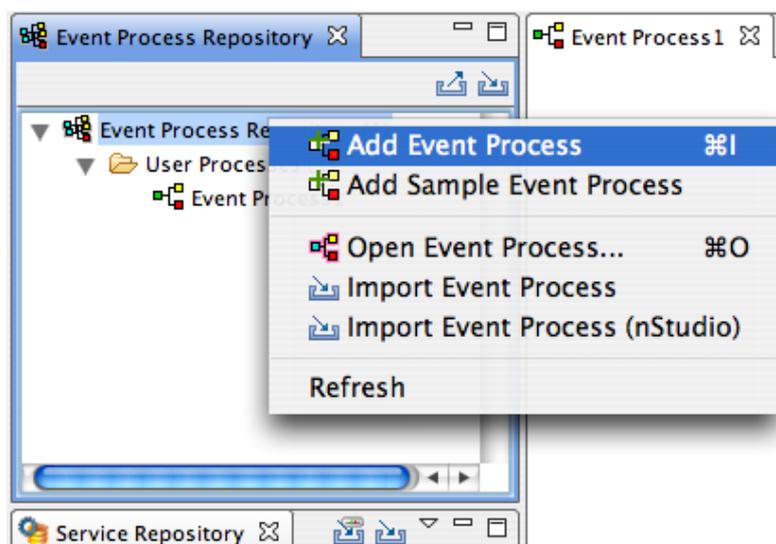


Figure 2.2.1: Creating new Event Process

- Specify the name and category of the Event Process project and click **Finish**. The specified Event Process appears under **Event Process Repository** node of Event Process Repository view.

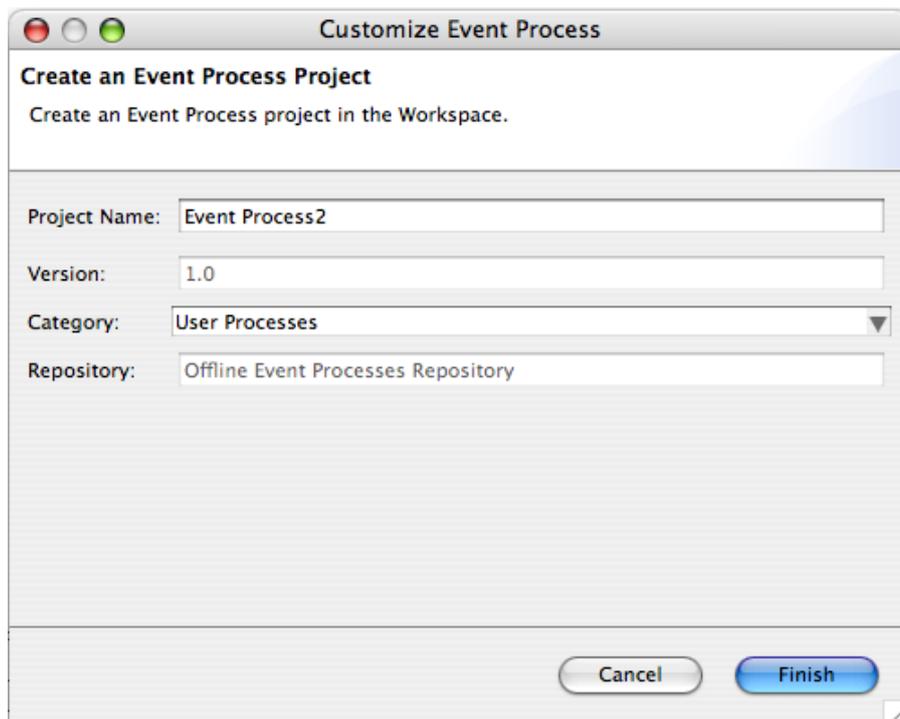


Figure 2.2.2: Customize Event process

- To see the graphical view of an Event Process, double-click the event process node, which opens the **Fiorano Orchestration** editor. For information on composing an Event Process, see [Chapter 5: Composing an Event Process](#).

2.2.2 Opening Sample Event Process

Few pre-configured sample event processes are shipped with the Fiorano installation. To open a pre-configured sample event process, perform the following steps:

- Right-click on the Event Process Repository node and select **Add Sample Event Process**. The **Add Sample Event Process** dialog box appears.

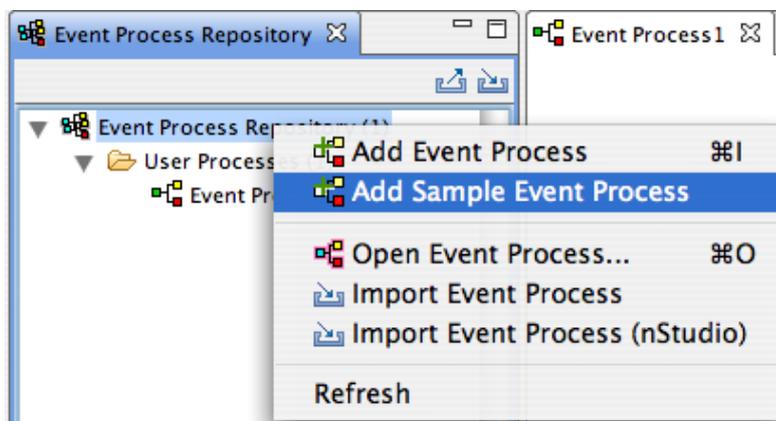


Figure 2.2.3: Add Sample Event Processes

2. Select the Event Process(s) to be opened by selecting the check box against each entry and click **Finish**. The selected Event Process(s) appears under Event Process Repository Node.

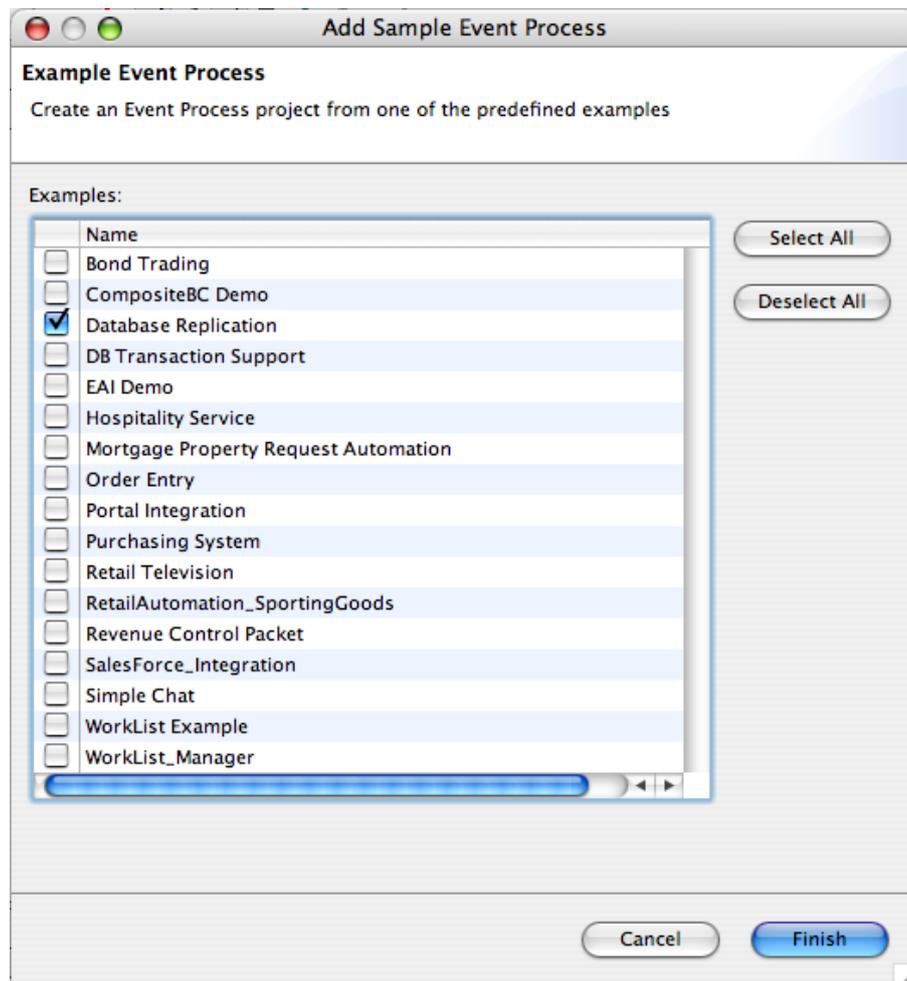


Figure 2.2.4: Sample Event Processes

3. To see the graphical view of an Event Process, double-click the event process node which opens in the Fiorano Orchestration editor.

Note: The samples that are added to the repository already will not be visible in the Add Sample Event Process wizard.

2.2.3 Import and Export Event Processes

The following sections describe the procedure for exporting and importing an event process.

2.2.3.1 Exporting an Event Process

Event Process can be exported to local disk or to a server from the Offline Event Process Development perspective.

To export an Event Process onto a local disk, perform the following steps:

1. Right-click on the Event Process to be exported from the Event Process Repository view and select Export from the menu (Figure 2.2.5).

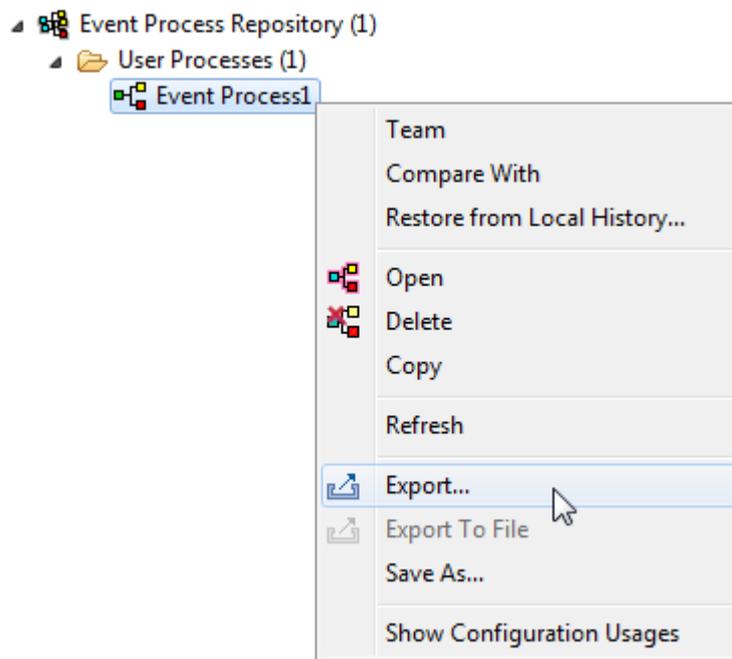


Figure 2.2.5: Export Event Process to local disk

2. The Export Event Process dialog appears as shown in the Figure 2.2.6. The dialog provides a list of all the Event Processes selected for exporting and also the named configurations used in those Event Processes. User can choose the artifacts to be exported by selecting the check boxes next to their names.

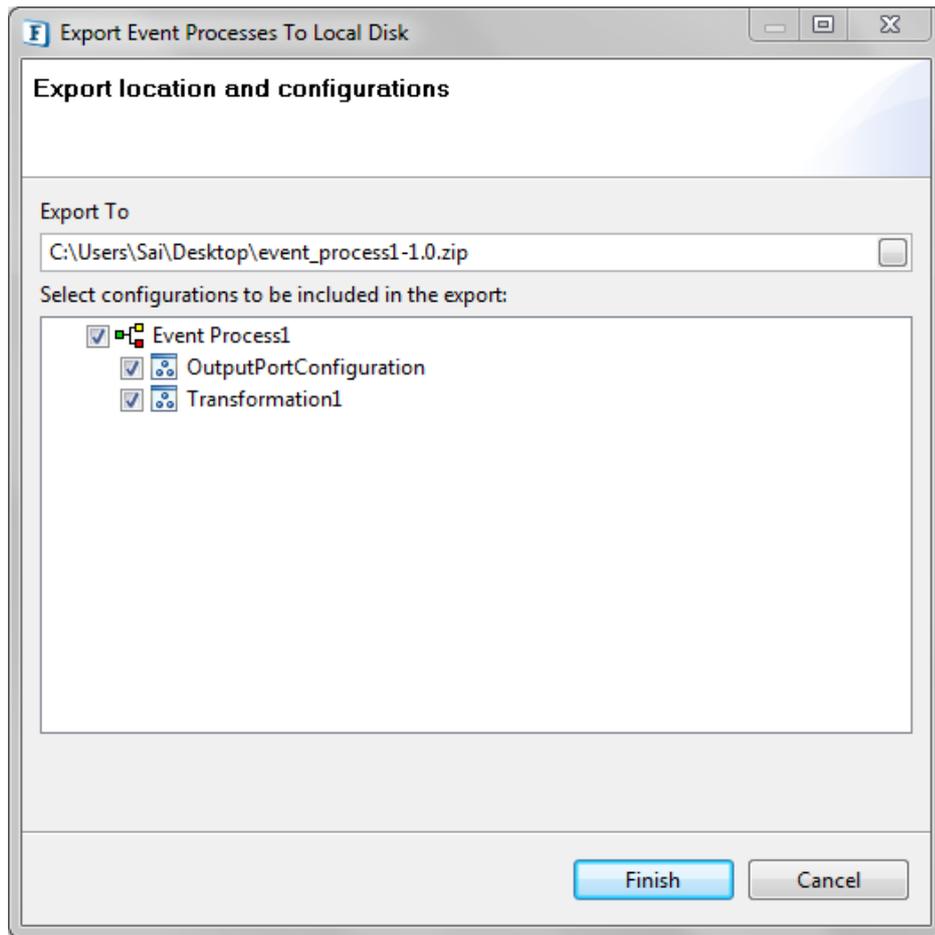


Figure 2.2.6: Selecting Configurations to export

3. Specify the file name and location to save and click **OK**. The Event Process project along with the configurations chosen will be saved as a .zip file.

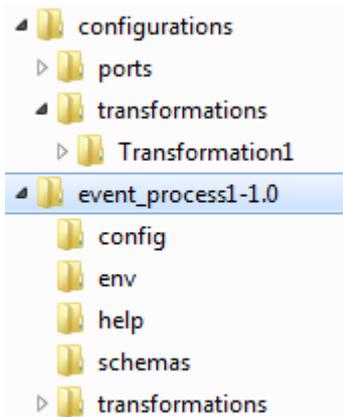


Figure 2.2.7: Contents of the exported ZIP file

To export an Event Process to Server, perform the following steps:

1. Click on **Export Event Process to Server** icon located on Event Process Repository view tool bar as shown in the figure 2.2.8. The **Select Event Process To Be Exported** dialog box appears listing all the Event Processes in offline repository and shows the Servers list specified in Fiorano Preferences. For more information on configuring servers please refer to [Chapter 12: Fiorano Preferences](#).

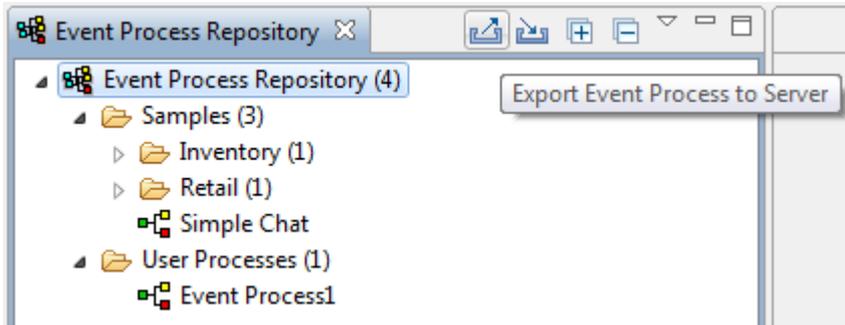


Figure 2.2.8: Export Event Process to Server

2. Select the Event Process and Configurations to be exported and the Server onto which is to be exported and click **Finish**.

Note:

While exporting to a server, if an Event Process or a Configuration already exists on the Enterprise server, it is shown in **Red** color. On clicking **Finish**, a confirmation dialog will be shown asking if the conflicting artifacts are to be overwritten. Choose the appropriate action.

If a configuration used in an Event Process is missing from the Configuration Repository, the missing configurations will be indicated by a **Red Cross** across the configuration icon (✘). Such Event Processes cannot be exported to a server.

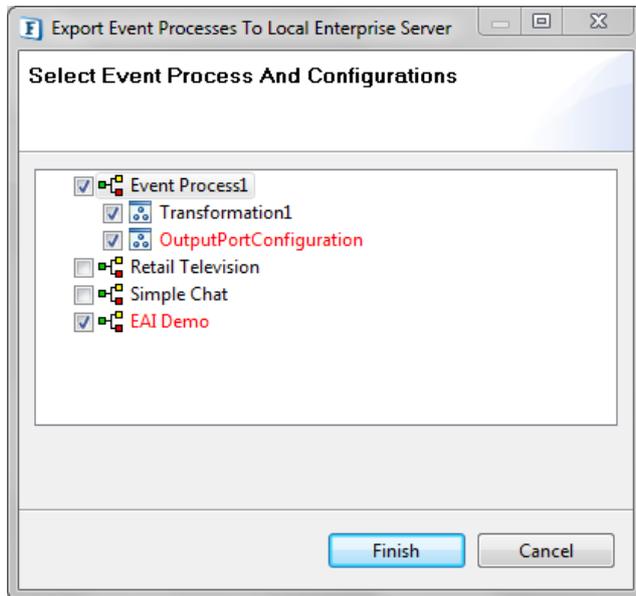


Figure 2.2.9: Select Event Process and Configurations to be exported

2.2.3.2 Importing an Event Process

Event Process can be imported from the local disk and from the Server.

To import an Event Process from local disk, perform the following steps:

1. Right-click the **Event Process Repository** node and select **Import Event Process** from the menu as shown in figure 2.2.10.

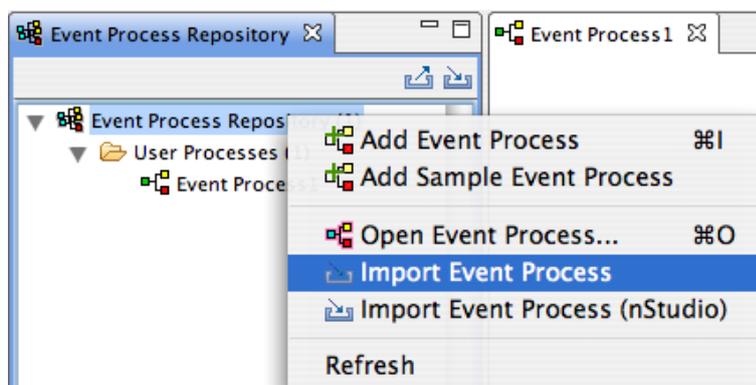


Figure 2.2.10: Import Event Process from local disc

2. Specify the location of Event Process zip file and click **OK**. A list of all Named Configurations used in the Event Process is also provided. The user can select the configurations to be imported along with the Event Process in the **Import Applications wizard** (Figure 2.2.11).

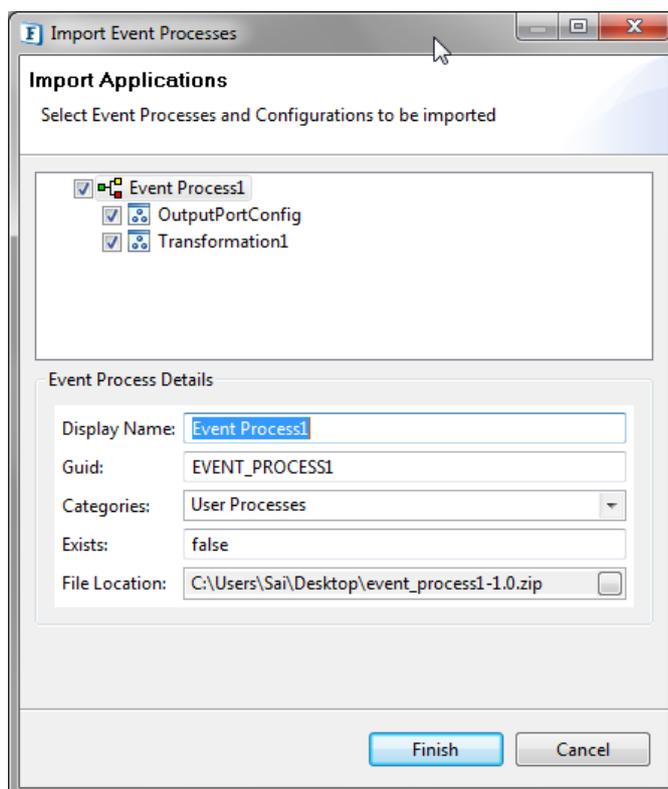


Figure 2.2.11 Import Application Wizard

Note: Event Processes or Configurations that are already present in the repository are shown in **Red** and the configurations which are used in the Event Process and are not found the import ZIP file are indicated using a Red cross (✖).

To import an Event Process from the Server, perform the following steps:

1. Click on **Import Event Process from Server** icon present on **Event Process Repository** view tool bar as shown in the figure 2.2.12.

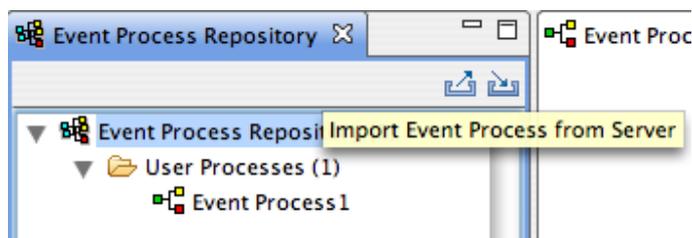


Figure 2.2.12 Import Event Process from Server

Select a Server dialog box appears listing all servers specified in Fiorano ESB Connection Preferences page.



Figure 2.2.13: Select Enterprise Server

2. Select the Server from which Event Process has to be imported and click **OK**. The **Select Event Process To Be Imported** dialog box appears which lists all the Event processes deployed in the server as shown in Figure 2.2.14.

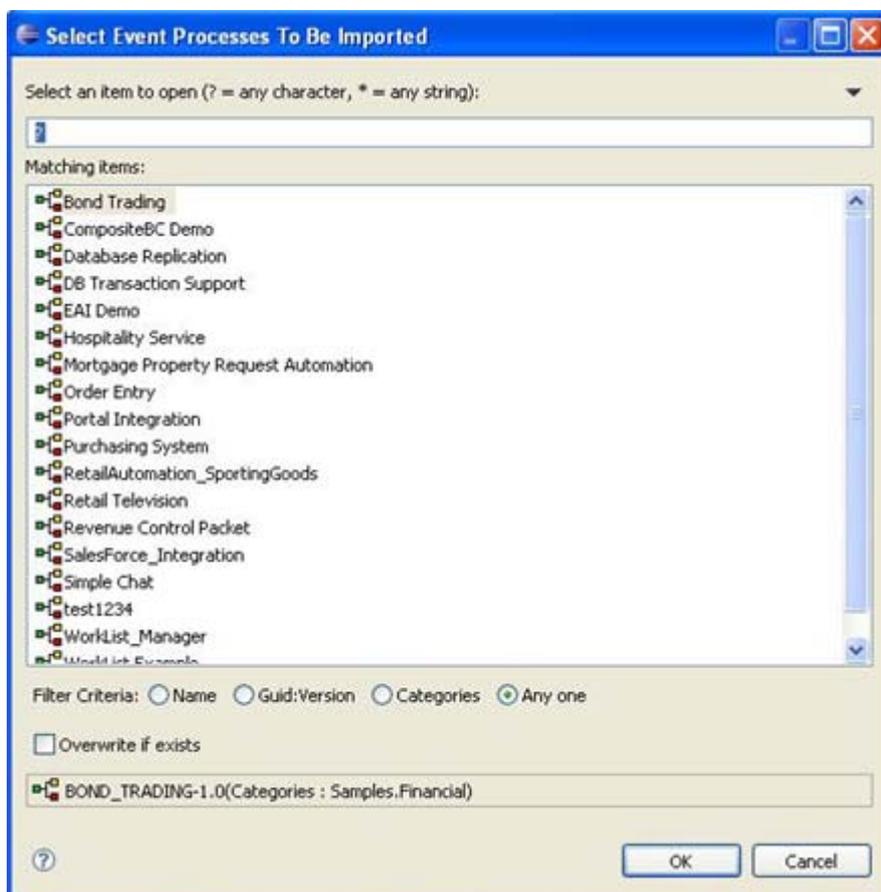


Figure 2.2.14: List of Event Process in server

3. Select the Event Process to be imported and click **OK**. The Import Applications wizard (Figure 2.2.11) is shown. Selected the Applications and configurations to be exported and click **Finish**.

2.2.4 Importing nStudio Event Processes

Event Processes that are developed and exported from nStudio can be imported into eStudio using the Import Event Process (nStudio) option present on the context menu of Event Process Repository node.

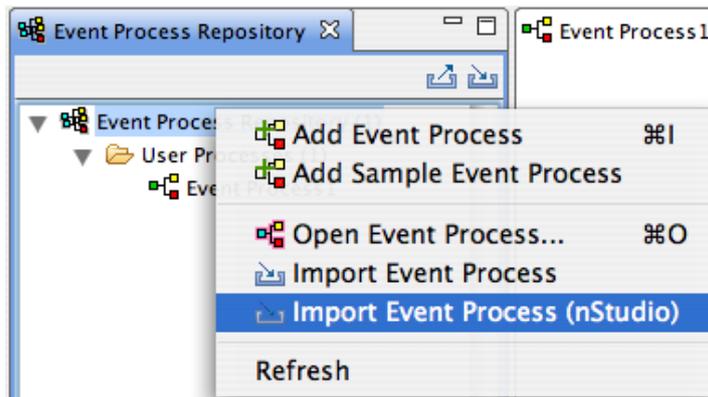


Figure 2.2.15: Import nStudio Event Process

Selecting the import option opens an Import Wizard as shown in Figure 2.2.16.

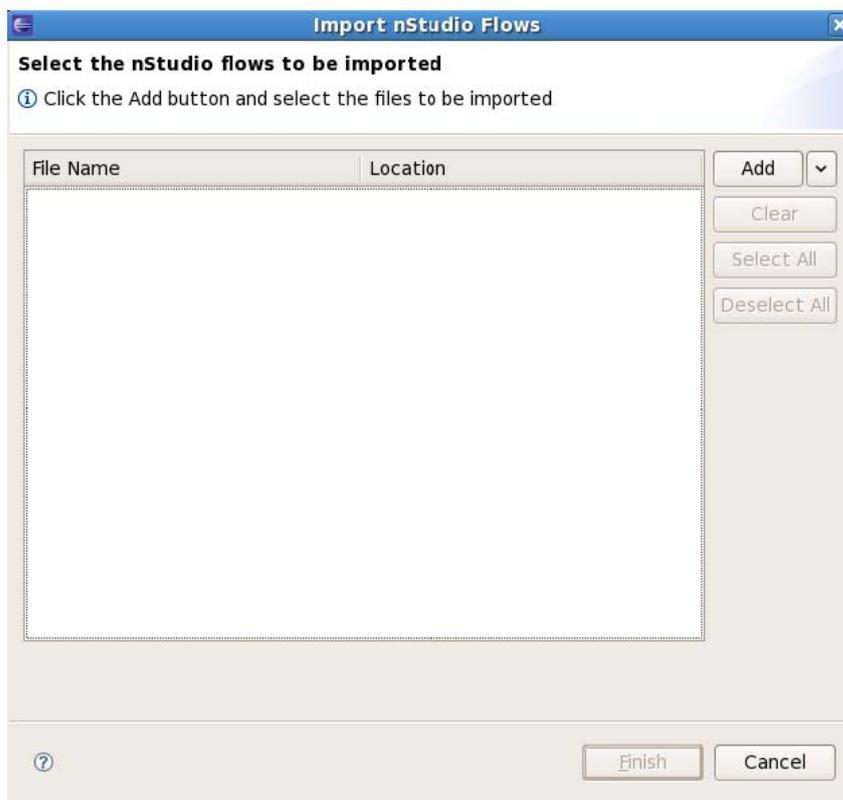


Figure 2.2.16: Import nStudio Flows

The Event Processes to be imported can be added to the table by clicking the **Add** button. A file chooser dialog appears where the nStudio flows can be selected. Multiple files can also be selected at once.

To import all the Event Processes present in a particular folder, select the **Add From Folder** option present in the drop-down button located on the right side of the add button. All the supported flows present in the folder and all of its sub-folders will be added to the table.

The state of the wizard after adding the flows is shown in Figure 2.2.17:

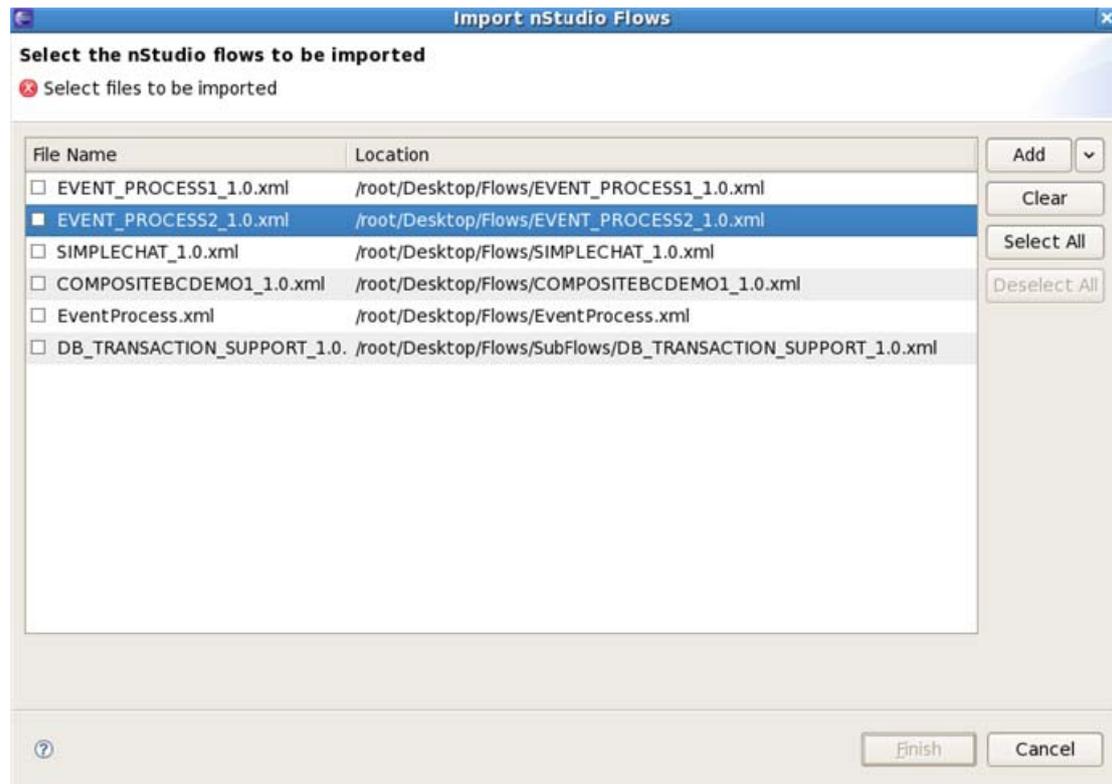


Figure 2.2.17: Select nStudio flows to import

After adding the selected files to the table, the flows to be imported can be selected using the check box against each entry present in the table. Click the **Finish** button to import the selected flows to the current repository. If any of the selected flows already exist in the repository, the user is prompted with a dialog box with the options to overwrite/ignore/rename the flow. The imported flows can be viewed under the Event Process Repository Node.

2.3 Service Repository (Offline Event Process Development)

Fiorano eStudio has an independent service repository in Offline Event Process Development perspective, which enables services to be configured offline (without connecting to the Enterprise Server).

The service repository can be viewed by opening the Service Repository view, which displays categorized services as shown in Figure 2.3.1.

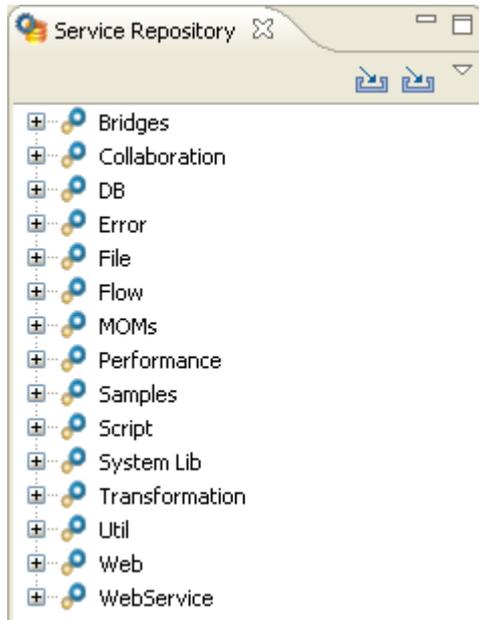


Figure 2.3.1: Service repository

2.3.1 Deploying Services to Server

A service can be deployed to an Enterprise server by right-clicking the component in Service Repository view and selecting **Export Service to Server** from the context menu. The **Export Service To Server** dialog box appears as shown in Figure 2.3.2.

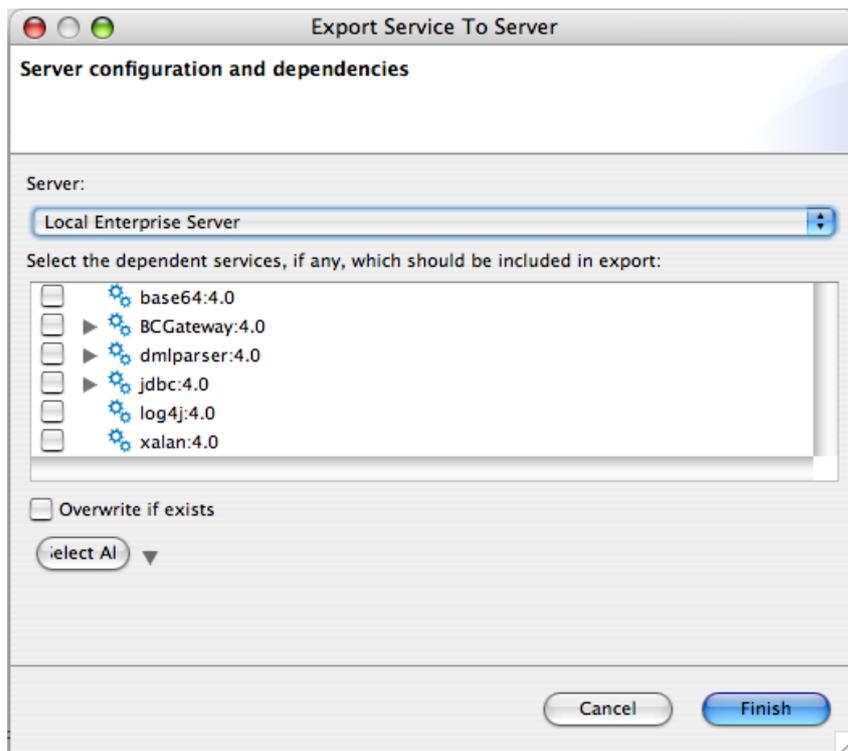


Figure 2.3.2: Export service to server

The dependencies are shown in a tree format. This excludes the actual Service (which gets exported by default). To export any dependencies of this Service, select the Dependency and click **Finish**.

If the **Overwrite If Exists** checkbox is selected, the services in the server will be overwritten by the one in the Service Repository, otherwise conflicting services will not be export to the server.

2.3.2 Fetching Services from Server

1. The services present on server can be imported into the service repository by selecting the **Import from Server** option as shown in Figure 2.3.3.

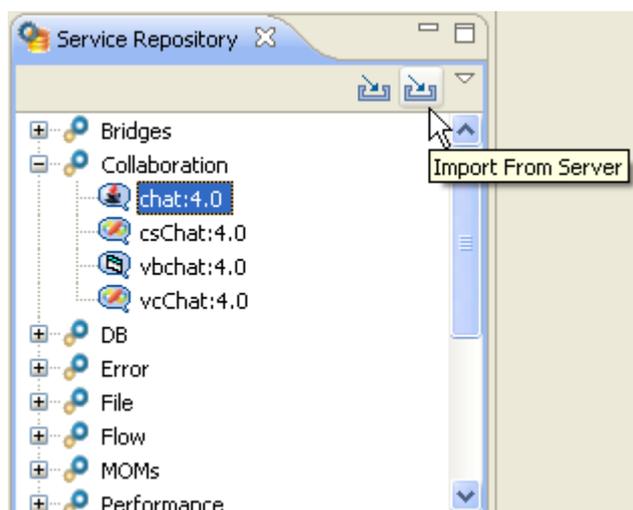


Figure 2.3.3: Import from Server option

This opens **Import Service From Server** dialog box as shown in Figure 2.3.4.

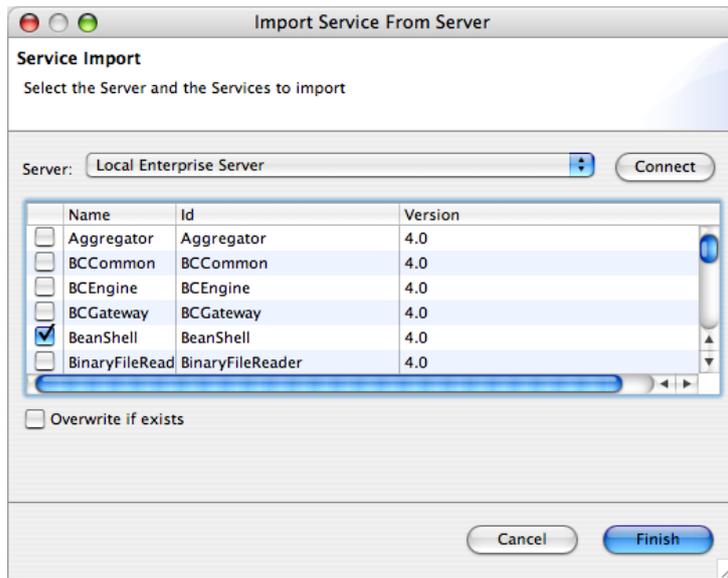


Figure 2.3.4: Import service from server

2. Select the server from where services have to be imported and click the **Connect** button. This displays all the available services in that server.
3. Select the services to be imported and click the **Finish** button to import the service.

If the **Overwrite If Exists** checkbox is selected, service in the Service Repository will be over-written by the one in the server, otherwise conflicting services are not imported from the server.

2.3.3 Exporting Services to Local Disk

The Services in Service Repository can be exported to a local disk by right-clicking the Service and selecting the **Export Service To Local Disk** option from the context menu. This opens the **Export Service To Local Disk** dialog box as shown in Figure 2.3.5.

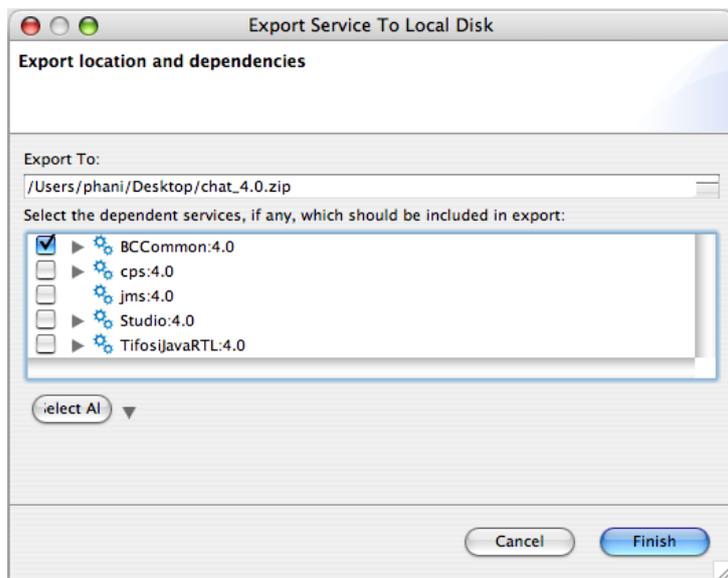


Figure 2.3.5: Export service to Local Disk

You can choose the export location, by default only the selected service gets included in the export. Select other services from the tree to be exported if required, and click the **Finish** button to export the service.

2.3.4 Importing Services from Local disk

The components can be imported from the file system. This can be done by clicking the **Import From Local Disk** button as shown in Figure 2.3.6.

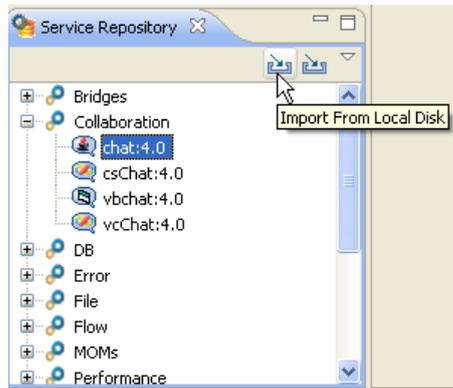


Figure 2.3.6: Import from Local Disk button

This opens the **Import Services** file selection dialog box with which the zip file containing services on the disk is selected. Upon selection a dialog box is shown in which the services in the zip file are shown in the form of a dependency tree as shown in Figure 2.3.7.

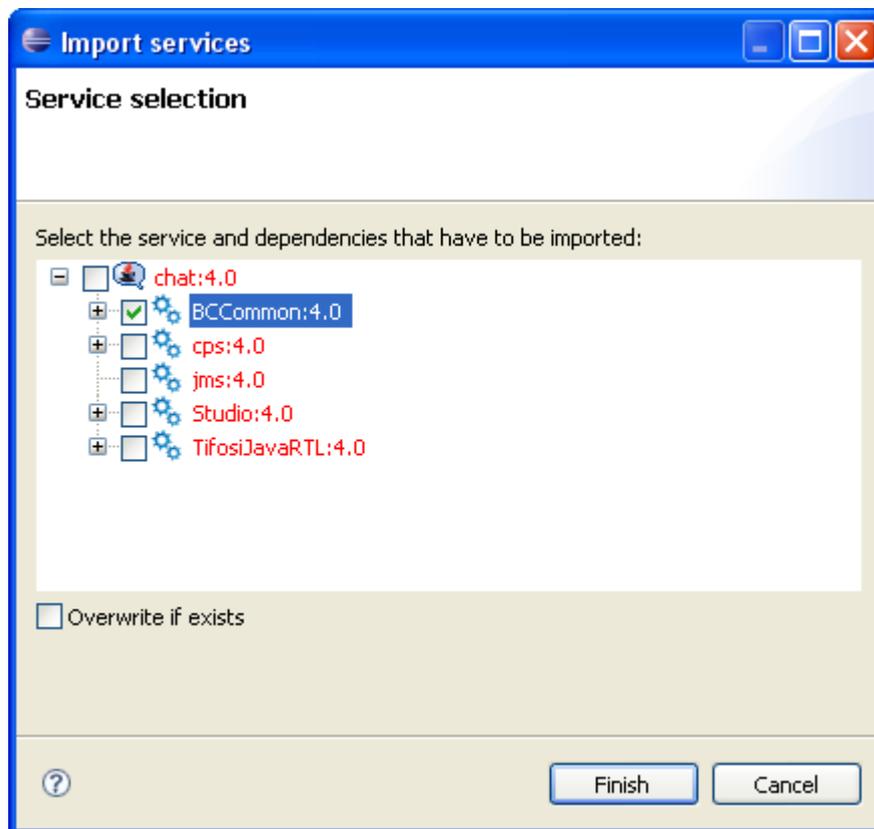


Figure 2.3.7: Import services dialog box

Components that are already present are labeled in red, and those not present in the repository are labeled in black.

If the **Overwrite If Exists** checkbox is selected, service in the service repository will be over written by the one in the zip file, otherwise conflicting services are not imported from the local disk.

Chapter 3: Online Event Process Development Perspective

To open the Online Event Process Development perspective, perform the following steps:

1. Click **Windows** on the menu bar, select **Open Perspective** and click on **Others..** option from the drop-down menu. Or click the **Open Perspective** button from the shortcut bar and select **Other...** from the drop-down menu. The Open Perspective dialog box appears.
2. Select the Online Event Process Development to open online perspective. Click the **OK** button.

Online perspective contains all the views and editors required for online application development. During online Event Process development, event process development can be done after logging into the Enterprise Server.

After switching onto Online Event Process Development mode, select the Enterprise Server node, right-click and select **Login** to login into the Enterprise Server.

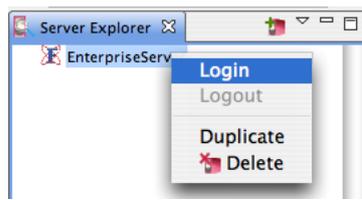


Figure 3.1.1: Enterprise Server node

By default the configurations of the Enterprise Server running locally is set on the Enterprise Server node. These can be changed from the properties view if required.

Each time during login, eStudio fetches the information of Services, Event Processes and Peer Servers from the Enterprise Server and populates the online repository. The default location of the online repository for a particular Enterprise Server is `$FIORANO_HOME/runtimedata/eStudio/workspace/.repositories/Online/<Enterprise Server name>`. Screenshot of the Online Event Process Development mode is shown in Figure 3.1.2

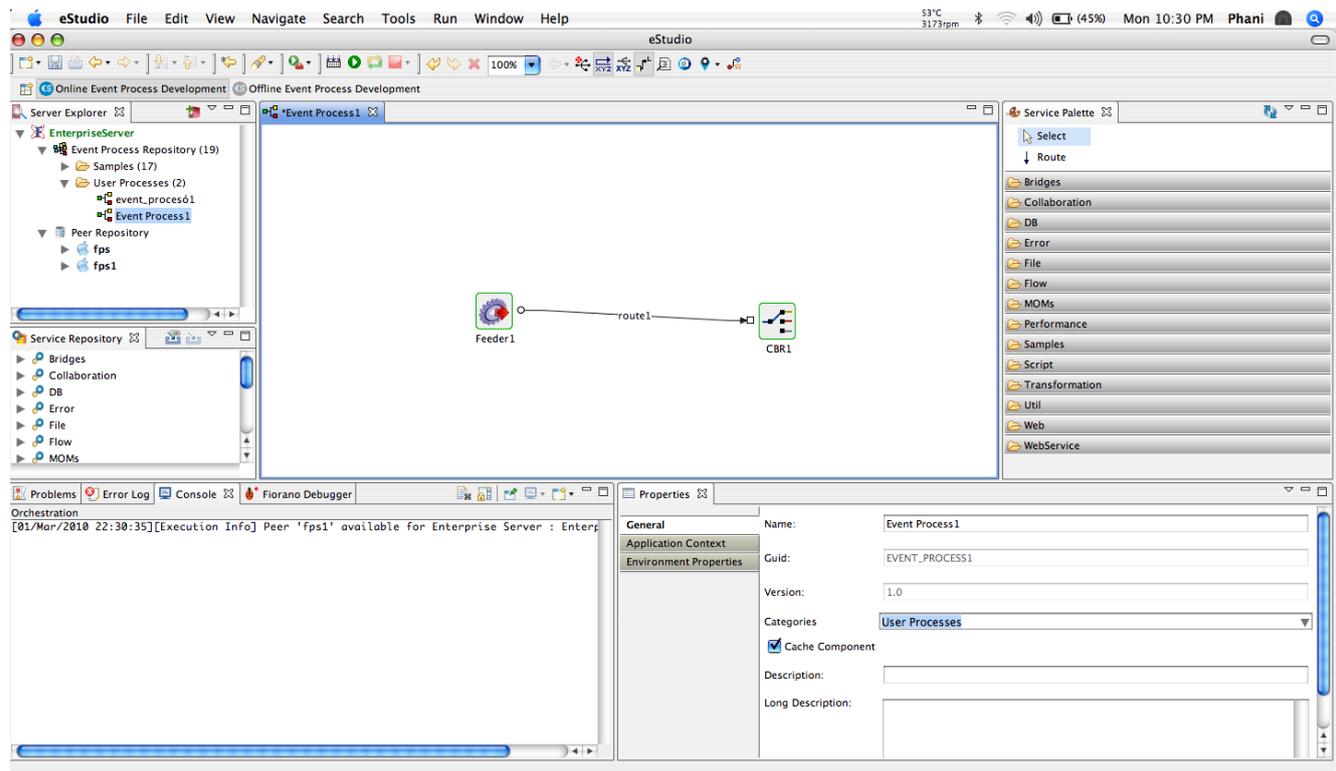


Figure 3.1.2: Online Event Process Development perspective

The Event Process Repository node contains a tree structure of various Event Processes in the Enterprise Server.

The Peer Repository node contains the information of Peers connected to the Enterprise Server.

3.1 Fiorano Views

All the views described in Offline Event Process Development mode are available in Online mode. There are additional views specific to Online mode. These views are described in this section.

3.1.1 Server Explorer

The Server Explorer view shows the Enterprise servers, which contains Event Process Repository and Peer Repository nodes.

The Server Explorer view is available under Window > Show View > Fiorano > Server Explorer.

The Event Process repository is centrally stored in the Enterprise Server. The Enterprise Server provides API access to the event processes such as save, view, export, launch, debug, stop, and similar actions as required. The Fiorano eStudio provides an easy-to-use GUI to manage event processes. The Peer Repository shows the peer servers connected to the Enterprise Server.

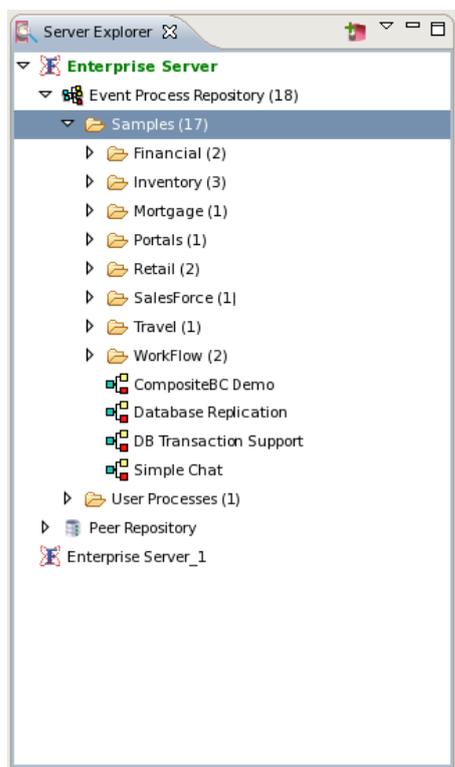


Figure 3.1.3: Server Explorer

3.1.2 Fiorano Debugger View

The Fiorano Debugger view shows the list of routes on which debugger is enabled and messages trapped within each route. This gives users the ability to take action on debug message.

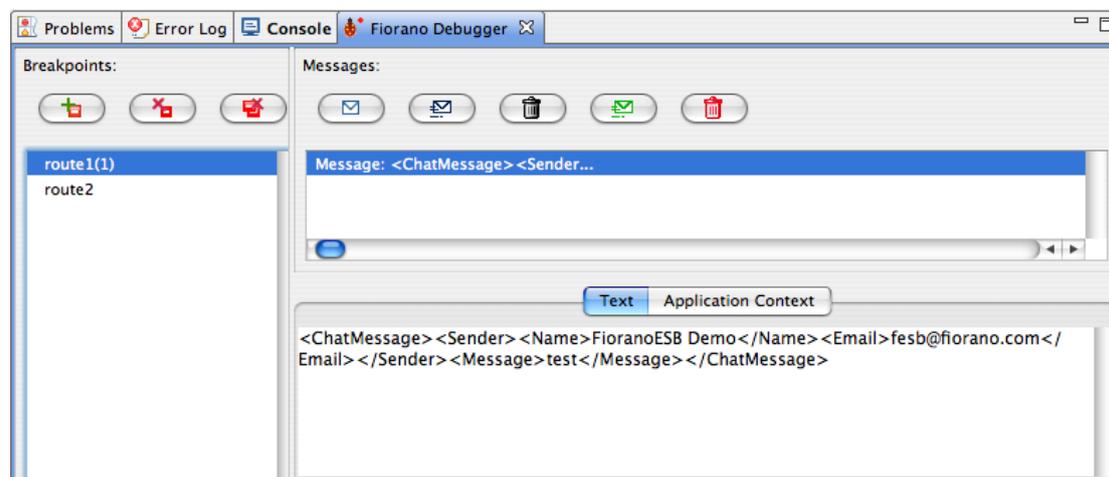


Figure 3.1.4: Fiorano Debugger view

3.2 Service Repository (Online Event Process Development)

In Online Event Process Development perspective, the services present in the connected enterprise server are shown in the service repository.

The service repository can be viewed by opening the Service Repository. This view which display the categorized services as shown in Figure 3.2.1.

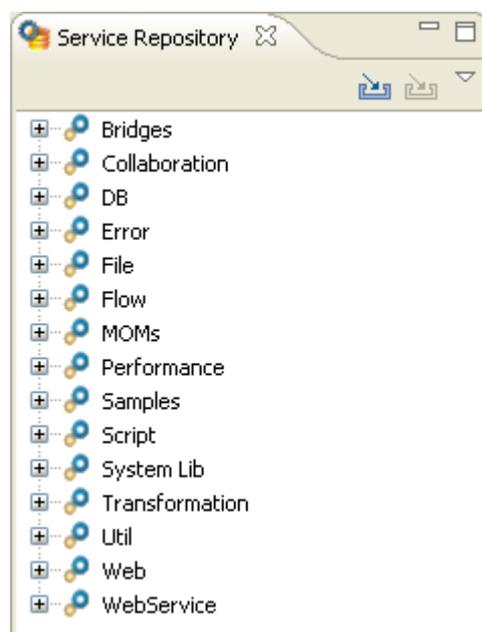


Figure 3.2.1: service repository (Online Event Process Development perspective)

3.2.1 Exporting Services to Local Disk

The Services in Service Repository can be exported to local disk by right-clicking the service and selecting Export to Local disk option from context menu. This opens a dialog box as shown in Figure 3.2.2.

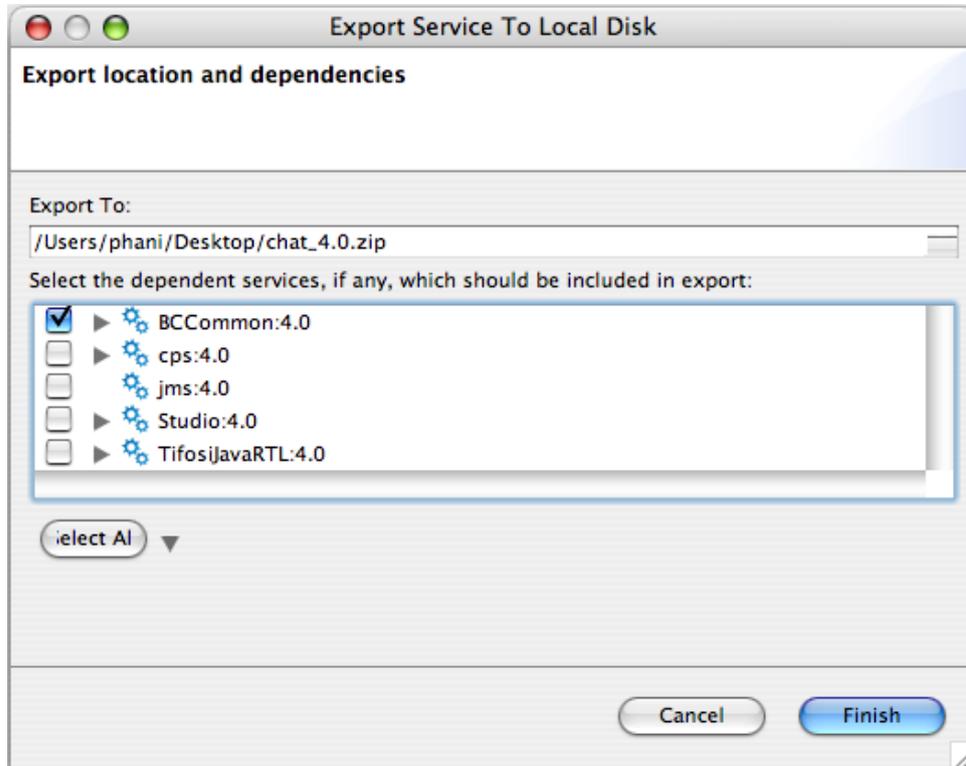


Figure 3.2.2: Export service to Local Disk

You can choose the export location by clicking the Browse button and specifying the location to be exported. By default, the selected service gets included in the export. To export Dependent services have to be selected from the tree as shown in Figure 3.2.2.

3.2.2 Importing Services from Local disk

Components can be imported from the file system. This can be done by clicking the **Import From Local Disk** button as shown in Figure 3.2.3.

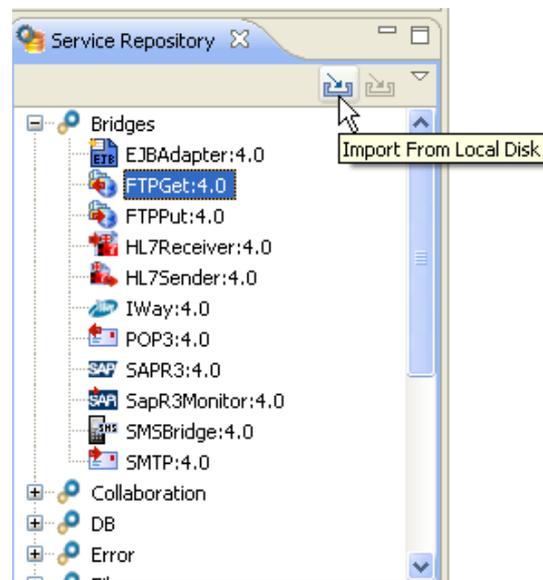


Figure 3.2.3: Import from Local Disk

This opens a file selection dialog box with which the zip file containing services on the disk is then selected. Upon selection, the **Import services** dialog box appears where the services in the zip file are shown in the form of a dependency tree, as shown in Figure 3.2.4.

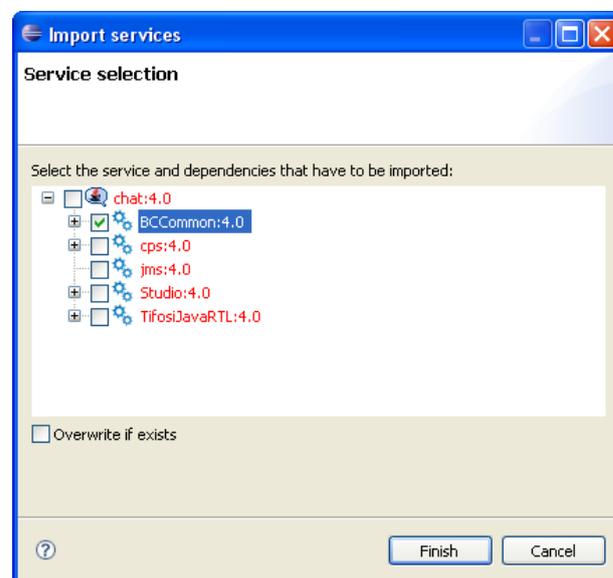


Figure 3.2.4: Import services dialog

Note: Components that are already present are labeled in red, and those not present in the repository are labeled in black.

If the **Overwrite If Exists** checkbox is selected, the service in the service repository will be over written by the one in the zip file, otherwise conflicting services are not imported from the local disk.

Chapter 4: Mapper Perspective

The eStudio incorporates Eclipse based Fiorano eMapper as a separate perspective. To open Fiorano eMapper, perform the following steps:

1. Click the **Open Perspective**  button from the shortcut bar on the left-hand side of the Workbench window.
2. Select **Other...** from the drop-down menu.
3. Select the **eMapper Perspective** to open Fiorano perspective. Click **OK** button. The eMapper perspective containing Project Explorer and Funclet View appears.

More information on eMapper is present in [Chapter 10 eMapper](#).

Chapter 5: Composing Event Processes

Composition of Event Processes is based on component-based programming model. An Event Process is composed of services (also known as Business Components) linked to each other by Data Routes.

The Event processes are designed by drag-drop-connect function of service components. The components are customized by configuration rather than by custom code. The routes between components are drawn by visually connecting the component ports. Every component instance in the flow can be configured so that it can be deployed on different ESB network nodes.

The following sections describe how to compose an Event Process, adding remote service instances and adding external Event Processes. The sample Event Process illustrated below connects Fiorano Chat Business Components with bidirectional Event Routes. The two instances will be configured to run on different nodes in the network.

5.1 Adding Components

To add components, perform the following steps:

1. Open the **Service Palette** and click the **Category** tab (**Collaboration**) corresponding to the service.
2. Drag and drop the business component icon (**Chat**) onto the **Event Process** editor.

Each icon in the Event Process editor represents an instance of the service. By default, the name of each instance of the service is the service GUID followed by the instance ID count. The Service instances can be renamed if required.

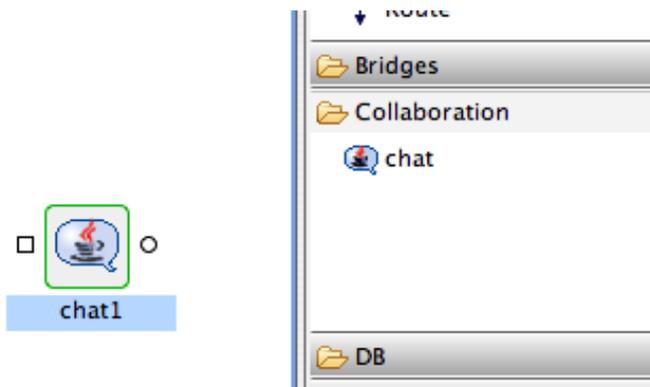


Figure 5.1.1: Adding components

5.2 Connecting Routes

For the data to flow between two service instances, they need to be linked through Event Routes. The Route represents the Brokered Peer to Peer data Route.

The Event Routes are unidirectional and always originate at output Event Port of the source service and end at input Event Port of the target service.

Connect the Route from the output channel (OUT_PORT) of Chat1 service icon to the input channel (IN_PORT) of the Chat2 service icon and vice versa, as shown in the Figure 5.2.1. By default, each Route is identified by an Event Route name such as; Route1 and Route2. The suffix represents the instance count of the Route. You can edit the Route name using the Properties window.

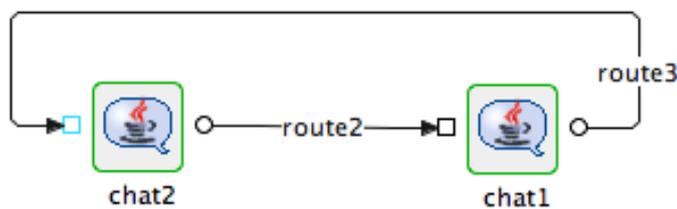


Figure 5.2.1: Connecting components through routes

5.3 Configuring Components

All the services contain configuration information that can be provided in the Custom Property Sheet (CPS) dialog.

To review the custom property sheet associated with any component, double-click the component in the event process editor.

A Sample Database component CPS is shown below, containing all the details of the Database connection, SQL, and so on. Sample CPS is shown in Figure 5.3.1.

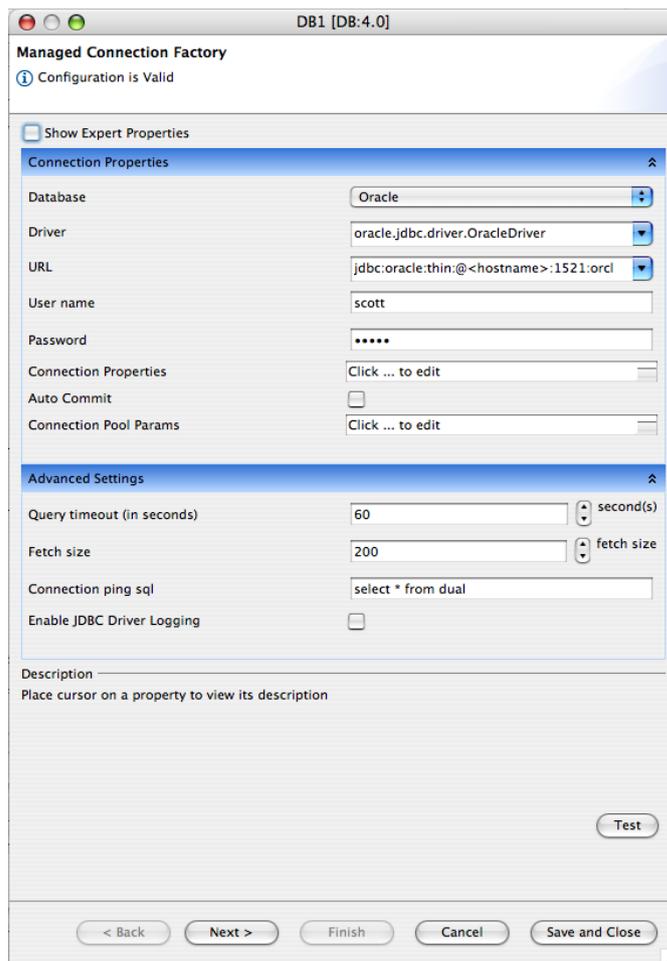


Figure 5.3.1: DB Custom Property Sheet (CPS)

During configuration, clicking the **Test** button provided in the CPS can also test the configuration.

Components configurations are saved in `EventProcess.xml` file. This file is in a simple XML format.

Service instances contain configuration information that is used for execution at runtime. The data flows from service instances through connected routes.

5.4 Configuring Component Properties

Apart from the component specific properties that can be configured using the CPS, there is a set of properties associated with every component. These properties are shown in properties view when a component is selected. The properties are categorized into various sections as shown in Figure 5.4.1

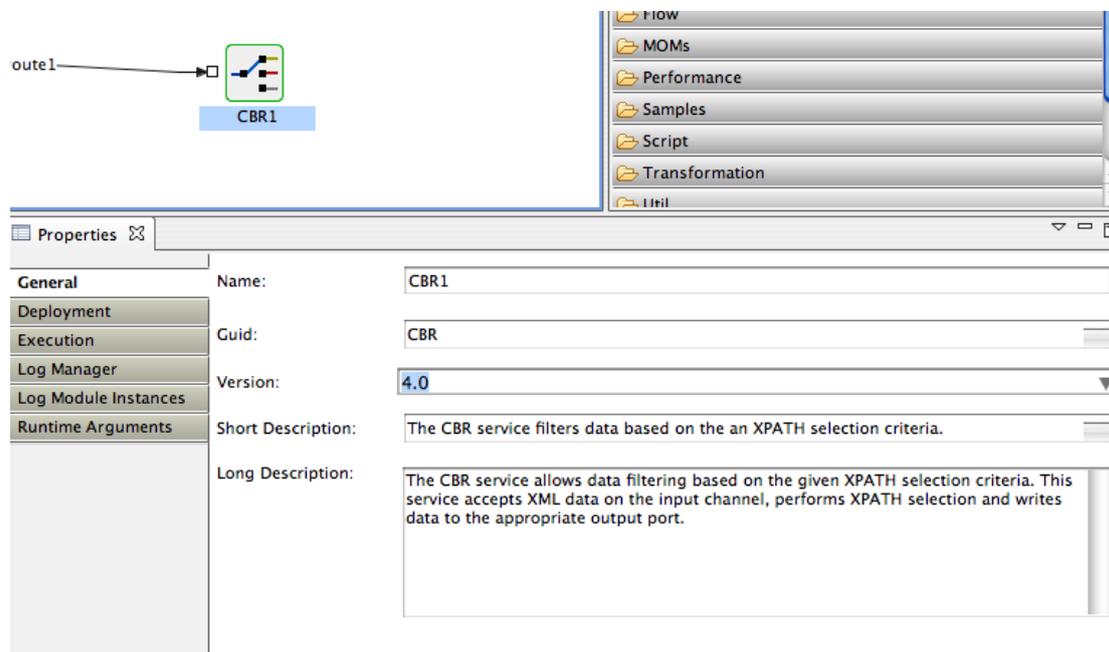


Figure 5.4.1: Component properties

General: Contains the general information of the service likeName, GUID, Version Short Description, and Long Description.

Deployment: Contains the deployment information of the component. The Peer Server node on which the component has to be launched can be configured here.

Clicking on ellipsis button against the Nodes property opens the **Select Nodes** dialog box where the Peer Server can be selected.

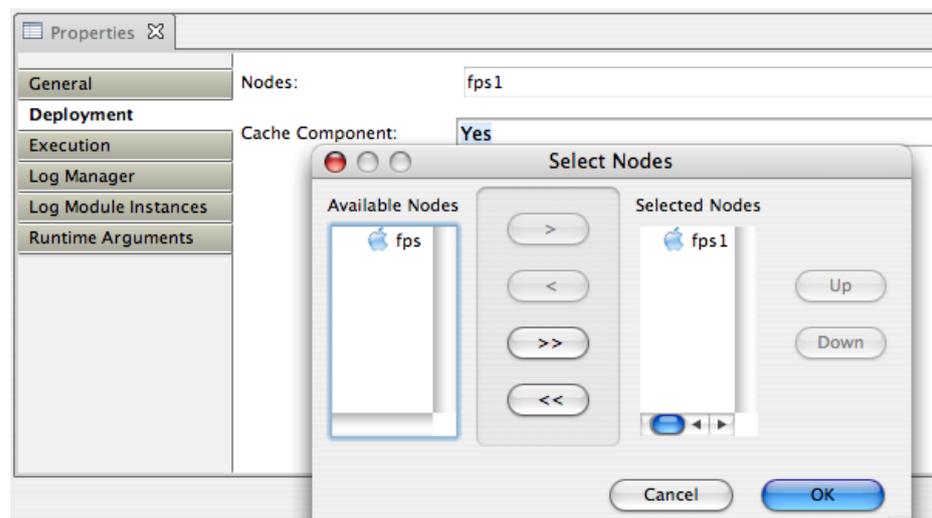


Figure 5.4.2: Component ports

Based on the selected Peer Server, the component color changes to give a visual clue as to which Peer the component is configured to launch. As shown in Figure 5.4.3, the Feeder component is configured to launch on fps Peer and CBR is configured to launch on Peer fps1.

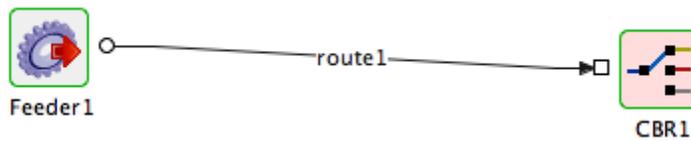


Figure 5.4.3: Components configured to launch on different Peer servers

By default, when a Peer Server is added to an Enterprise Server, a unique color is chosen. The user can customize this color using colors from Peer properties by selecting the Peer Server in Peer Repository.

A property called Cache component specifies whether component resources have to be re-fetched each time when Connectivity and Resource Check (CRC) is done. When Cache Component is set to yes, the resources are fetched for the first time when CRC is done. This property is set to **No** only when the component resources have been updated.

This property is also available at the Event Process level.

Execution: The Execution section contains information about the launch type, connection factory properties, and so on. Components can be launched in Separate Process (separate JVM for each service instance), inMemory (launches in Peer Server JVM), Manual (manual launch mode where the user has to launch the service instance manually) and None (no launch mode) modes.

Log Manager: Contains logging information like the type of Logger Handler, log directory, and so on.

Log Module Instances: Log levels for various loggers available for the service can be configured in this section.

By default the log level is set to SEVERE. This can be changed to the desired level. For example, the log level can be set to CONFIG when working on the Development environment.

Name	Value	Append Service I
All Levels	<Different Values>	Yes
com.fiorano.services.display.DisplayService	WARNING	Yes
com.fiorano.services.display.transport.jms	WARNING	Yes
com.fiorano.services.display.engine	WARNING	Yes
ERR_HANDLER	SEVERE	Yes
OUT_HANDLER	INFO	Yes

Figure: Component log module instances

When the component is launched all the component log module instance names are modified to include Event Process name, Service Instance name and Service GUID to maintain uniqueness.

For example, if the log module instance name of a component is `com.fiorano.services.display.DisplayService`, when the component is launched some extra information is included and is changed to `EVENT_PROCESS1.COM.FIORANO.SERVICES.DISPLAY.DISPLAY1.DISPLAYSERVICE` and the logger is created with the changed name.

In the above example the log module instance name has the service guid name `Display`.

If the original name doesn't have Service GUID name, then Application Name, Service GUID and Service Instance name is pre-pended to the log module name.

For example if the log module instance name is `org.apache.ws.security.processor.SignatureProcessor`, when the component is launched then it is changed to `COMMUNICATIONS_IN.WSSECURITY.WSSECURITY1.ORG.APACHE.WS.SECURITY.PROCESSOR.SIGNATUREPROCESSOR` and the logger is created.

Changing the log module instance names at component startup is required to maintain uniqueness of components logs especially when multiple instances of a component are running in the same JVM.

However, this does make logging from 3rd party jars difficult to manage when working with custom components. To avoid this the property `Append service Info` is useful for controlling log levels of loggers used by third party libraries, since the handlers for such loggers will be created without appending service info to the logger names used by them.

Note: This property should not be set to yes for default loggers used by Fiorano pre built components and loggers generated by default in custom components.

Runtime Arguments: Contains the information about the runtime arguments for the service.

JVM_PARAMS section contains the JVM parameters that are used while launching the component. Whenever a change is made in JVM PARAMS section, the Update all Service Instances dialog box appears asking whether the change has to be updated for all the service instances in all Event Process having the same JVM PARAMS value.

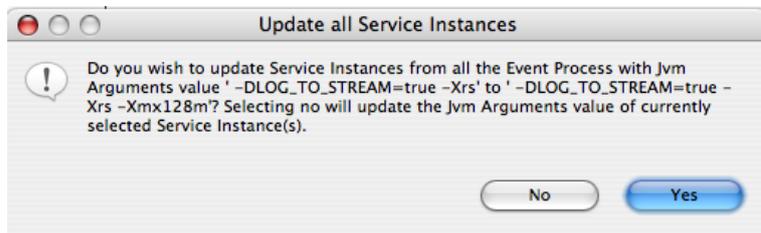


Figure 5.4.4: Update all Service Instances dialog

If **No** option is selected, then it updates to the current service instance. If **Yes** option is selected, a dialog listing the service instances with same JVM PARAMS value appears and the required service instances can be selected for an update.

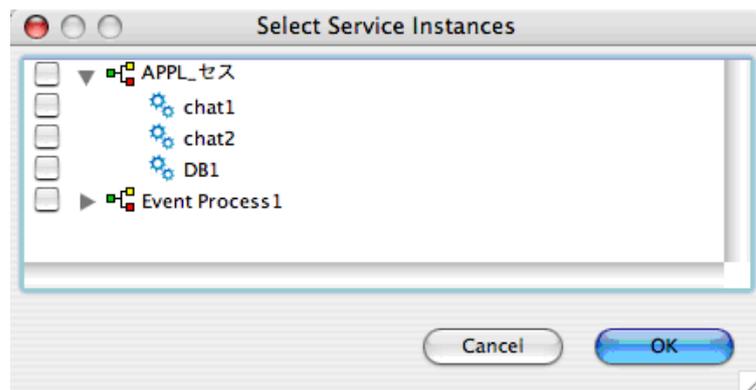


Figure 5.4.5: Select Service Instances

5.5 Adding Remote Service Instance

The Fiorano SOA Platform allows you to compose an Event Process with Business Component instances from other Event Processes. The Remote Service instance is one of the available options for communication between different event processes. If the *producer* component is in a *calling* event process, then the *producer* component needs to send messages to the *consumer* component in a *called* event process, then a remote instance of the *consumer* component can be used in the *calling* event process.

The imported service instance is the reference to the service instance in the parent Event Process. Any changes made to the imported service instance in the parent Event Process are reflected in the current Event Process. Current Event process can be launched only when the Event Process of the remote service instance is running.

To add a remote service instance, perform the following steps:

1. Click the **Insert Element into Event Process**  icon and select **Insert Remote Service Instance** option (or) right-click on orchestration editor and select **Insert Remote Service Instance**.



Figure 5.5.1: Insert Remote Service Instance option

The **Select Remote Service Instance** wizard starts, as shown in Figure 5.5.2.

This dialog box lists all the Event Processes and their service instances.

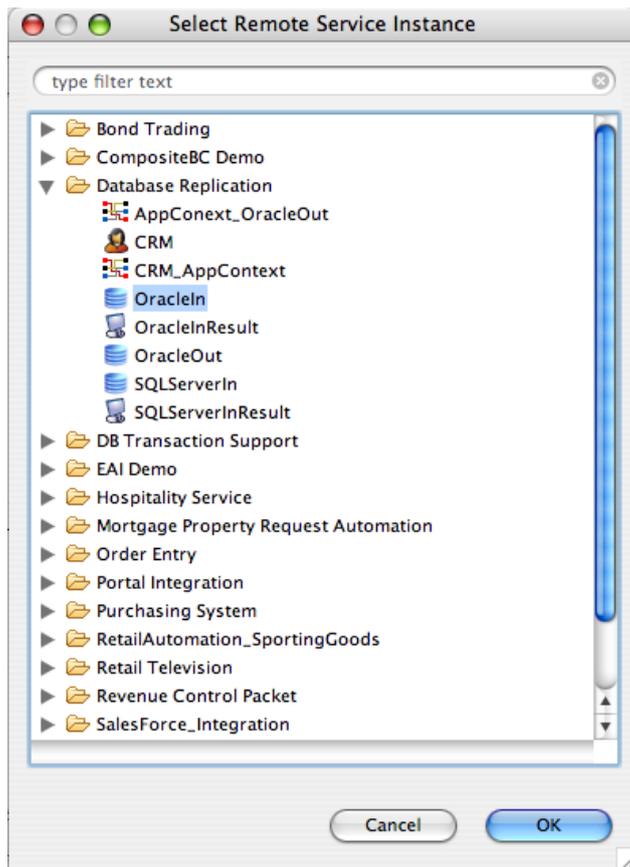


Figure 5.5.2: Select Remote Service Instance dialog

2. Select the service instance you want to add as Remote Service Instance and click the **OK** button.

The Remote service is added to your Event Process with a satellite like icon in the component as shown in the Figure 5.5.3.

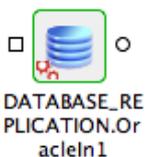


Figure 5.5.3: Remote service added

A Remote Service instance can be used in a similar manner to normal a service instance. Routes can be created between other service instances in the Event Process and the ports of the Remote Service instance.

Note: While using Remote Service instance with Event Process Life Cycle Management (EPLCM), and if a component is running in a configured mode (say Testing) in the parent Event Process and if this component is used as a Remote Service instance in a caller Event Process, then changing the mode in the caller Event Process will not have any effect. It still uses the mode used in parent Event Process.

5.6 Adding External Event Process (Subflow)

Subflow concept is used to ease the Event Process development when composing large Event Processes. When an Event Process B is copied into another Event Process A, all the data (service instances, routes etc) in B is copied and shown as a single entity (icon) in A. By default, the icon takes the name of the added Event Process (that is, B). When we double-click on the icon it shows all the service instances/routes and so on. The ports of the inserted Event Process B can be exposed for communication with Event Process A.

The External Event Process is explained with an example. Steps to add EAI_DEMO Event Process in Simple Chat Event Process are explained below:

1. Open an Event process (Simple Chat) and click the **Insert Element into Event Process**  icon and select the **Insert Event Process** option from the drop-down list (or) right-click on the Orchestration Editor and select **Insert Event Process**.



Figure 5.6.1: Insert Event Process option

2. The **Select Event Process** dialog box appears as shown in Figure 5.6.2. Select the Event Process from the list and click the **OK** button.

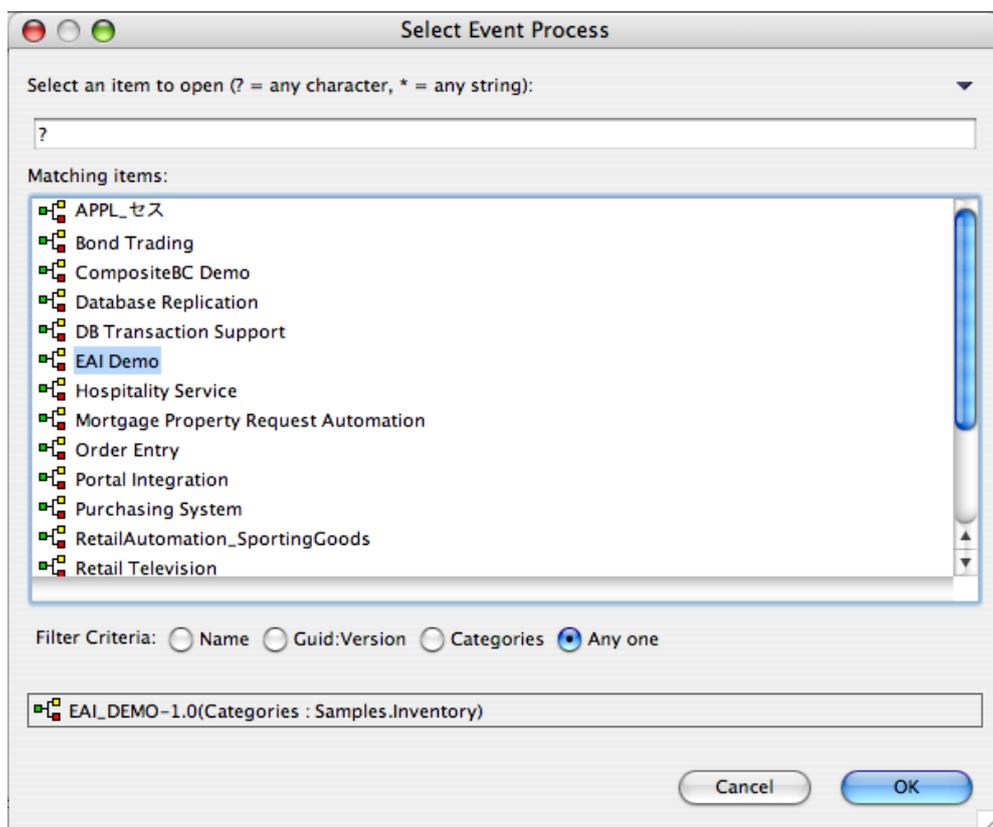


Figure 5.6.2: Select Event Process dialog

3. The Event Process instance representation appears on the **Event Process** editor, as shown in Figure 5.6.3.

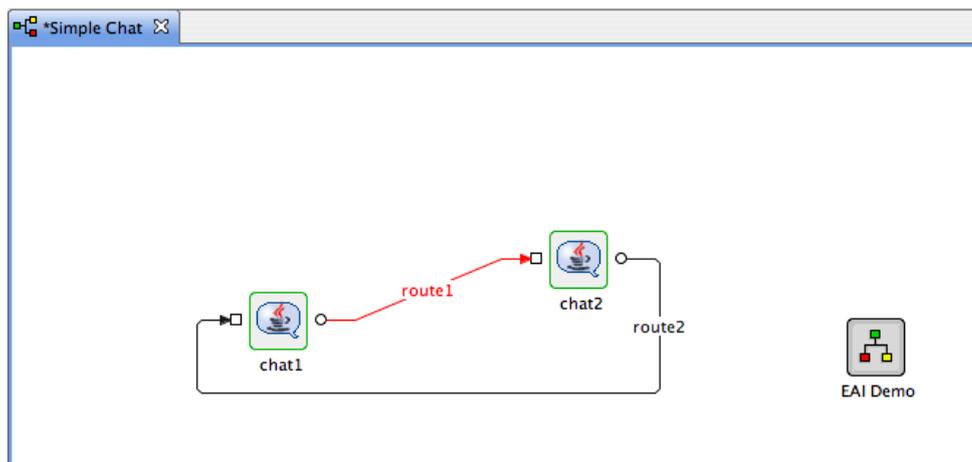


Figure 5.6.3: Inserted Event process

Note: This concept is different compared to the Remote Service Instance. In the Remote Service Instance, remote instance will refer to the original instance but in this case a copy of the selected Event Process is made and used in the Event Process. This is basically a visual representation that makes the composition easier when working with large event processes.

4. Double clicking the EAI Demo icon shows all the service instances and routes inside as shown in Figure 5.6.4.

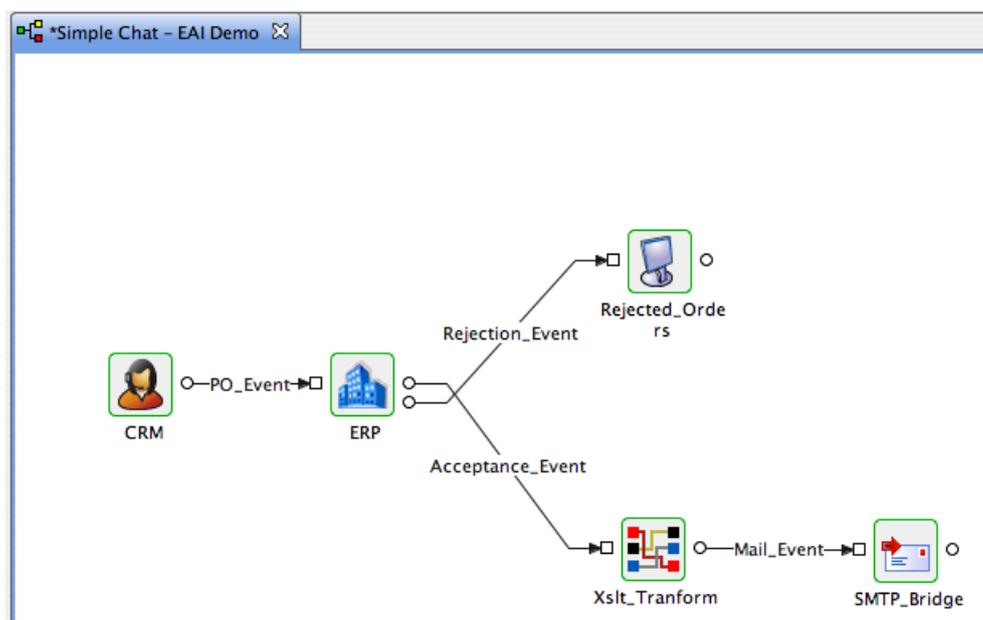


Figure 5.6.4: Component ports

By default, no input and output ports are shown for an inserted Event Process instance. The user can expose the required input and output ports of service instances present in the Event Process instance from the Properties tab as shown in Figure 5.6.5.

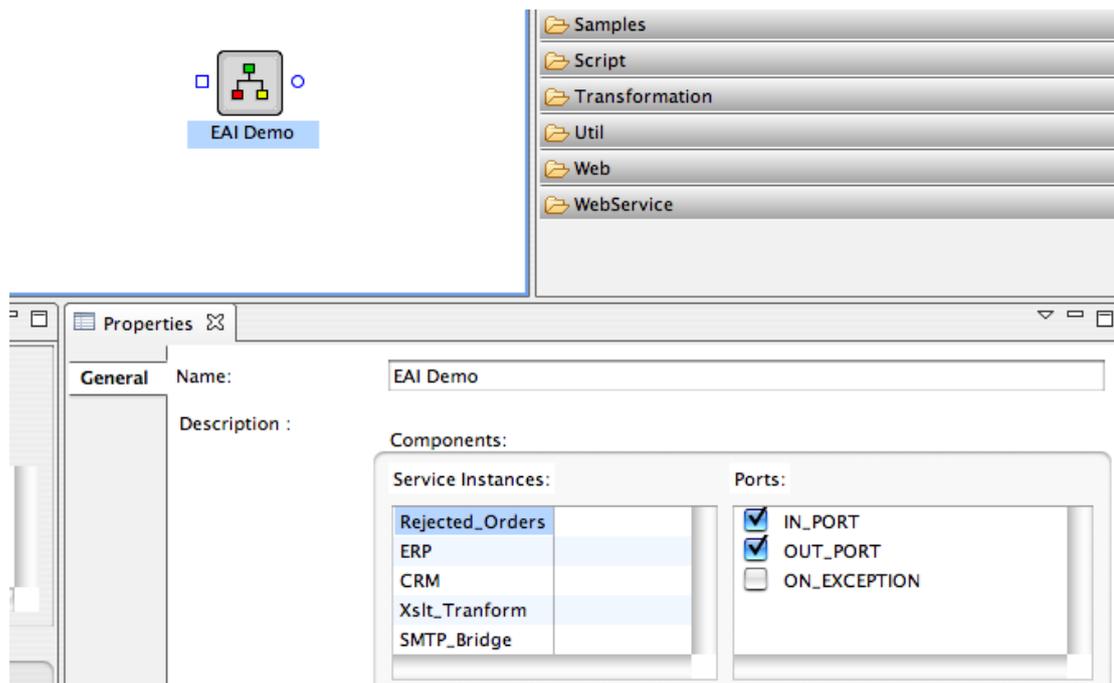


Figure 5.6.5: Properties tab

- Routes can be connected between other service instances in the Event process and subsequently inserted into the Event Process. This allows the connected service instances to communicate with each other.

5.7 Document Tracking

A workflow in Fiorano terminology consists of an entry point, an intermediary points and an end point. The entry and intermediary points are defined as Workflow Items and the end point is defined as a Workflow End.

To track the documents going through the Service Instances, document tracking can be enabled on service instance ports. If tracking is enabled, the documents that pass through that port are stored in a database. By default these documents are stored in the H2 database that runs inside the Enterprise Server. It is recommended to use an external database for document tracking. An external database can be configured for document tracking by providing the database configuration details in *sbwdb.cfg* file located in the Enterprise Server profile.

A workflow starts with Workflow Items and ends at Workflow End. A workflow is defined within an Event Process scope through which a large number of documents pass. Whenever a new document enters into the workflow, a new workflow instance is generated. Each workflow instance has a unique ID assigned by the Fiorano SOA environment. In a state enabled workflow, all the states that these workflow instances traverse are stored for tracking purposes.

Each workflow instance contains information about documents that pass through. Each time a document passes through a trackable state, a state event is generated and the document is given a new Document ID by that trackable state. Information related to the documents can be viewed in the Fiorano Web Console.

eStudio provides a state-based workflow view that enables tracking and monitoring of documents from one state to another.

To enable document tracking in an Event Process, perform the following steps:

1. Select the Service Instance Port on which document tracking has to be enabled. The Properties pane appears (if the Properties pane does not appear, go to Window->Show View-> Others-> and select Properties).
2. To enable the **Enable Document Tracking**, select **Workflow Item/Workflow End** option in the Workflow property drop-down list as shown in Figure 5.7.1.

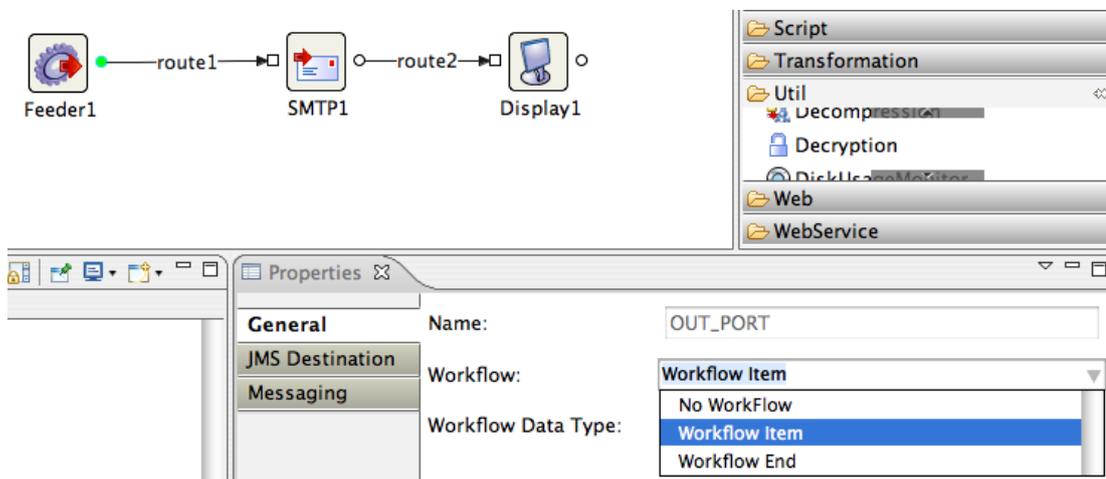


Figure 5.7.1: Enabling Document Tracking

3. In the sample Event Process shown below, the workflow starts at Feeder Output port. The SMTP Output port is marked as an intermediary point and the workflow ends at Display Input port.

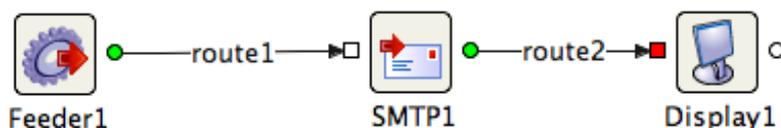


Figure 5.7.2: Event Process with Document Tracking enabled

4. Workflow items are filled in with the color **green** and Workflow End is shown in the color **red**.

In the Event Process, the state tracking is enabled for Feeder1 output port, SMTP1 output port and Display1 input port. All the messages which pass through these are tracked.

The default Workflow data type is set to Message Body. This implies that only the JMS message body is tracked. This can be configured by clicking on ellipsis button against the Workflow Data Type property to track Message Header, Message Body, Attachments, Application Context or all of these items.

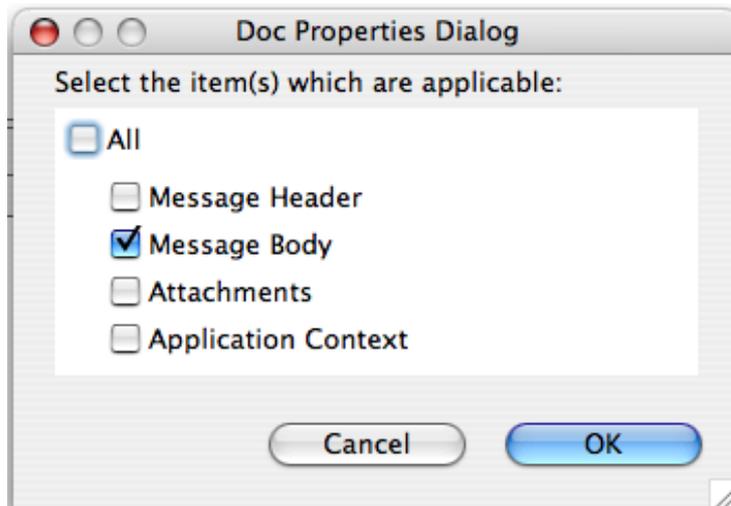


Figure 5.7.3: Document Tracking Properties

5.8 Defining Route Transformations

In addition to XSLT component, transformations can also be defined on routes having schema mismatch. In the example shown below, there is a schema mismatch between Feeder output port and CBR input port and hence the route is shown as dotted line.

A route transformation can be defined in any one of the following ways:

- Defining a Mapper Project,
- Providing a Custom XSL, or
- Using a Named Configuration

Defining a Mapper Project allows the user to define mappings between the source and target port schema using Fiorano eMapper.

To define the transformation using eMapper, perform the following steps

1. Right-click on the route and select Mapper Project option in Configure Transformation sub menu.

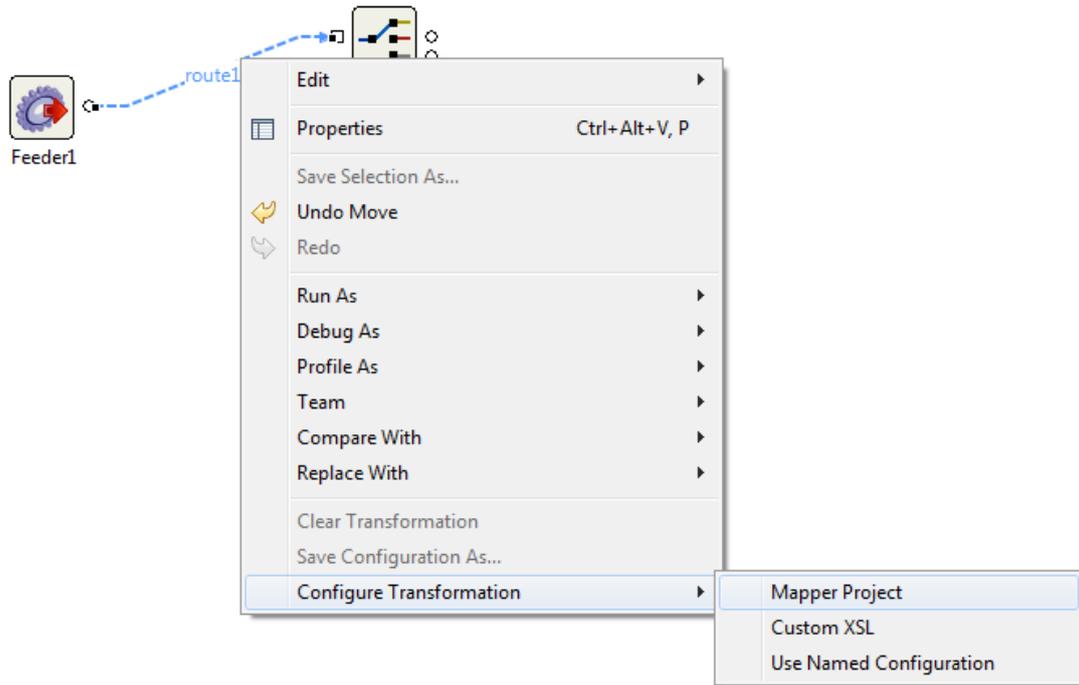


Figure 5.8.1: Route Transformation

2. Route transformation editor will be opened which automatically picks up the schemas on connected ports.

Transformations can be defined on the schemas by connecting elements from input structure with output structure elements. Additional computations on elements can also be made by using functions present in Funcllet view.

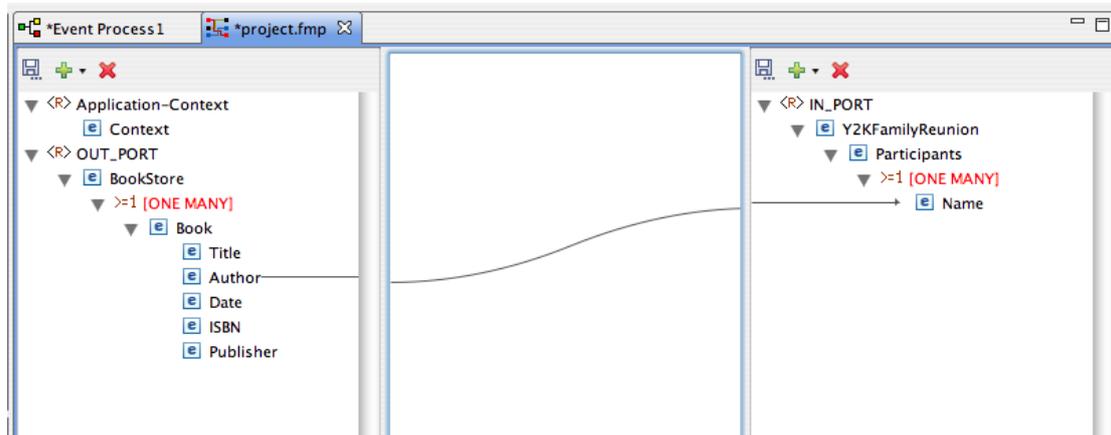


Figure 5.8.2: Route transformation editor

Note: While the editor is open for defining transformations, the Event Process editor will be in non-editable state to prevent further changes till the transformation editor is closed. This becomes editable when the transformation editor is closed.

3. Once the transformation is defined and closed, transformation will be set on the route and the route is shown as bold.

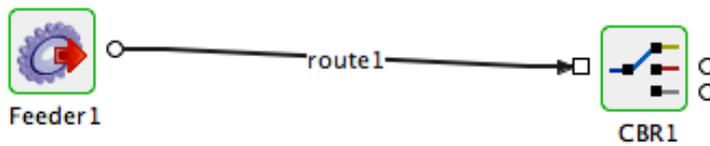


Figure 5.8.3: Route with transformation set

To provide a Custom XSL, perform the following steps:

1. Right-click on the route and select **Custom XSL** from Configure Transformation menu.
2. The Custom XSL Dialog is opened. Provide the XSL in the first tab and XSL for the JMS Message (if any) in the second tab. Click **OK** to set the transformation on the route. The XSLT Engine to be used (Xalan / Saxon / XSLTC) can be specified from the drop-down in the top right corner.
3. The XSL provided can be tested by clicking the **Test** button.

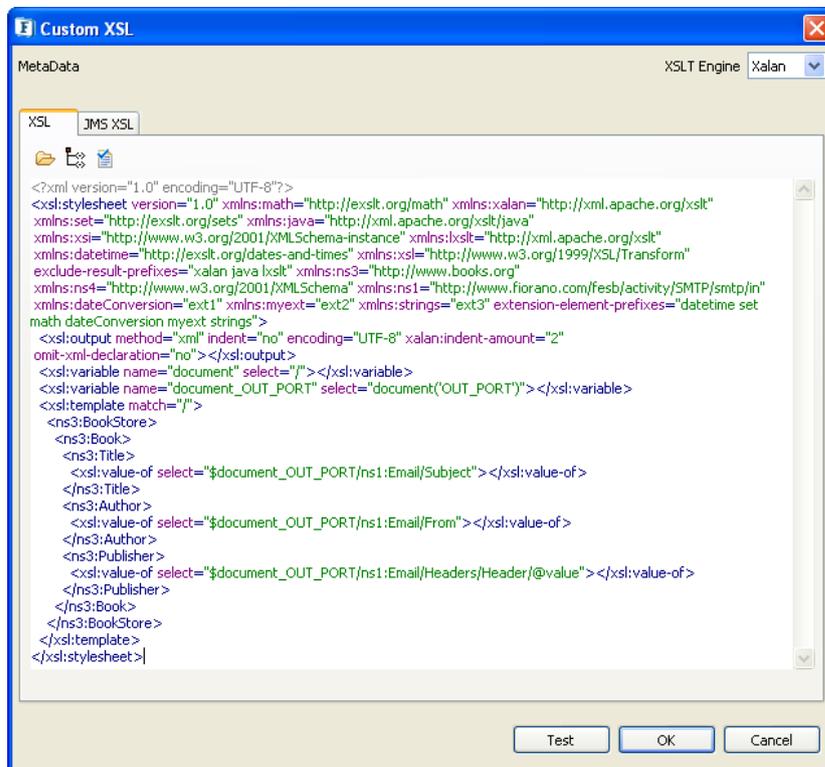


Figure 5.8.4: Custom XSL Dialog

Transformations can also be saved in the **Configuration Repository** as Named Configurations and can be used wherever needed. Creating a transformation Named Configuration is explained in detail in Section 15.8.

To Use a Named Configuration as a route transformation, right click on the route and select **Use Named Configuration**. Choose the configuration to be used from the list of available named configurations.

The transformation defined on route is executed inside the Peer server. So it is advised to not to define complex mappings (involving huge schemas and mappings) using Route Transformations since it may affect the Peer server performance. For complex transformations XSLT component can be used.

The transformation defined on route can be cleared by selecting **Clear Transformation** option in the right-click menu option on the route.

Route transformation can be changed on the route at both configuration time and runtime. During runtime if the transformation is changed, the changes are automatically deployed to the server. The user need not explicitly synchronize the event process for the changes to take effect.

5.9 Configuring Selectors on Routes

eStudio allows you to define Selectors for the data flow through an event route. Take an example of an Event Process containing two instances of a Chat service and an instance of a Display connected through routes as shown in Figure 5.9.1.

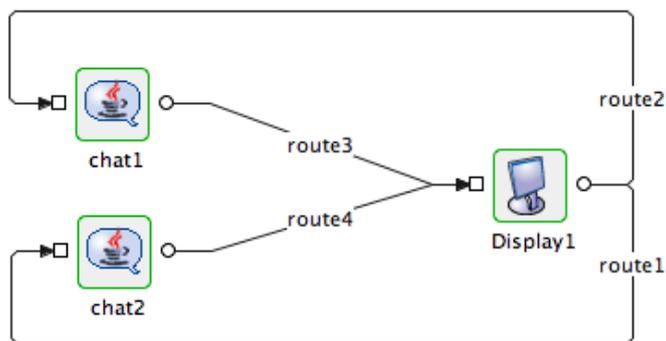


Figure 5.9.1: Out-port of chat1 and chat 2 to display in-port, out-port of display to in-port of Chat1 and chat2

In the above event process, the event routes exist as defined below:

- **Route1:** Connects OUT_PORT of Display1 to IN_PORT of Chat2
- **Route2:** Connects OUT_PORT of Display1 to IN_PORT of Chat1
- **Route3:** Connects OUT_PORT of Chat1 to IN_PORT of Display1
- **Route4:** Connects OUT_PORT of Chat2 to IN_PORT of Display1

When the Event Process is launched, if a message is sent from Chat1 component, it is received by Display component and the message is sent back to both Chat1 and Chat2 components from Display output port.

Now, let's define conditional data flow from the Display business component instance. Assume that Display1 has to send only those messages on Route1 which are sent from Chat2. Similar conditions should also apply to Chat1 that it should also receive only those messages that it sends to Display1.

To define conditional data flow through route1, perform the following steps:

1. Select **Route 1**, the properties of this route is displayed in the **Properties** tab.
2. In **Selectors** tab, choose the option chat2 from **Sender** properties as shown in Figure 5.9.2.

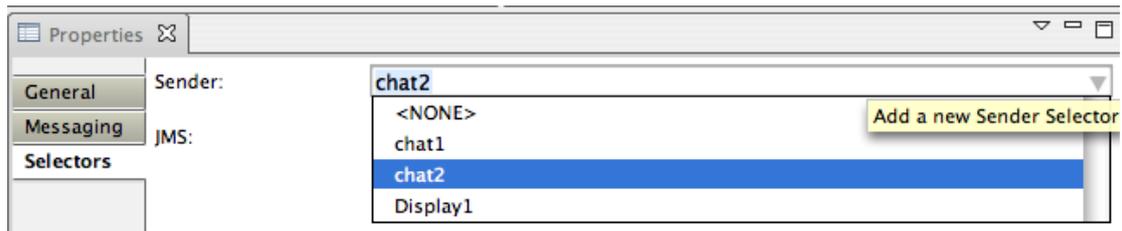


Figure 5.9.2: Configuring Selectors on route

This ensures that data sent only by Chat2 will travel through this route. Similarly, set this value to Chat1 for Route2. This ascertains conditional flow of data. After the changes, if the event process is launched, messages sent by Chat1 are received only by Chat1 and messages sent by Chat2 are received only by Chat2.

Apart from Sender selector, the JMS selector can also be defined which checks for a particular value for a JMS message property and routes the message.

5.10 Configuring Application Context

There are times when a target Event Port needs information that was produced by a service instance that occurred earlier in the workflow. Consider an event process representing a ten-step business process. Each step is implemented using a service instance. By using application context, a service instance representing the tenth step in the process can use the information generated by the service instance in the second step.

Application context is set as a JMS Message Property on the message and is available throughout the function.

To define Application Context for an application, perform the following steps:

1. In the **Event Process** project, click on the **Orchestration Editor** and open the **Properties** view.
2. Click the **Application Context** tab in the **Properties** view.
3. To define an Application Context for the Event Process, enable the **Application Context** option.
4. Select DTD/XSD option and provide the schema content.
5. Click the **Save Content** button to save the changes.
6. Select root element from the list of available roots in the **Root** drop-down list. Now the application context schema is defined for the Event Process.

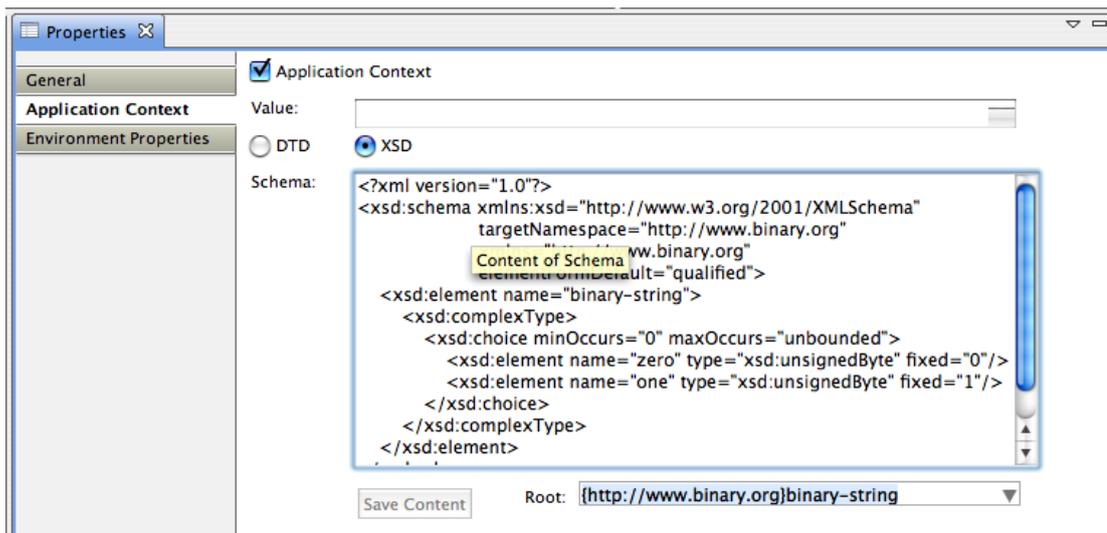


Figure 5.10.1: Application Context option

7. The default value of Application Context defined can be provided in the **Value** section. If not provided here, this can be defined in the context menu on the Output port of a component or using an XSLT component.
8. To define from Output port option, Right-click on the Output port and select **Application Context** option.

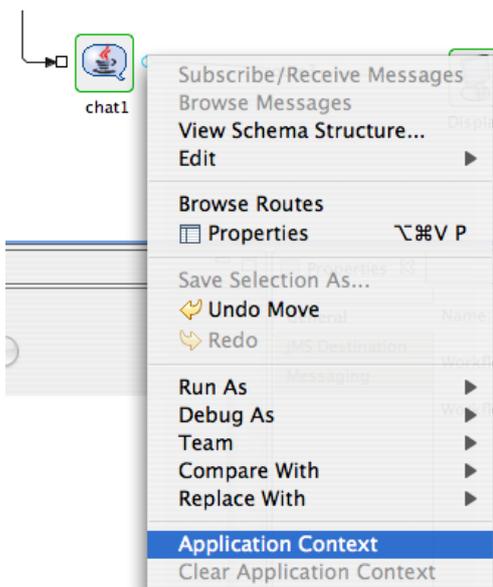


Figure 5.10.2: Application Context on output port

9. The Mapper editor opens up where the mapping for the Application context can be defined. Save the mappings. The port figure will be shown in bold font to give a visual representation.

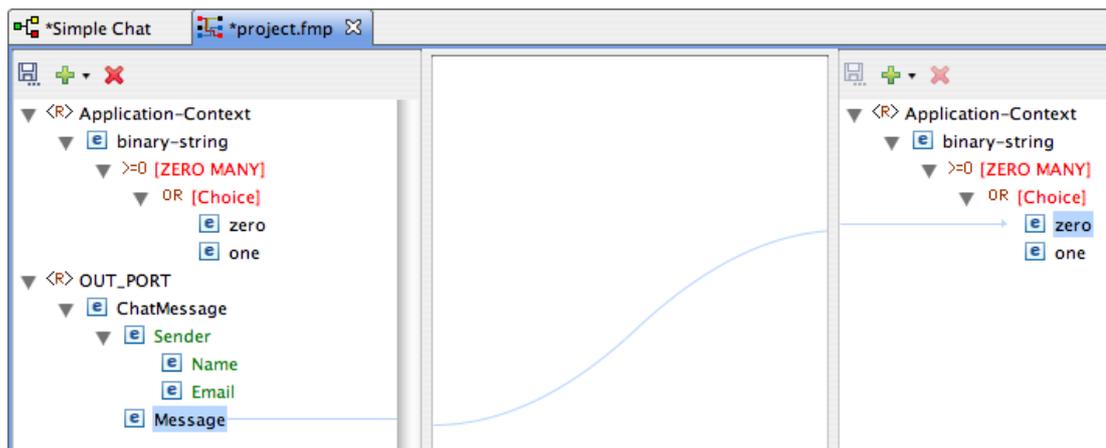


Figure 5.10.3: Mapper editor

10. Once Application Context is configured at one of the out ports, the value is propagated in the message flow.
11. The application context can be used anywhere in the event process using Xslt component or Route Transformation.

5.11 Check Resource and Connectivity

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 of the grid-nodes (Fiorano Peers) on which the components are to be deployed.

To do a connectivity and resource check, perform the following steps:

- Select the Event Process. Click the **Check Resource Connectivity**  button from the tool bar as shown in Figure 5.11.1.
- All the resources required by the component at runtime will be deployed to the configured Peer Server.

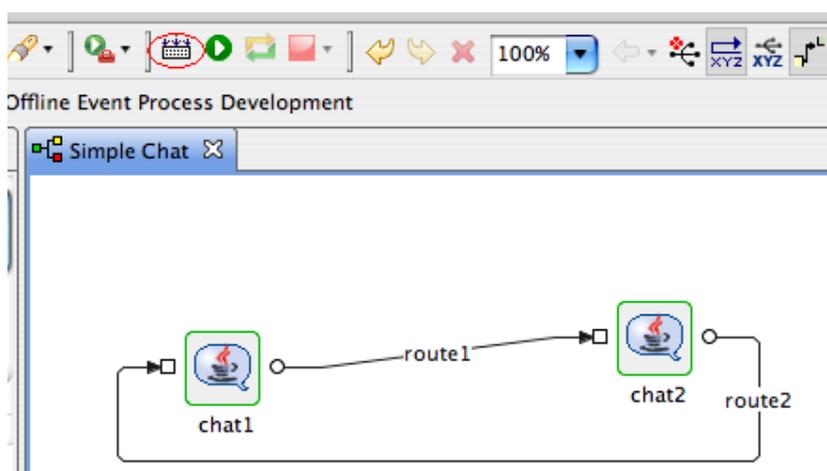


Figure 5.11.1: Check Resources and Connectivity

During the development process, 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, eStudio has an option; Cache Component in its Properties view, a deployment configuration at both the Event Process as well Component levels, that optionally force the resources of the component to be re-fetched each time a Connectivity and Resource Check is done.

5.12 Running Event Process

To run the event process, perform the following steps:

- Select the Event Process. Click the **Launch Event Process**  button as shown in Figure 5.12.1.

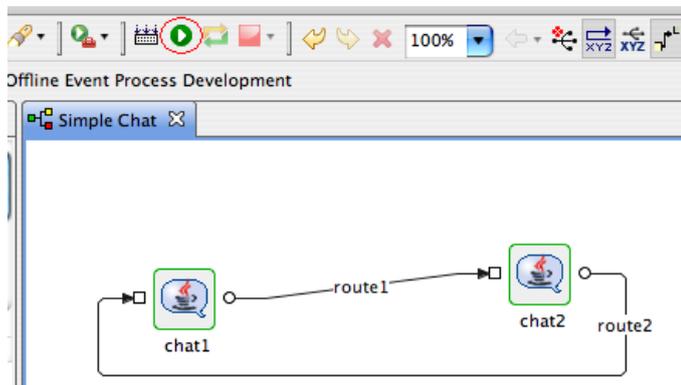


Figure 5.12.1: Launching an Event Process

When the Event Process is launched successfully, all the service instances label names turn green in color.

5.13 Stopping an Event Process

To stop the event process, perform the following steps:

- Select the Event Process. Click the **Stop**  button on the toolbar; all running component instances in the Event Process are stopped and the Event Process is stopped.

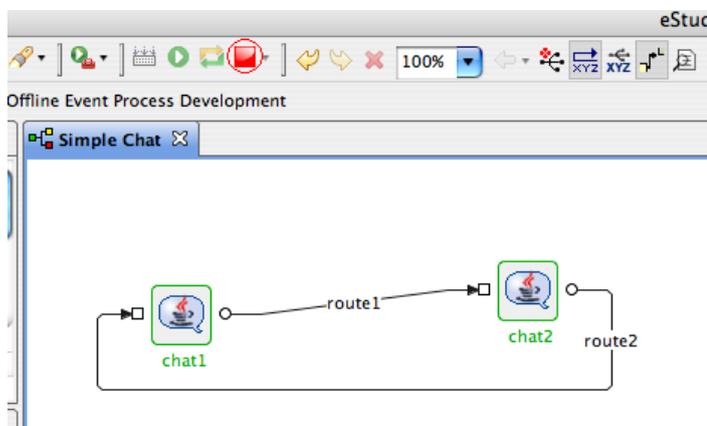


Figure 5.13.1: Stopping an Event Process

Stop button can also be used to stop selected service instances or all service instances without stopping the Event Process. When a component is stopped, its name turns to red in the orchestration editor.

5.14 Synchronizing an Event Process

Fiorano has the capability to modify existing running applications on the fly. For example, launch the Simple Chat event process and once the event process is successfully launched, add another component-instance to the event process from the component palette. Configure the component and connect the routes. The application then needs to be synchronized to reflect the changes.

To synchronize event processes, perform the following steps:

- Select the Event Process. Click the **Synchronize Event Process** button. Now the newly added component starts and turns green in color and the synchronize button is in a disabled state.

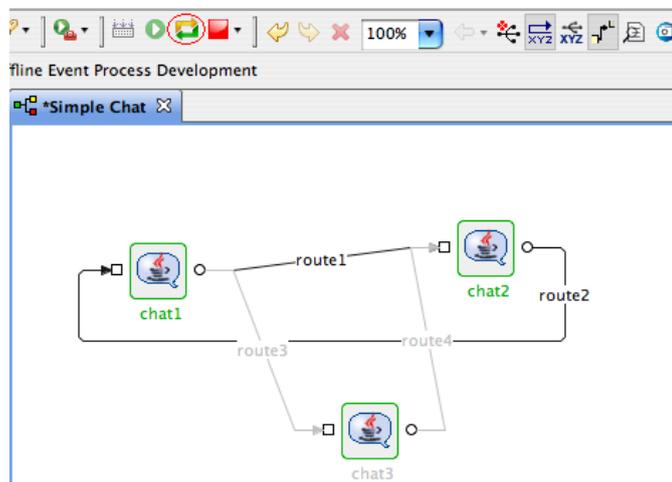


Figure 5.14.1: Event Process Synchronization

Chapter 6: Event Process Life Cycle Management

The Event Process Life Cycle Management refers to deployment of an Event Process in various environments like Development, Testing, Staging, and Production. The user does not have to create different Event Processes for different environments; instead the user can simply specify the properties for service instances comprising Event Processes for various environments in a single Event Process.

6.1 Setting Properties of Service Instances for Different Environments

When a Service instance is dragged and dropped in the Orchestration editor, the default environment is set to development. To configure a different environment, select the **Target Environment** in the Environment Properties tab of the Event Process comprising the service instance. Hereafter, the environment dependent service properties will be written to the corresponding `env.xml` file and will be picked up from that file when Event Process is launched in that particular environment.

You can specify these properties for more than one environment by switching the Target Environment label in the properties of an event process. Configuring service instances for different environments is made easy, since the configuration properties of the service instance in a new environment will be picked up from the previously configured environment when the CPS is opened.

This way service instances can have different set of properties while running on different environments. For example a File Reader instance can be configured to read from a `dev.txt` file in a Development environment and from a `test.txt` file in a Testing environment.



Figure 6.1.1: Environment Properties Tab

6.2 Running Event Process on an Environment

To run an Event Process on a particular environment, follow the steps as mentioned in the example below:

1. Take a flow containing File Reader and Display components. Configure the File Reader providing different inputs in different environments as mentioned in the above section. Select the **Target Environment** in the **Environment Properties** tab of the Event Process. The environment specific properties for the service instances in the flow can be viewed from the Environment Properties table view present below the Target Environment section.
2. Do the **CRC** and launch the flow. When the flow is launched in development environment, the contents from the dev.txt will be read and these messages can be viewed in the display. Similarly when launched in testing environment the contents from the test.txt will be read and these messages can be viewed in the display.

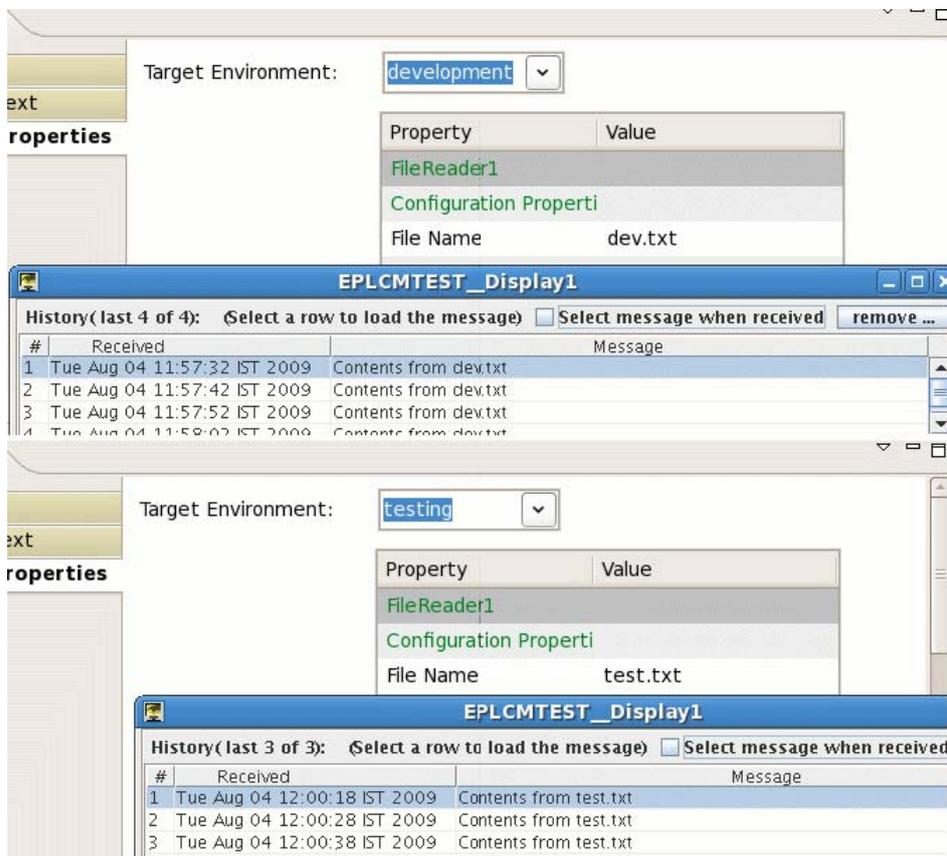


Figure 6.2.1: Display showing messages received from file reader in different environments

Note: These properties cannot be edited from the table provided. But they can be edited from the CPS of the specific service instance and from the deployment tab of the service instance in the properties view.

Chapter 7: Debugging Event Process

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 service instances within Event Processes and also provides a mechanism to setup *event interceptors* to capture, view, modify and discard messages flowing between service instances on the same or different machines across the network.

Note: Breakpoint can be added in Online Event Process Development perspective only.

7.1 Adding Breakpoint

Breakpoint can be added from context menu present on the route or from the Fiorano Debugger view.

7.1.1 Context Menu option

Right-click on the route on which breakpoint has to be added and select the **Add Breakpoint** option.

When the breakpoint is added, the route color is changed to Red.

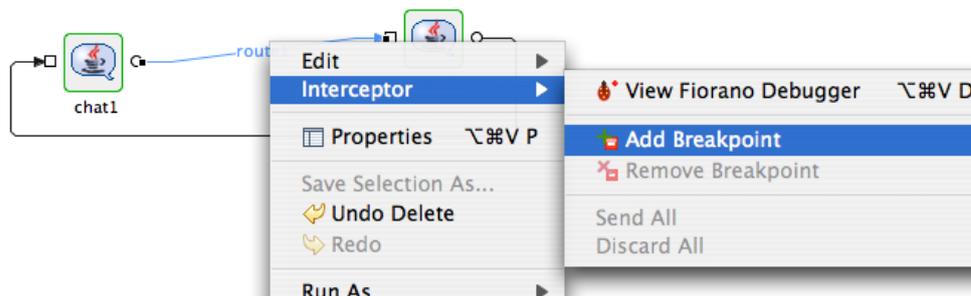


Figure 7.1.1: Adding breakpoint from context menu

7.1.2 Debugger View

To add a breakpoint to a route, perform the following steps:

1. Go to Fiorano Debugger pane and click the **Add BreakPoint** button as shown in Figure 7.1.1. All the available routes in the Event Process are listed as shown in Figure 7.1.2.

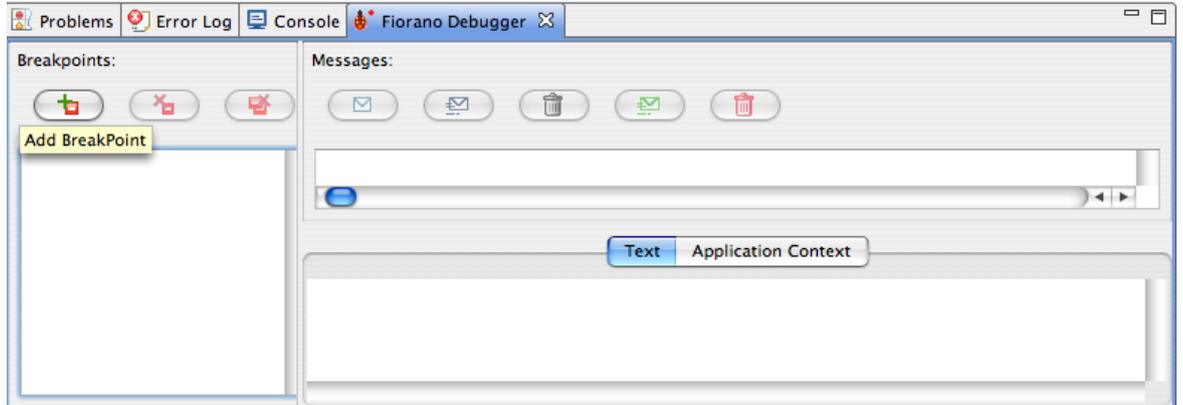


Figure 7.1.2: Adding break point from debugger view

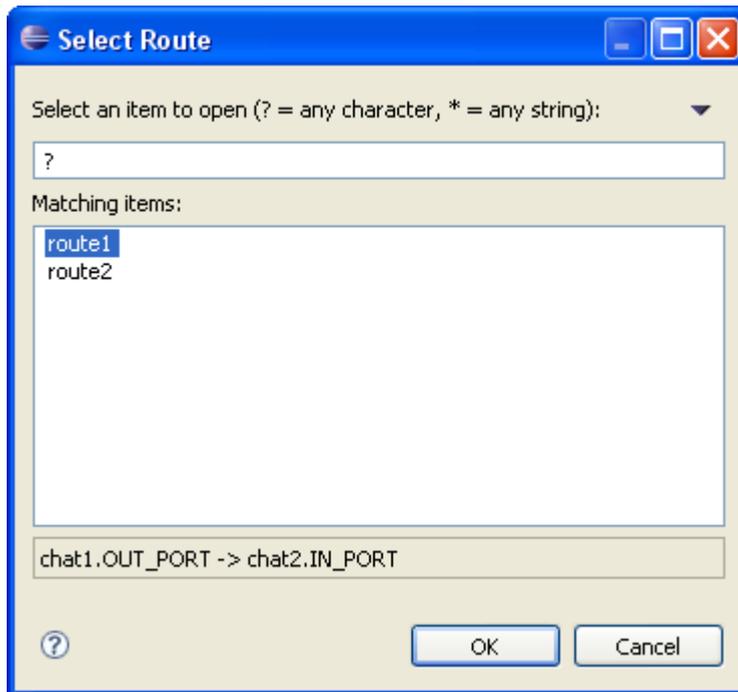


Figure 7.1.3: Select route to add breakpoint

2. Select the route on which the breakpoint has to be added and click **OK** to add the breakpoint.

When a breakpoint is added on a route, at runtime the messages passing through the route are intercepted by the breakpoint. The intercepted messages can be viewed, edited or forwarded to the next service instance.

Message body, message properties and the application context can be viewed in the debugger view. When an intercepted message is selected, the properties are shown in the Properties view.

The Application context is shown in the Application Context tab.

7.2 Viewing Messages at Breakpoint

All the messages sent to a route having breakpoint set on it are visible in the breakpoint view when clicked on that particular route as shown in Figure 7.2.1.

When the messages are intercepted on the route, the route blinks and the message count will be appended to the route name.

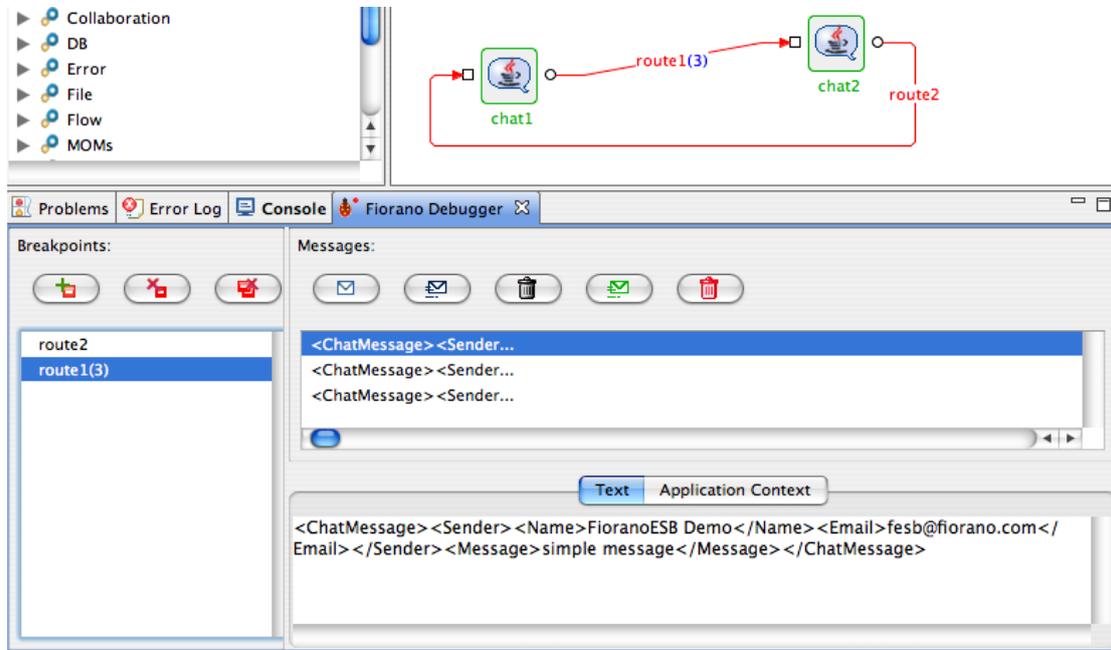


Figure 7.2.1: Message at breakpoint in Fiorano Debugger

7.3 Editing Messages at Breakpoint

To edit a message at debug time, perform the following steps:

- Select the message to be edited and edit it in the **Text** section as shown in Figure 7.3.1.
- The message is saved.

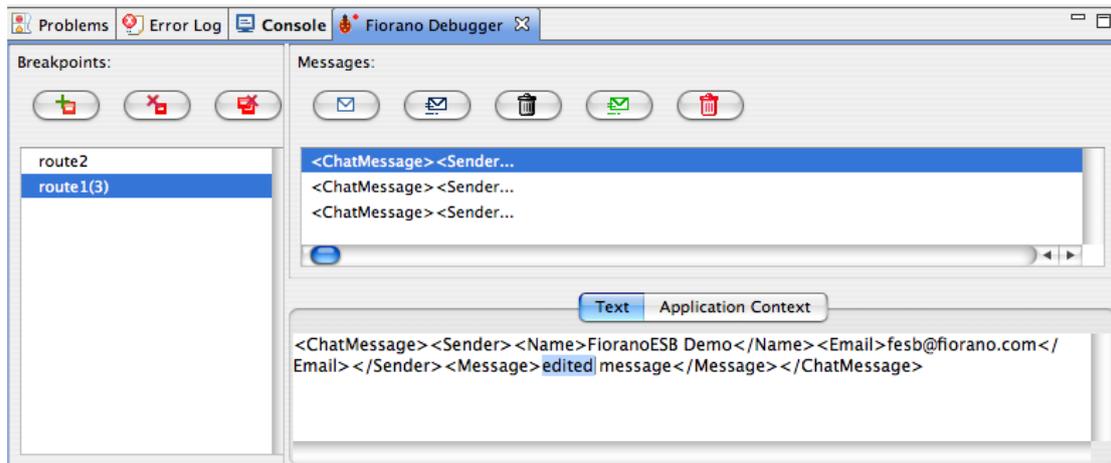


Figure 7.3.1: Edit message in Fiorano debugger

7.4 Inserting Messages into Breakpoint

New messages can be inserted into breakpoint at debug time without the message being sent by the source component.

To insert messages into breakpoint, perform the following steps:

1. Click the **Create** button in the Messages pane as shown in Figure 7.4.1.

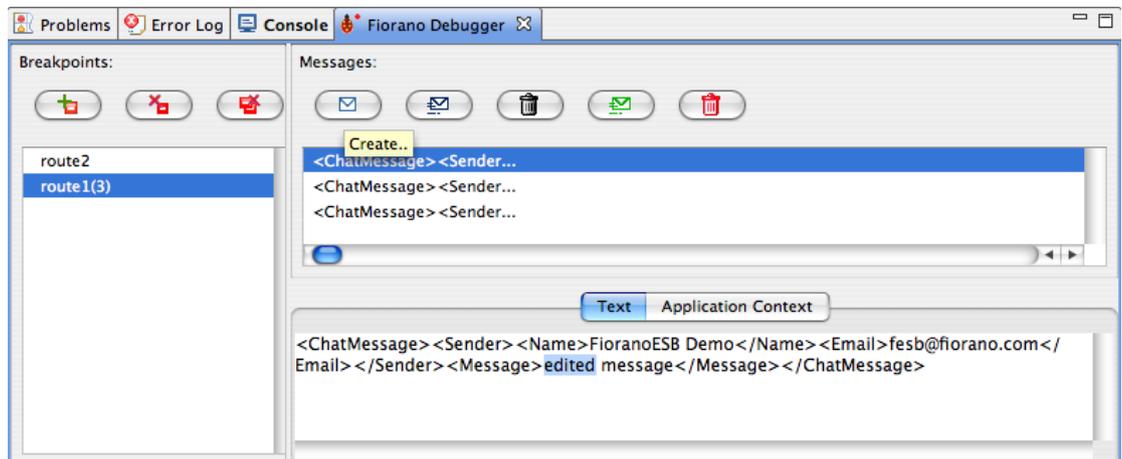


Figure 7.4.1: Create message in Fiorano debugger

2. Choose the type of message to be created (either XML or Text message) as shown in Figure 5.4.2 and click **OK**. For a Text type, a default message is inserted, which can be edited in the **Text** section. For XML type, the XML schema of the message is shown and the user can click on **Generate Sample** button to generate a sample XML data and can edit the data in the **Text** section.

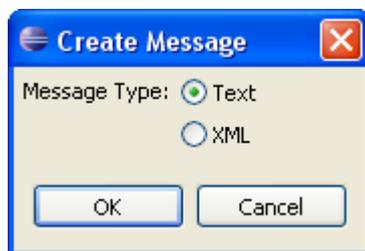


Figure 7.4.2 select type of new message

7.5 Releasing Messages from Breakpoint

The messages present on a breakpoint can be released anytime so that they reach their destination.

To release messages from the breakpoint, perform the following:

1. Select the message to be released and click the **Send** button shown in Figure 7.5.1. The message will be sent to the next service instance in the event process.

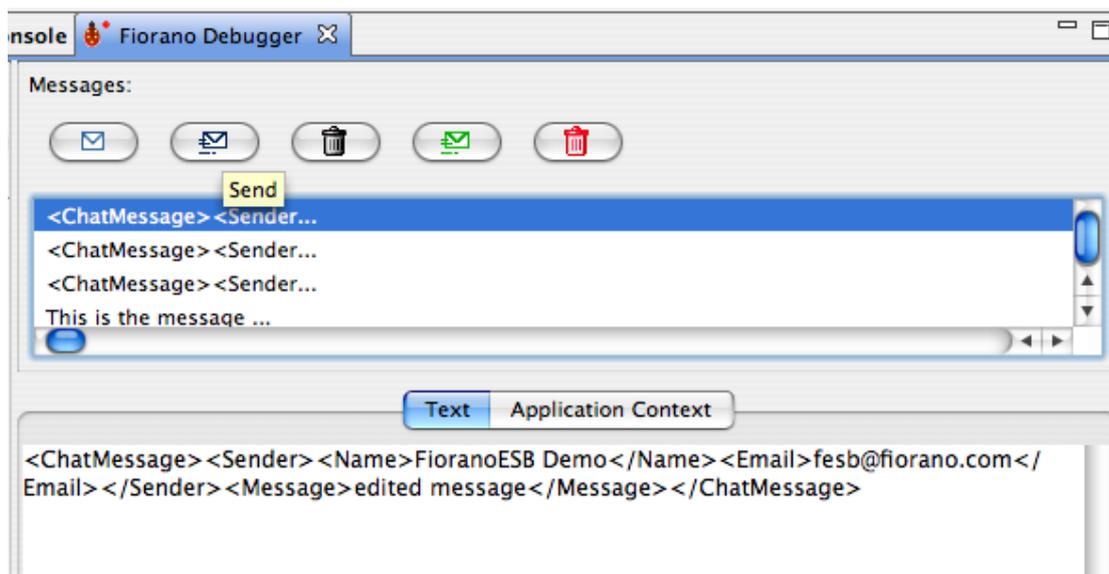


Figure 7.5.1: Send message in Fiorano debugger

2. All messages on Breakpoint can be released at a time by clicking on the **Send All** button as shown in Figure 7.5.2.

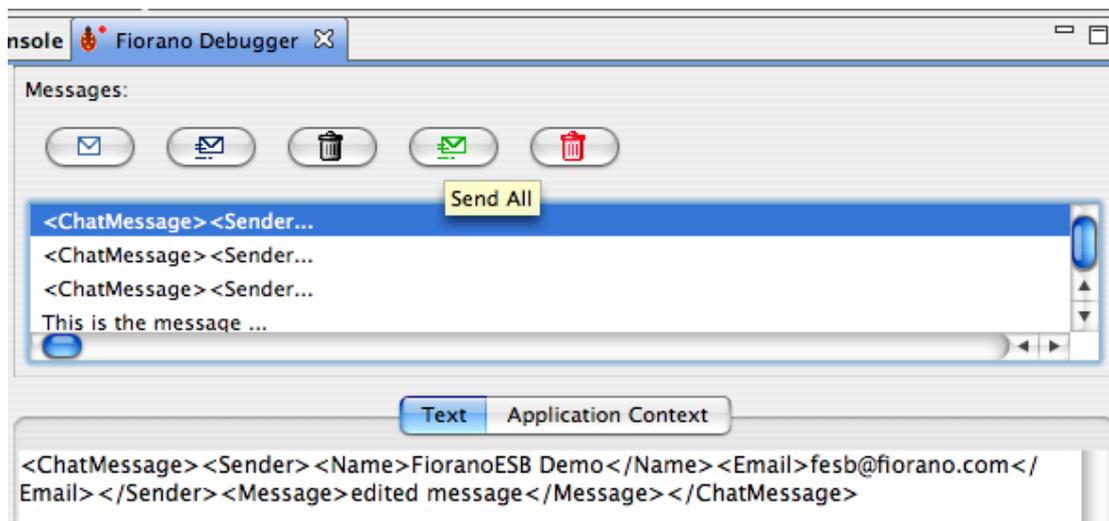


Figure 7.5.2: Send all messages in Fiorano debugger

All the messages can also be sent at a time from route context menu by right-clicking on the route and by selecting the **Send All** option.

7.6 Discard Messages from Breakpoint

To discard the messages from the breakpoint, perform the following:

1. Select the message to be discarded and click the **Discard** button shown in Figure 7.6.1. The discarded message will be removed from Breakpoint.

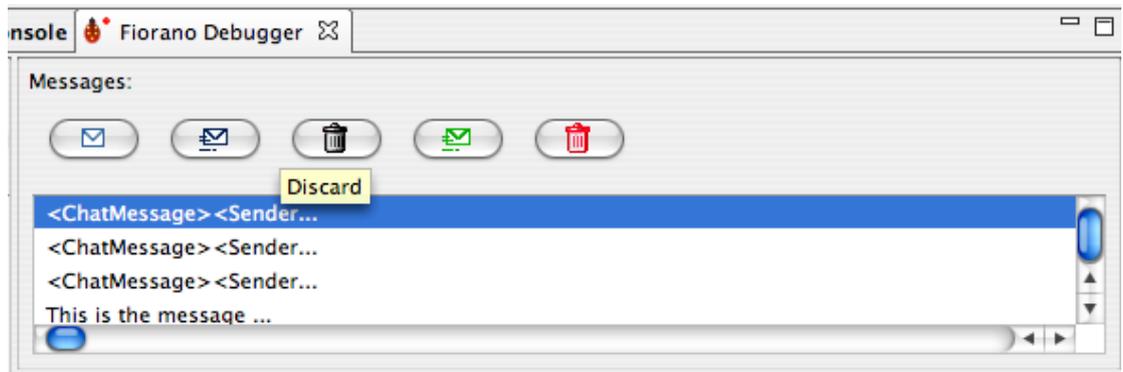


Figure 7.6.1: Discard message in Fiorano debugger

2. All messages on Breakpoint can be discarded all at a time by clicking on the **Discard All** button as shown in Figure 7.6.2.

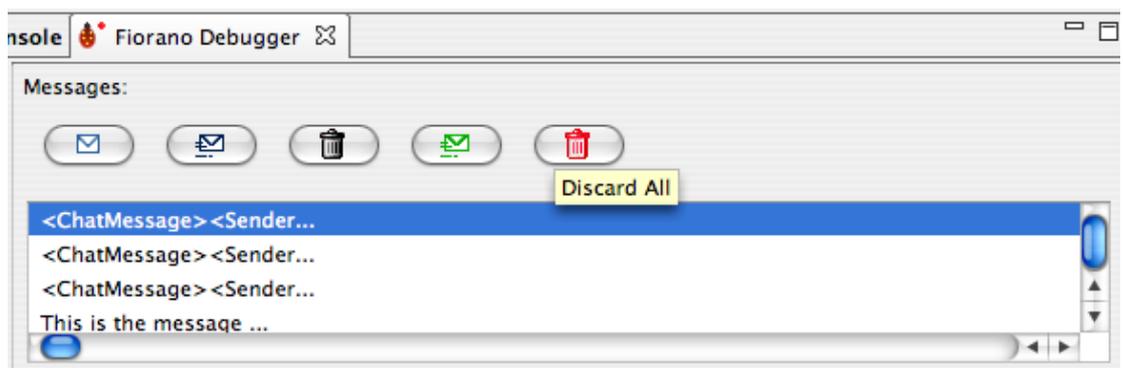


Figure 7.6.2: Discard All messages in Fiorano debugger

All the messages can also be discarded at a time from the route context menu by right-clicking on the route and by selecting the **Discard All** option.

7.7 Remove Breakpoint

To remove the breakpoint set on a route, perform the following:

1. Select the route on which the breakpoint has to be removed and click the **Remove Breakpoint** button shown in Figure 7.7.1. The breakpoint will be removed on that route.

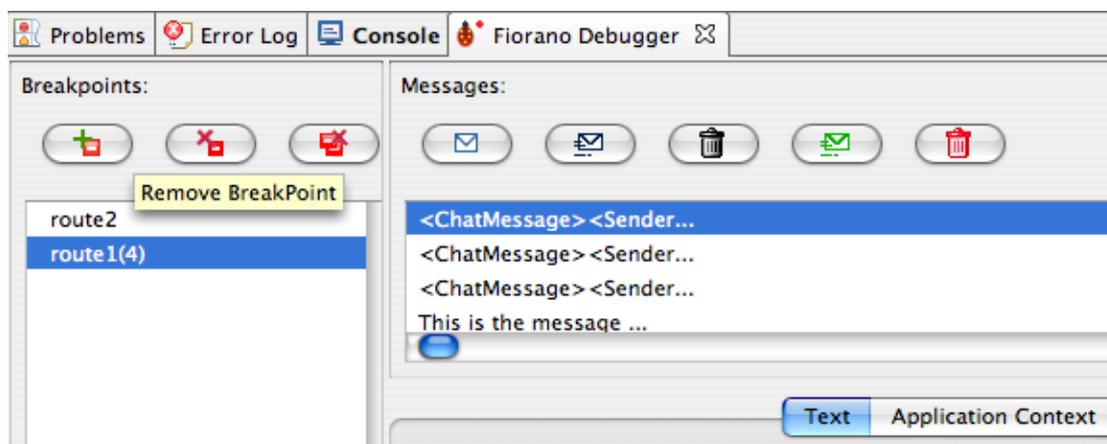


Figure 7.7.1: Remove Breakpoint in Fiorano debugger

Note: When removing a breakpoint an input dialog box comes up asking whether to send the messages or discard the messages. The user can choose the appropriate option.

Breakpoint can also be removed from context menu options on the route.

2. Breakpoints on all the routes can be removed by clicking on the **Remove All Breakpoints** button as shown in Figure 7.7.2.

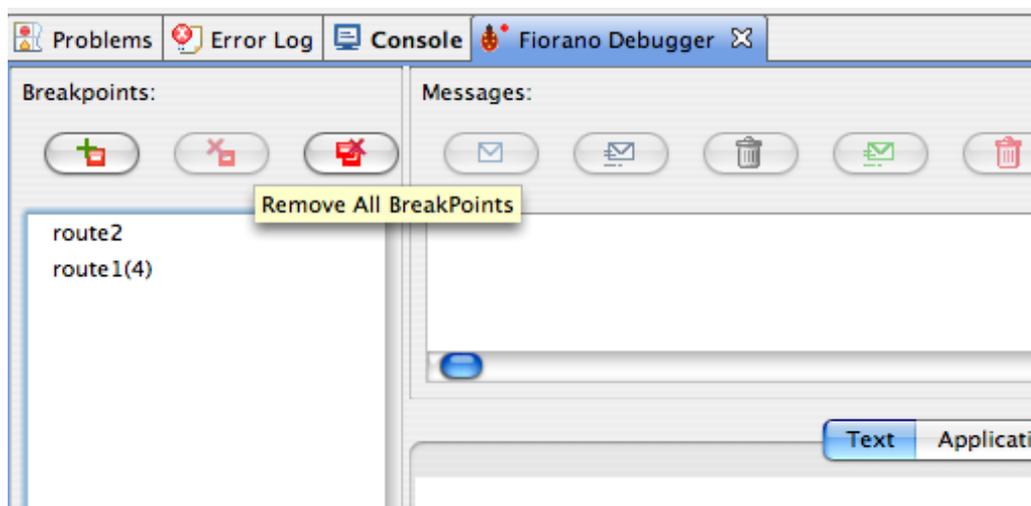


Figure 7.7.2: Remove All BreakPoints in Fiorano debugger

All the messages can also be discarded at a time from route context menu by right-clicking on the route and by selecting the **Discard All** option.

Chapter 8: Services

8.1 Service Descriptor Editor

A service can be customized using the Service Descriptor Editor. To customize a service, perform the following steps:

1. Right-click the service in the Service Palette or in the Service Repository view and select the **Edit...** option as shown in Figure 8.1.1.

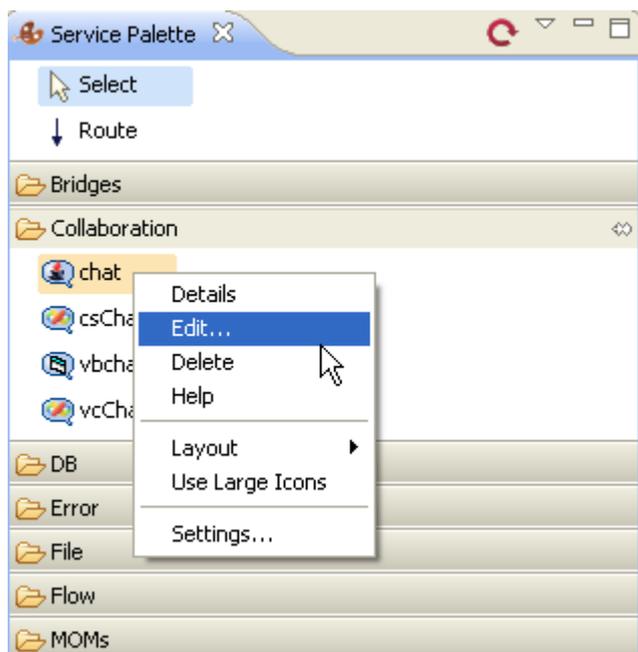


Figure 8.1.1: Edit option

2. The **ServiceDescriptor.xml** of the selected service is opened in the Service Descriptor Editor as shown in Figure 8.1.2.

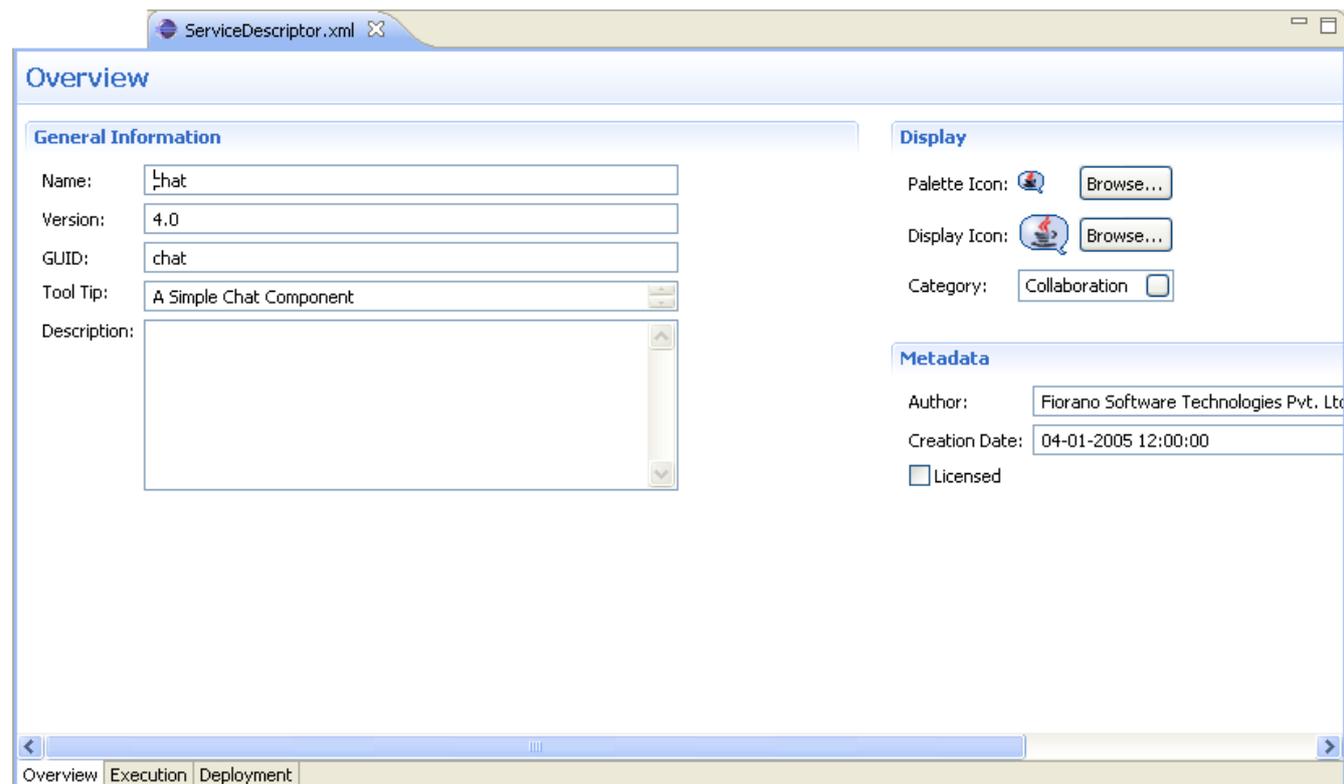


Figure 8.1.2: ServiceDescriptor.editor

The Service Descriptor Editor has three sections:

- Overview
- Execution
- Deployment

These sections are further divided into sub-sections. A brief explanation of these sections and subsections is provided below.

The sections can be accessed using the tabs provided at the bottom left corner of the editor as shown in Figure 8.1.3.

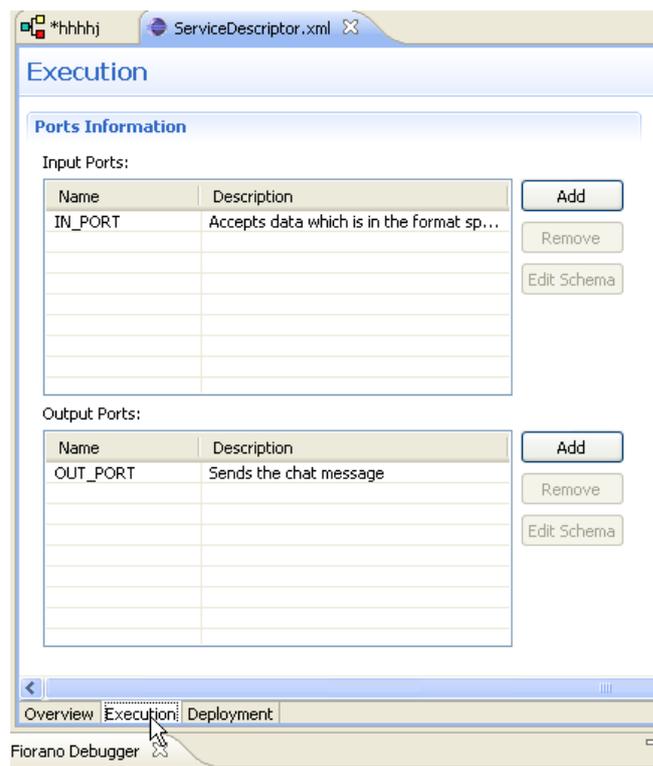


Figure 8.1.3: Sections under Service Descriptor

8.1.1 Overview Section

The Overview section has three sub-sections – General Information, Display, and Metadata.

The information used to identify the service is shown under the General Information section. The user can change the Name, Version, GUID, Tool Tip, and Description of the component in this section. Figure 8.1.4 illustrates the General Information section.

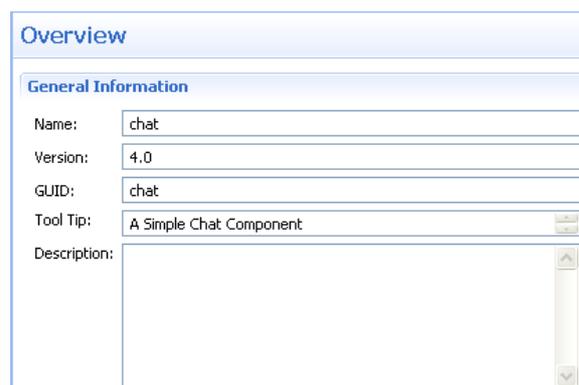


Figure 8.1.4: General Information

In the Display section, icons used to represent the service and the categories under which the service are provided. Categories can be selected using the Category Selection dialog box, which is similar to the one used during Service Creation (Figure 8.1.5).

In the Metadata section, the information about authors of the service, creation date and time of the service and licensing mode are provided (Figure 8.1.5).

Note: The Creation Date field cannot be changed manually.

The screenshot displays two sections of the Fiorano eStudio interface:

- Display Section:** Contains three rows of controls:
 - Palette Icon:** A small icon followed by a 'Browse...' button.
 - Display Icon:** A small icon followed by a 'Browse...' button.
 - Category:** A text box containing 'Collaboration' and an unchecked checkbox.
- Metadata Section:** Contains three rows of controls:
 - Author:** A text box containing 'Fiorano Software Technologies Pvt. Ltd.'
 - Creation Date:** A text box containing '04-01-2005 12:00:00'
 - Licensed:** An unchecked checkbox.

Figure 8.1.5: Display and Metadata sections

8.1.2 Execution Section

The Execution section has following subsections – Port Information, Support, Launch Configuration, Log Modules, and Runtime. A brief explanation of these subsections is provided below.

8.1.2.1 Port Information

Each Asynchronous Service Component (also referred as Event Driven Business Component) can have any number of inputs and outputs as determined by the developer of the component. The input and output ports can be added or removed in the Service Descriptor Editor as applicable to the component in the Port Information section (Figure 8.1.6).

The Add, Remove and Edit Schema buttons can be used to add, remove and/or edit the ports of services. Name and Description of any port can be modified from their respective columns in each table.

Ports Information

Input Ports:

Name	Description
IN_PORT	Accepts data which is in the format sp...

Output Ports:

Name	Description
OUT_PORT	Sends the chat message

Buttons: Add, Remove, Edit Schema

Figure 8.1.6: Port Information section

8.1.2.2 Support

In the Support section, Failover Supported and Transaction Supported options are available as shown in Figure 8.1.7.

Support

Failover Supported

Transaction Supported

Component Control Protocol

Figure 8.1.7: Support section

Failover Supported

If the Failover Supported option is selected, then during the component's runtime if the Peer Server on which component is running goes down, the component keeps running on the next available Peer Server.

If this option is not selected at the component's runtime, if the Peer Server on which component is running goes down, the component stops.

Transaction Supported

Transaction Supported is used to specify whether the service allows transacted session or not.

Component Control Protocol

- **Checked** – Component listens, understands and responds to control events from Peer Server. Using this option allows components launched as separate process to cleanup when stopping
- **Unchecked** – Component does not handle control events from the Peer Server. The Peer Server will not send any control events to the component. Component launched in separate process is issued a destroy command to stop and the component process will be killed instantly without any cleanup.

8.1.2.3 Launch Configuration

In the Launch Configuration section, information about the type of component and the different launch type supports (None, Separate Process, In Memory, and Manual) are provided.

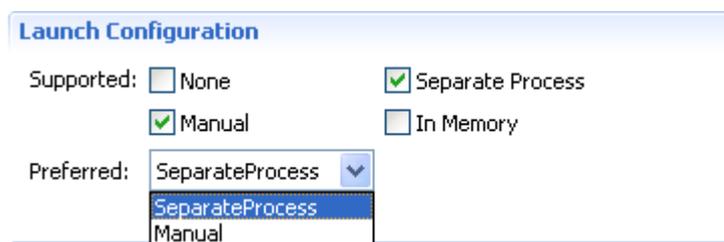


Figure 8.1.8: Launch Configuration section

8.1.2.4 Log Modules

In the Log Modules section, logging options of the service are provided. Loggers which are used to log messages during service runtime can be added or removed.

To add a new logger, click the **Add** button and specify log module name and the log level at which logging has to be performed. Messages logged at levels which are lower than the selected log level will not be written to the log files.



Figure 8.1.9: Log Modules section

8.1.2.4 Runtime

In the Runtime section, configurations required to launch services are provided. Executable specifies the Java class to be used to launch the service when it is launched in a Separate Process.

In Memory Executable specifies the Java class to be used when the service is launched in in-memory mode.

The Working Directory specifies the directory which will serve as the service's runtime directory when launched in a separate process.

A component while executing, might require parameters to execute different requests or details for 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 or by defining the command line arguments that can be passed to the component at the time the component is launched. These command line arguments are captured as runtime arguments in this panel.

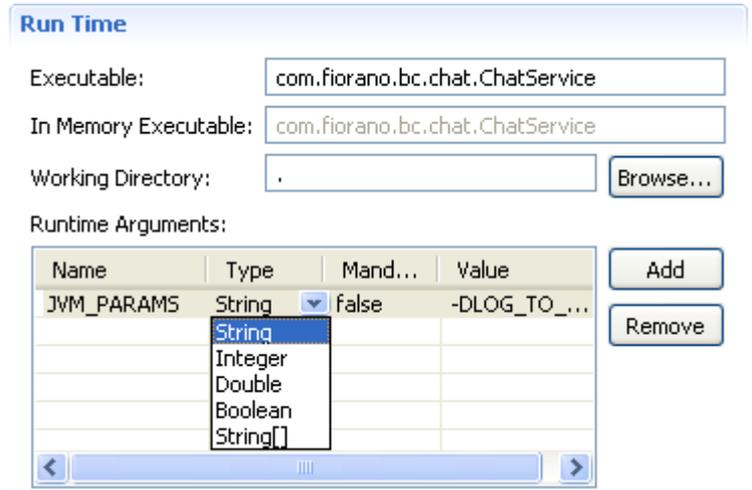


Figure 8.1.10: Runtime section

8.1.3 Deployment Section

The Deployment section contains subsections related to the deployment information of the component. The Resource/Service Dependencies required by the component can be configured in this section.

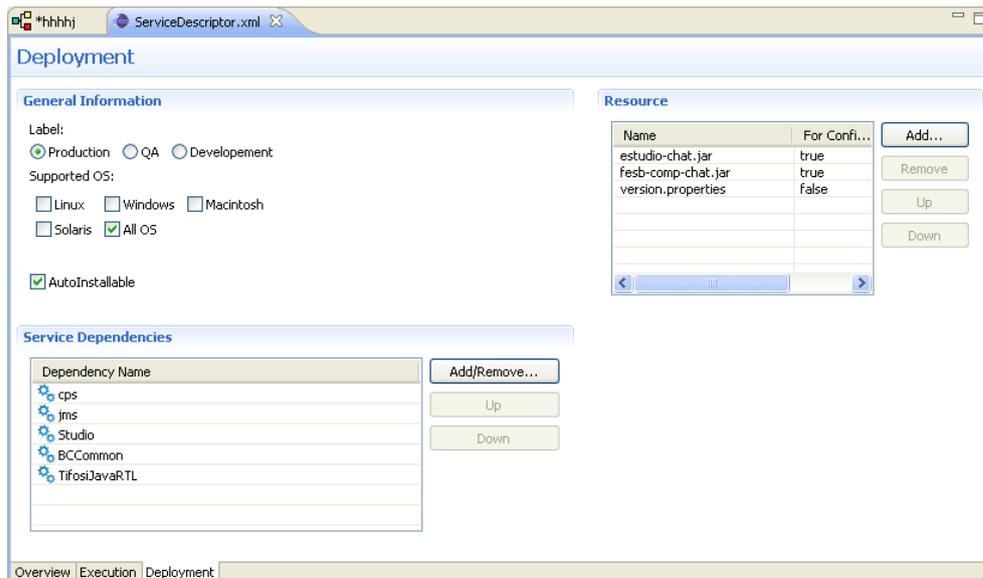


Figure 8.1.11: Deployment page

8.1.3.1 Resource

The resources required by the service (either during configuration time or runtime) can be added in this wizard. Resources can be any files which are used by the component. Typically resource files are – dll, zip, jar, so, and exe.

- To add a resource, click on **Add** and select required resource for the service.
- To remove a resource, select the resource and click **Remove**.
- To change the order of resources, select the resource and click the **Up** or **Down** button. The order is used to determine the classpath of the service.

8.1.3.2 Service Dependencies

Dependencies are predefined. Each component or system library registered can be added as a dependency.

Click the Add/Remove button to open the **Add Dependencies** dialog box. This contains a list of all available dependencies.

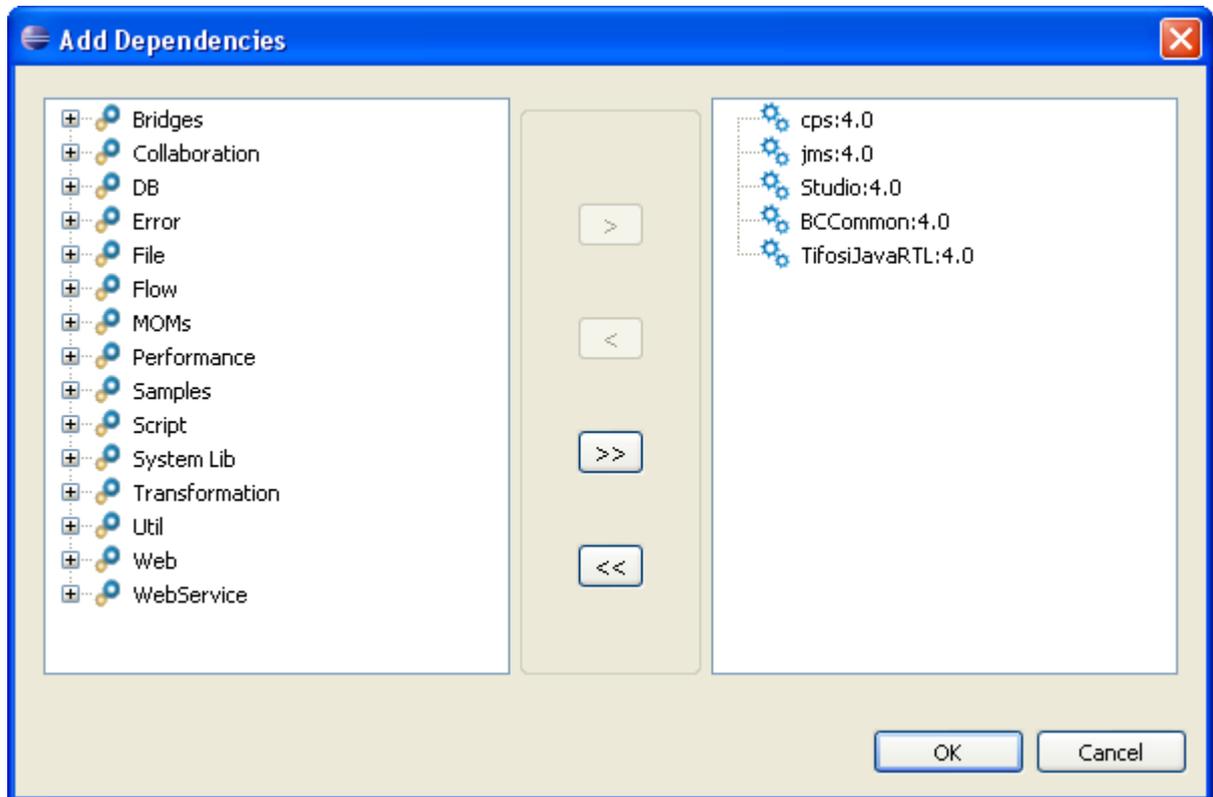


Figure 8.1.12: Service Dependencies section

- **To Add:** Select the dependency on the left side table and move it to the right side table.
- **To Remove:** Select the dependency on the right side table and move it to the left side table.

Chapter 9: Service Creation

Apart from the exhaustive list of pre-built services, custom services can be written, built, and deployed into the Fiorano SOA Platform by developers. To aid developers in service creation, the platform provides a template engine to generate the skeleton code for custom services in Java, C, C++, C# (.Net). User can create a component in any language, add the business logic and deploy it in the Fiorano environment.

9.1 Service Generation

To create a new service, goto **Tools -> Create Service Component** to open the Service Creation Wizard. All the details related to the creation of a new service must be specified in this wizard. Various steps in service creation are illustrated below.

9.1.1 Service Location

The destination folder in which the component source code and other required files to be generated has to be specified.

Note: A new folder name has to be specified here. If the folder name provided already exists, then the wizard does not allow proceeding to the next page.

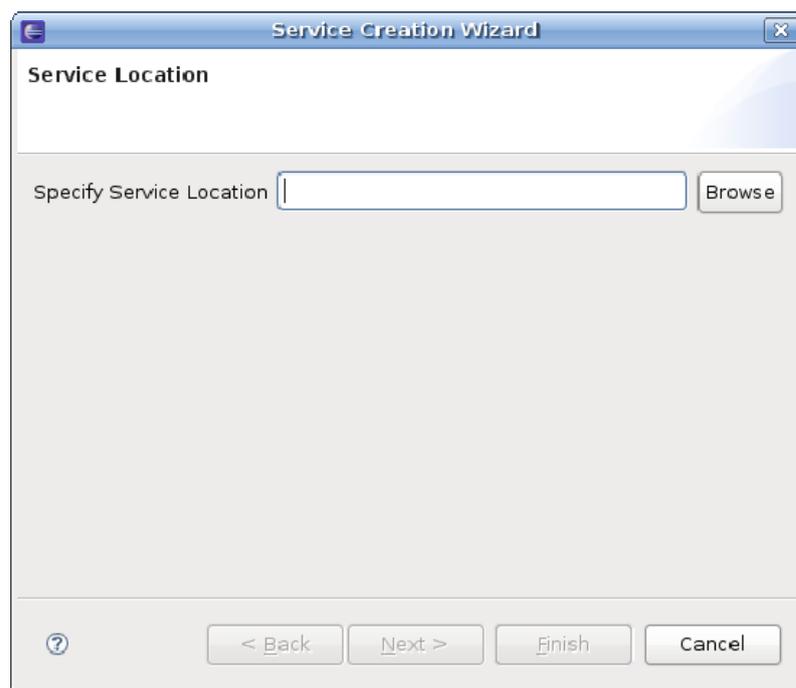


Figure 9.1.1: Specific Service Location

9.1.2 Basic Details

The Basic Details of the service like Service Guid, Name, Version, Category, and so on have to be provided here.

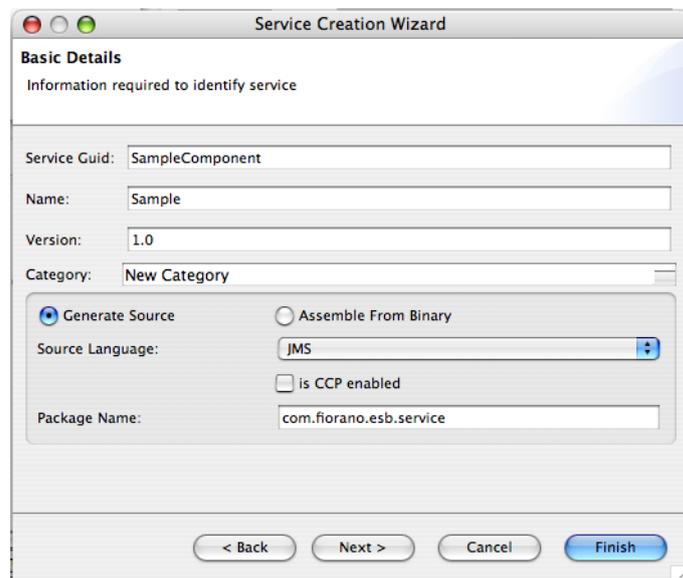


Figure 9.1.2: Service creation wizard

In the Category field, a new Category name can be provided for the component or an existing Category can be selected from the available categories. Existing Categories can be viewed by clicking the ellipsis button that appears against the Category field. On clicking ellipsis, the **Category Selection** dialog box appears as shown in Figure 9.1.3. Multiple Categories can also be selected in the Category Selection dialog box.

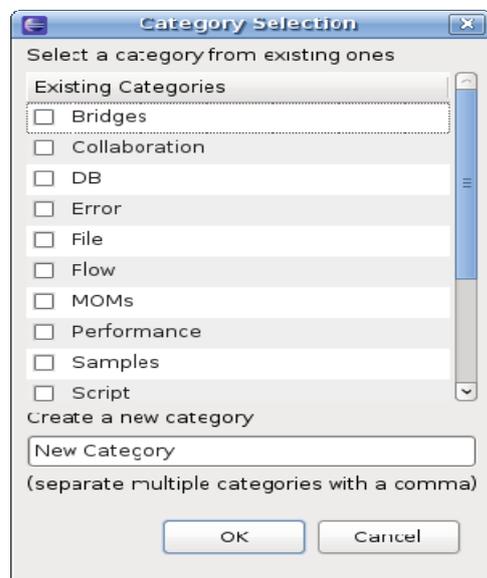


Figure 9.1.3: Category Selection dialog box

The option **Generate Source** is used to generate sources for various languages and the option **Assemble From Binary** is used to create System Libraries.

Is CCP Enabled

- **Yes** – Component listens, understands and responds to control events from Peer Server. Using this option allows components launched as a 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 be editable while editing the service from Studio.

For additional details on the Component Control Protocol refer to section 3.12 in Fiorano SOA User Guide

9.1.3 Ports Information

The input and output ports of the service can be configured here.

A new port can be added by clicking the Add button. By default Port Type is Input Port. The Port Type and other port properties can be changed in the Service Creation Wizard as required.

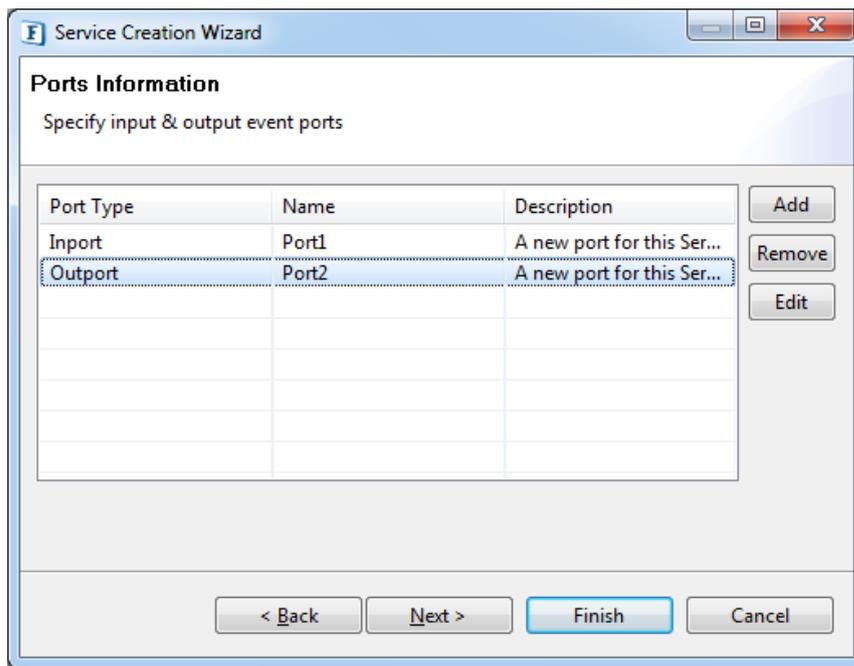


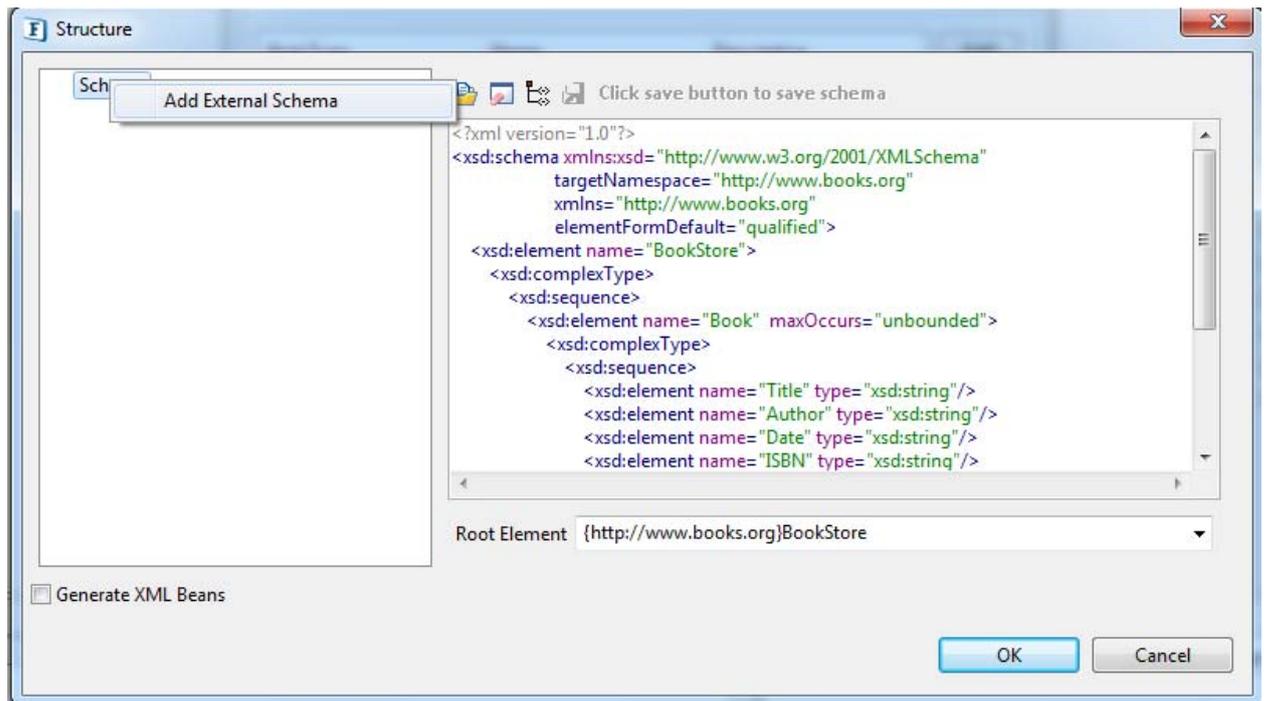
Figure 9.1.4: Ports Information

To provide schemas for the component ports, select the port and click the Edit button. A dialog will be launched where the schemas and external schemas (if any) can be provided.

Provide the schema and click the Save button. Select the root element and click Ok.

Note: Root element can be selected only when the schema provided in the text area is saved by clicking the Save button.

To provide external schemas, right-click on the Schema node and select Add External Schema option.



The option Generate XML Beans can be used to generate model objects based on the port schemas.

9.1.4 Resources

The resources required by the service (either during configuration time or runtime) can be added in Service Creation Wizard. Resources can be any file types 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. Resources can be added or removed using **Add** and **Remove** buttons.

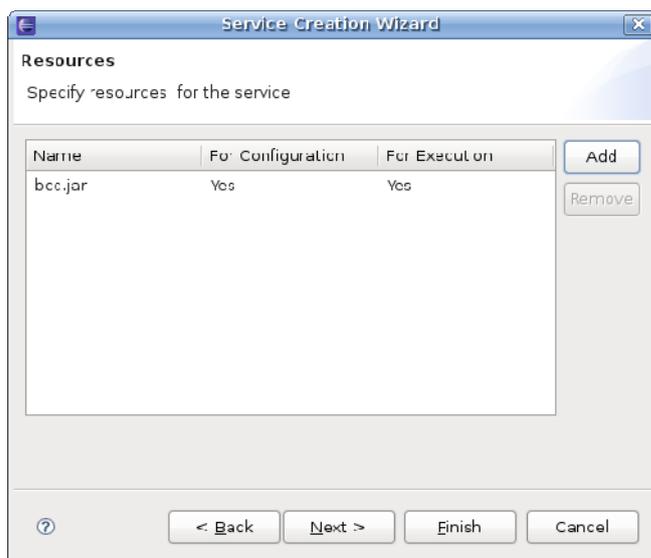


Figure 9.1.5: Resources section

9.1.5 Dependencies

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 into the component's folder.

Note: Dependencies are loaded only once when the components are launched in-memory of same peer server, there by reducing the memory footprint.

- **To Add:** Select the dependency on the left-hand side of the page and move it to the right-hand side.
- **To Remove:** Select the dependency on the right-hand side of the page and move to the left-hand side.

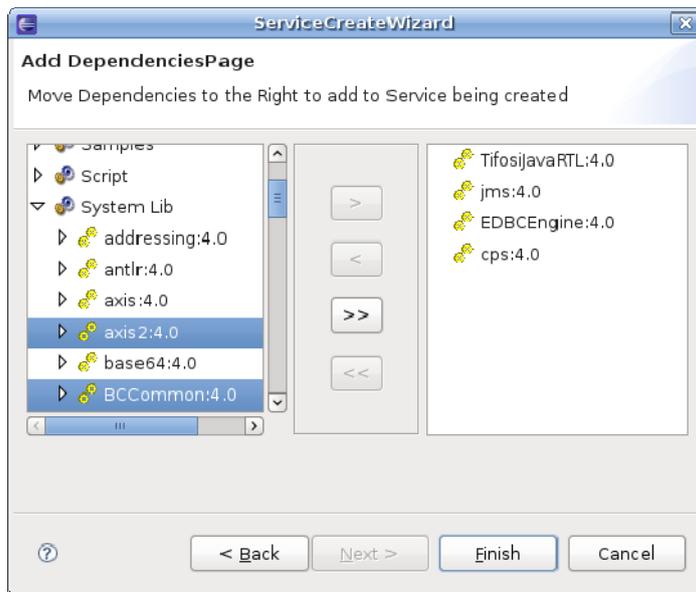


Figure 9.1.6: Dependencies

- Click the **Finish** button after adding the dependencies.

When the wizard is finished, sources are generated under the **src** directory in the directory specified in the Service Location Page. It also creates necessary files to build and deploy the components.

9.2 Building and Deploying Services

By default, the **build.properties** file contains the URL of the Enterprise Server running on the machine on which the sources are generated. If the service has to be deployed to an Enterprise Server running on a different machine, then the property server has to be changed in the **build.properties** file.

To register the service, perform the following steps:

1. Open the command prompt at the location where the sources are generated and execute the command **ant register**.

```
Terminal Tabs Help
an-desktop:~/Desktop/new$ ant register
```

Figure 9.2.1: Registering the service

2. This builds the service's sources and registers the service with the Enterprise Server.
3. The service is now available in the eStudio Service Palette and can be used in composing Event Processes.

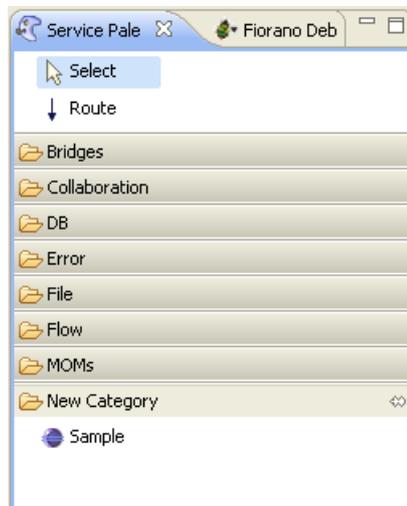


Figure 9.2.2: Service Palette

Chapter 10: eMapper

The Fiorano eMapper is a high-end graphical tool that presents the user with both source document structure and target document structure side-by-side and lets the user define semantic transformation of data by simply drawing lines between nodes, elements, and functions.

The Fiorano eMapper uses standards based XSLT (Extensible Stylesheet Language for Transformations), which is a language for transforming documents from one XML structure to another.

Additionally, Fiorano eMapper ensures that the source and target document structures conform to the DTD (Document Type Definition) standards.

10.1 Key Features of Fiorano eMapper

The Fiorano eMapper performs a variety of operations including:

- Transforming one or more XML, XSD or DTD files.
- Generating XML, XSD or DTD as output of the transformation.
- Option to convert a FIX message to any XML and viceversa.
- Using Funclets to define complex mapping expressions.
- Validating the transformation.
- Defining the transformation (mapping) with simple drag-and-drop actions.

10.2 Fiorano eMapper Environment

The Fiorano eMapper tool consists of the following interface elements:

- eMapper Projects Explorer
- eMapper Editor
 - Map View
 - MetaData
- Funclet View
- MetaData Messages View
- eMapper Console
- Node Info View

The interface of the Fiorano eMapper tool is displayed in Figure 10.2.1.

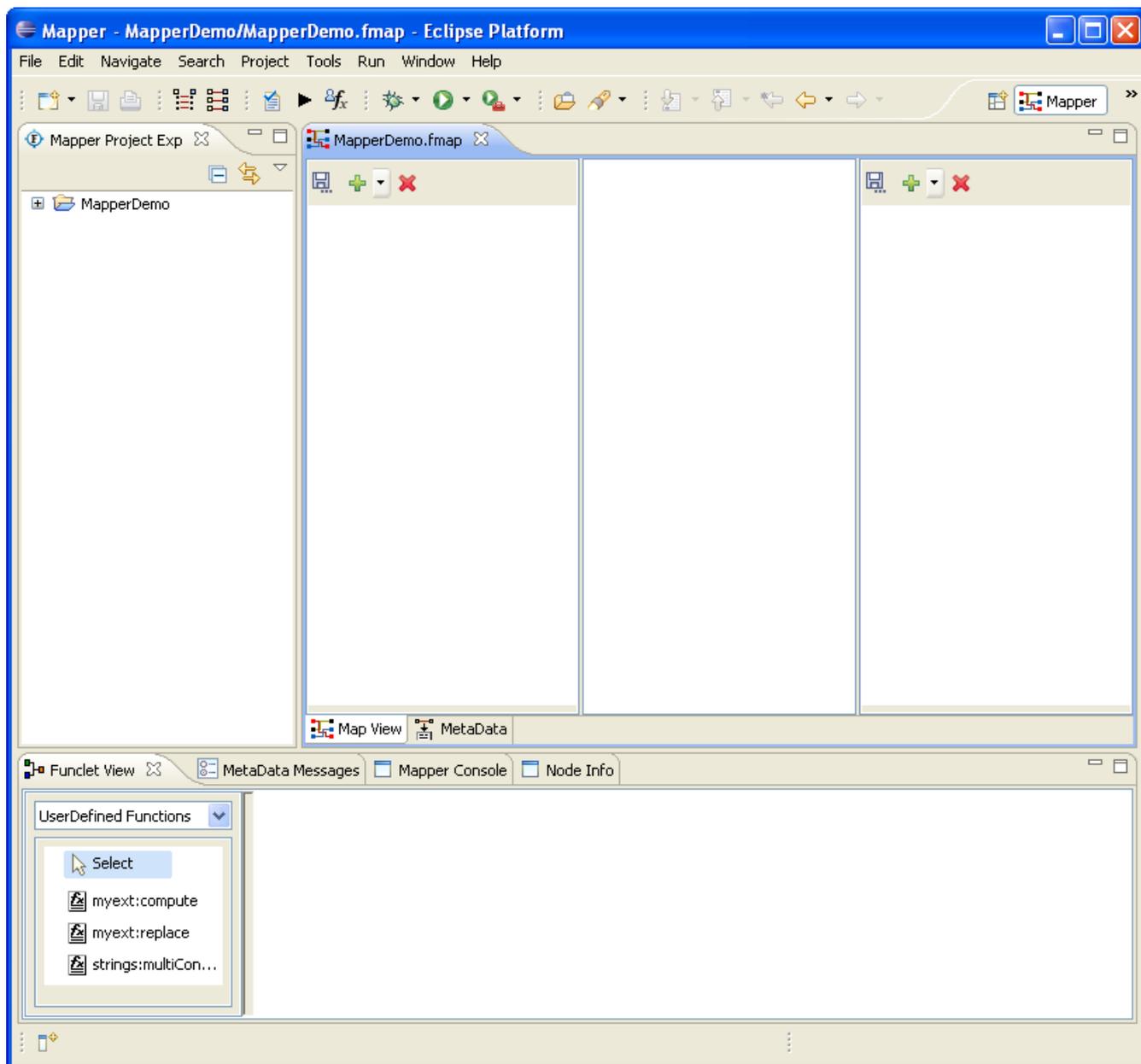


Figure 10.2.1: eMapper Perspective

10.2.1 eMapper Projects.

This view serves as an explorer for the eMapper Projects created by the User.

To create a new eMapper project, perform the following steps:

1. Right-click on the Mapper Projects node in eMapper Projects view and select **New**. The **New eMapper Project Wizard** is opened.
2. Provide a valid name for the project and click **Finish**. A new eMapper project is created.

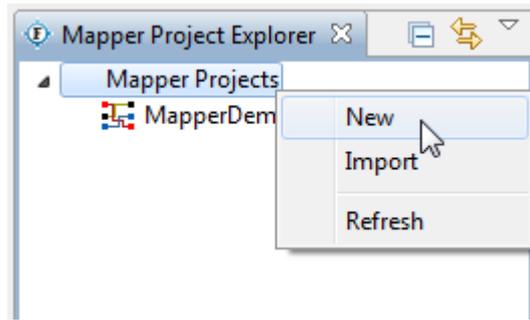


Figure 10.2.2: eMapper Projects

The eMapper Projects view also provides options to Import, Export, Rename and Delete mapper projects.

10.2.2 eMapper Editor

The eMapper Editor is a tabbed editor containing two tabs, Map View and MetaData.

10.2.2.1 Map View

The Map View shows the Input and Output Structures and the mappings defined in the pane. This view allows users to load the input and output structures and create mappings between them.

This view consists of the following panels:

- Input Structure Panel
- Graph Panel
- Output Structure Panel

Input Structure Panel

This panel shows the input specification structure in a tree format.

Graph Panel

The middle panel in Map View is the Graph 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 that particular function(s) as shown in Figure 10.2.3.

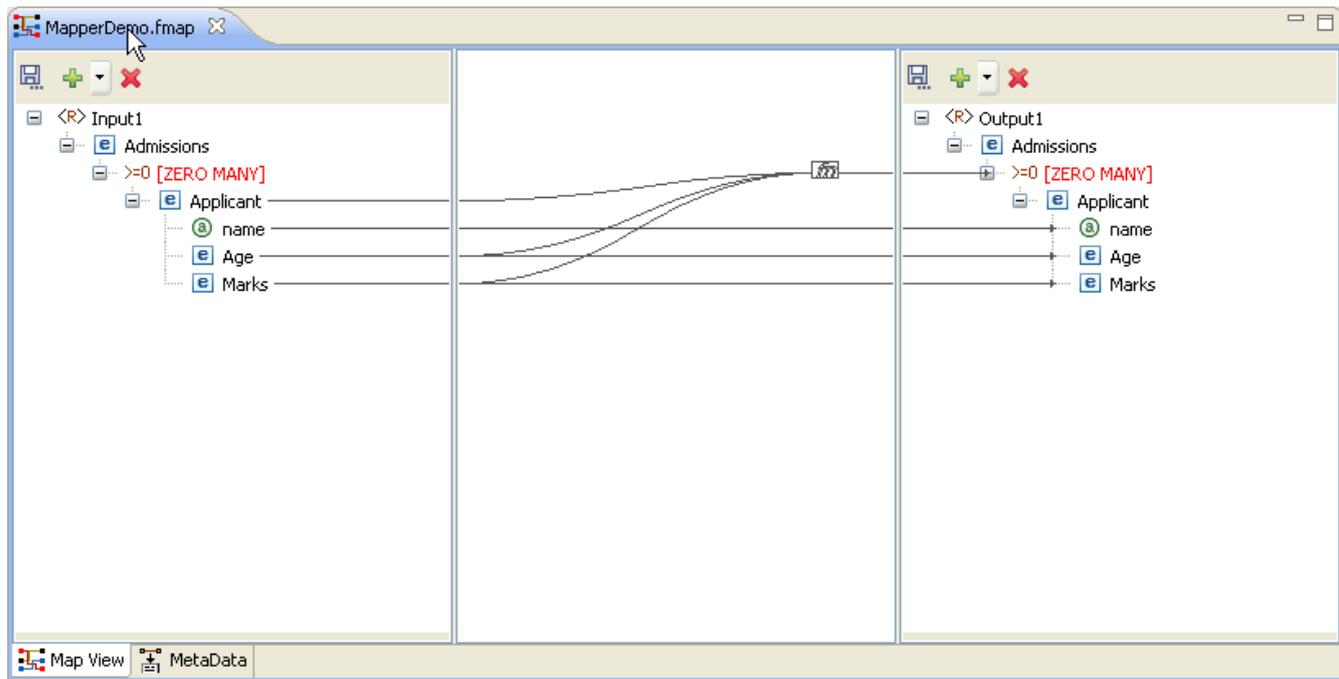


Figure 10.2.3: Map View

Output Structure Panel

This panel shows the output document structure in a tree format.

10.2.2.2 MetaData tab

The MetaData tab shows the transformation XSL generated from the mappings defined in the Map View for the selected output structure.

10.2.3 Funclet View

The Funclet view contains the Visual Expression Builder that provides a graphical view for the mappings defined in the Map View, as shown in Figure 10.2.4. It also shows the functions and their links with the input and target nodes/elements.

Note: The Funclet view is explained in detail in the Visual Expression Builder section later in this chapter.

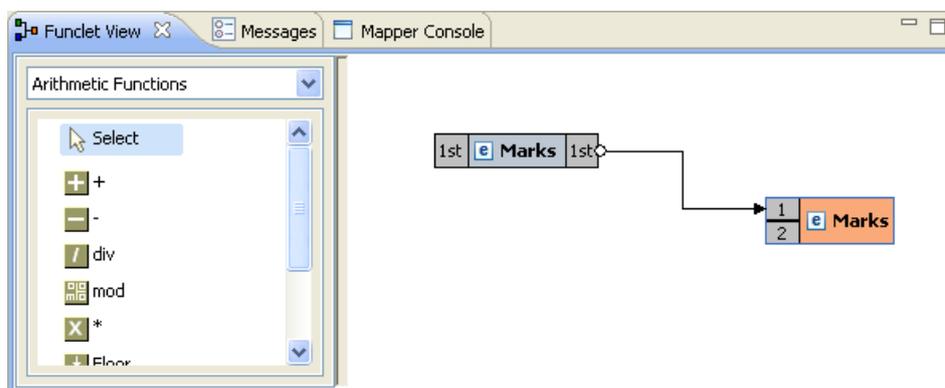


Figure 10.2.4: Funclet View

10.2.4 eMapper Console

The eMapper Console is used to display the various error and warning messages generated by the tool while parsing the input and output structures and while testing the generated XSL.

10.2.5 MetaData Messages View

Error or Warning Messages (if any) thrown while generating the transformation XSL are displayed in the MetaData Messages View. The view is shown in Figure 10.2.5.

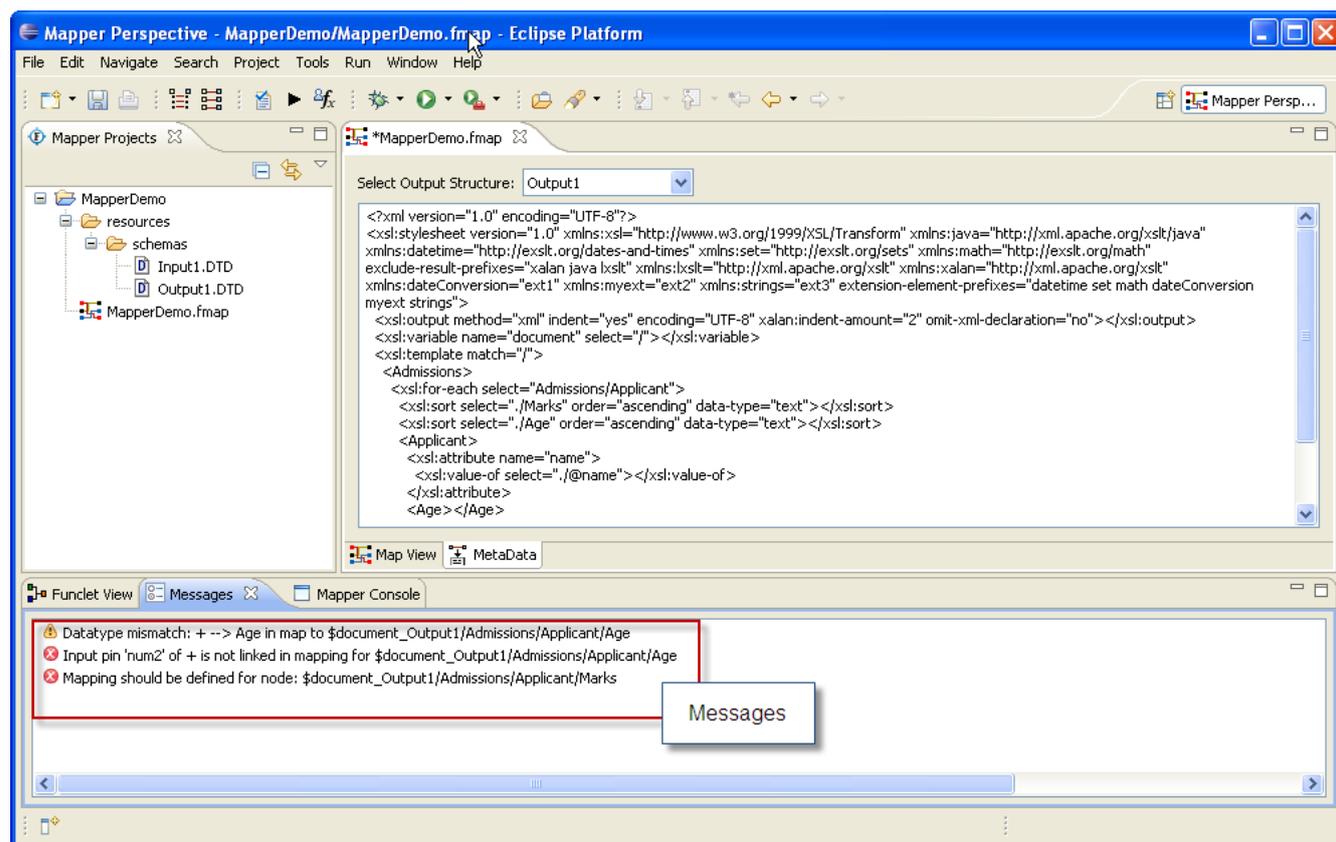


Figure 10.2.5: Meta Data and MetaData Messages view

10.2.6 Node Info View

The Node Info View shows the information about nodes in the Input and Output Structures. The view is shown in Figure 10.2.6. It has two panels that provide the data type and cardinality information about the selected input and output structure node/element.

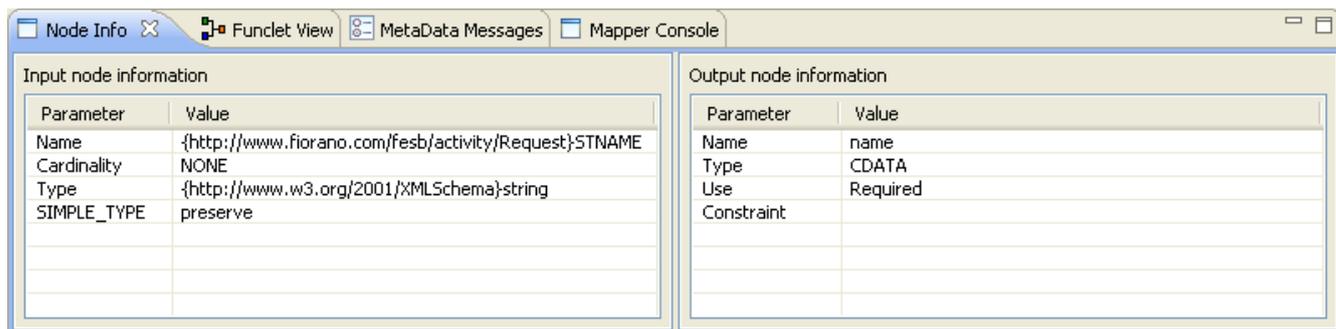


Figure 10.2.6: Node Info View

10.3 Working with Input and Output Structures

10.3.1 Loading Input/Output Structure

- An Input/Output Structure can be loaded in one of the following ways:
 - Click the **Add Structure** button from the tool bar in the **Input/Output Structure Panel** and choose the structure type from the drop down list. Or,
 - Right-click on the **Input/Output Structure Panel** and select **Add Structure** and choose the structure type from the sub-menu.
- The drop-down list or the sub-menu has the following options
 - XSD** For loading an XSD document
 - DTD** For loading a DTD document
 - XML** For loading an XML document
 - FIX** For loading a FIX message

10.3.1.1 Load Input/Output Structure From an XSD document

Select **XSD** from the **Add Structure** menu. The Load Input/Output XSD Structure Wizard appear as shown in the Figure 10.3.1. The wizard contains two pages, Structure Selection page and External XSDs page.

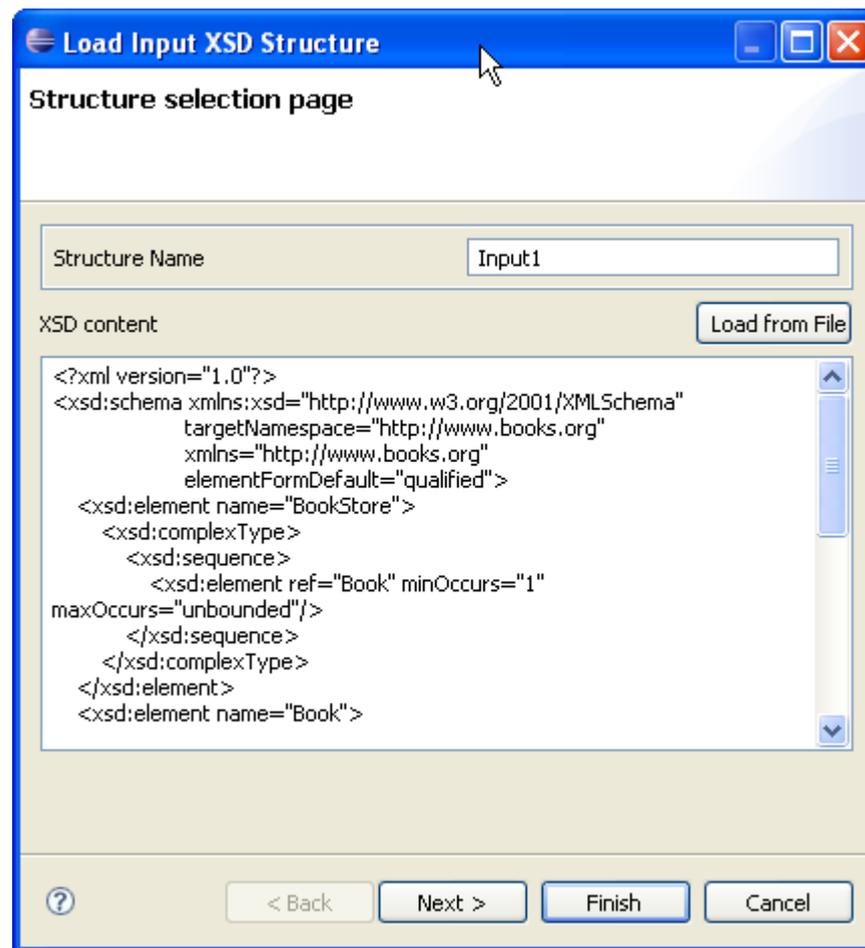


Figure 10.3.1: Structure Selection Page

Structure Selection Page

The name of the structure can be specified in the Structure Name text field at the top of the page.

Note: The structure name cannot contain special characters. Only alphabets, numbers and '_' are allowed in a structure name. Two structures with the same name are not allowed.

The XSD content can be defined in the text area provided in this page.

The schema can also be loaded from an existing file using the **Load from File** button. Clicking this button will open a file dialog through which you can browse through the file system to choose an existing file. Modifications, if any, to the schema are loaded from the file from this page.

External XSDs Page

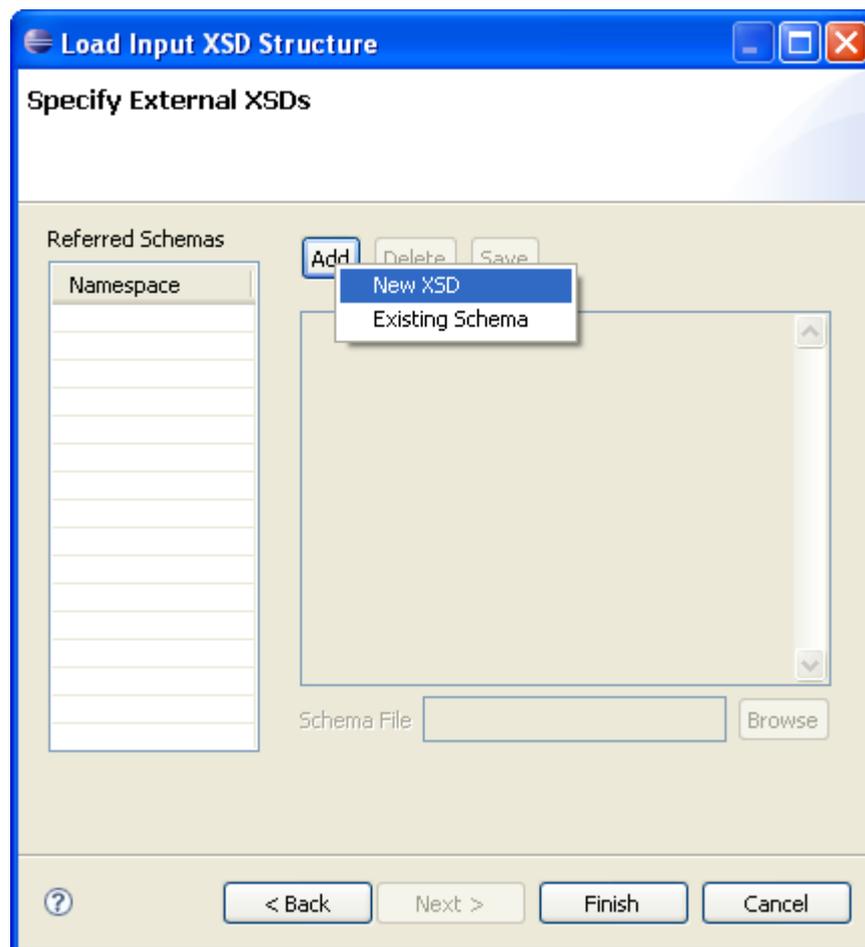


Figure 10.3.2: Adding External XSDs

Any external XSDs used by a structure can be added from this page. Figure 10.3.2 shows the External XSDs Page. External XSDs can be added by performing the following steps:

Click the **New** button to add a new external schema. A context menu will appear as shown in the Figure 10.3.2. The user can either add a New XSD or use an Existing Schema that is already present in the eMapper Project.

Adding a new XSD

- Click **New XSD**. This will enable the **Schema Content Area** where the content of the schema can be entered.
- A valid file name should be provided in the **Schema File** field. The provided XSD will be saved with the name specified in the PROJECT_HOME/resources/schemas directory.
- The content can also be loaded from a file using the **Browse** button. Click this button and browse through the file system and select the required file.
- After providing the XSD, it can be saved as an external XSD for the structure by clicking the **Save** button. As specified earlier, the XSD will be saved in the PROJECT_HOME/resources/schemas folder with the name specified in the Schema File field.
- The target name space of the schema is added to the list of **Referenced URIs** present on the left end of the page.

Adding an existing schema

- To use an XSD which is already present in the eMapper Project, click **Existing Schema** in the New context menu. A list of all the XSD present in the PROJECT_HOME/resources/schemas directory is shown. Choose an XSD and it will be saved as an external schema to the current structure.

Note: As target name space is used in referring to these schemas, therefore saving an XSD without a target name space is not allowed. Two schemas with same target name space cannot be added.

- External schemas can be removed by selecting the namespace of the structure to be deleted and clicking the **Delete** button.

10.3.1.2 Load Input/Output Structure from a DTD document

Select DTD from the **Add Structure** menu. The Load Input/Output DTD Structure wizard appears as shown in the Figure 10.3.3. The DTD content can be specified from the Structure Selection Page present in this wizard. Similar to the Structure Selection Page in Load Input/Output XSD Structure Wizard, this page allows the user to enter the structure content directly or by loading it from an existing file. To load content from an existing DTD document, click the **Load From File** button.

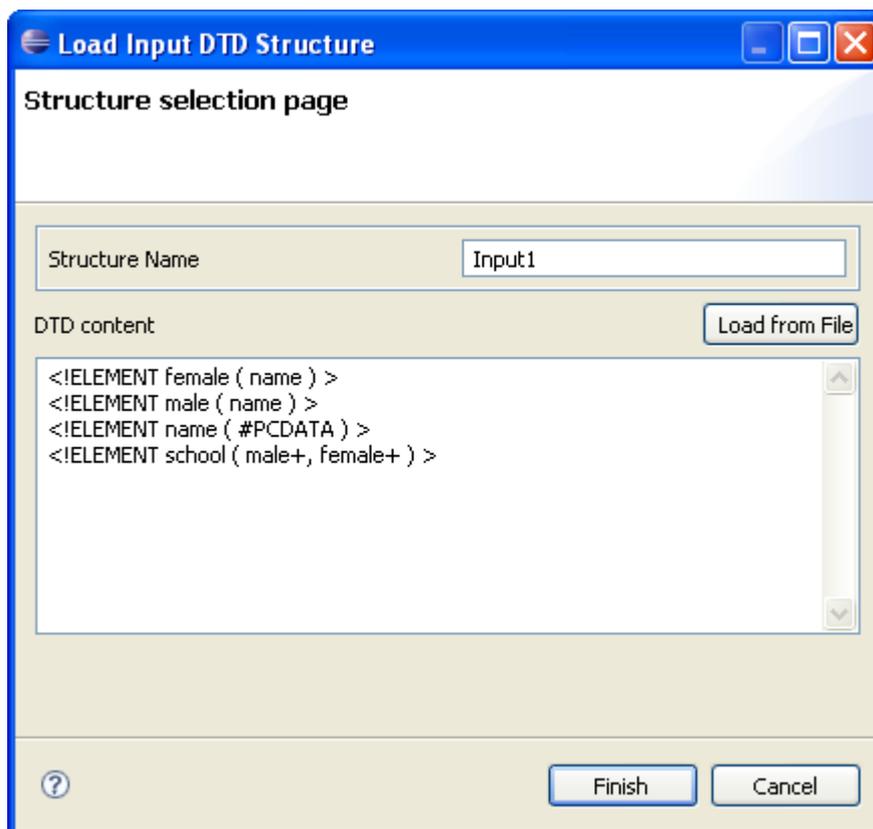


Figure 10.3.3: Load Input DTD Structure

10.3.1.3 Load Input/Output Structure from an XML document

Select XML from the **Add Structure** menu. The Load Input/Output XML Structure wizard appears as shown in the Figure 10.3.4. The dialog contains two panes: XML Content and Generated DTD.

The name of the structure can be specified in the Structure Name field. The structure name cannot contain special characters. Only alphabets, numbers and '_' are allowed in a structure name. Two structures with the same name are not allowed.

The XML can be provided in the text area present in the XML Content pane. This text area has a tool bar with two buttons, **Load From File** and **Generate DTD**. The content can be loaded from an existing file by clicking the **Load From File** button. Click the **Generate DTD** button to generate a DTD from this XML document. The DTD is shown in the text area present in the Generated DTD pane. This DTD document is used to load the structure. Modifications, if needed, can be made to this DTD.

The structure can be saved and loaded in the Input/Output Structure panel by clicking the **Finish** button. The content is saved in a file with name <Structure_Name>.<Mime_Type> in the PROJECT_HOME/resources/schemas directory. If the schema is not valid an exception is logged in the Error Log view.

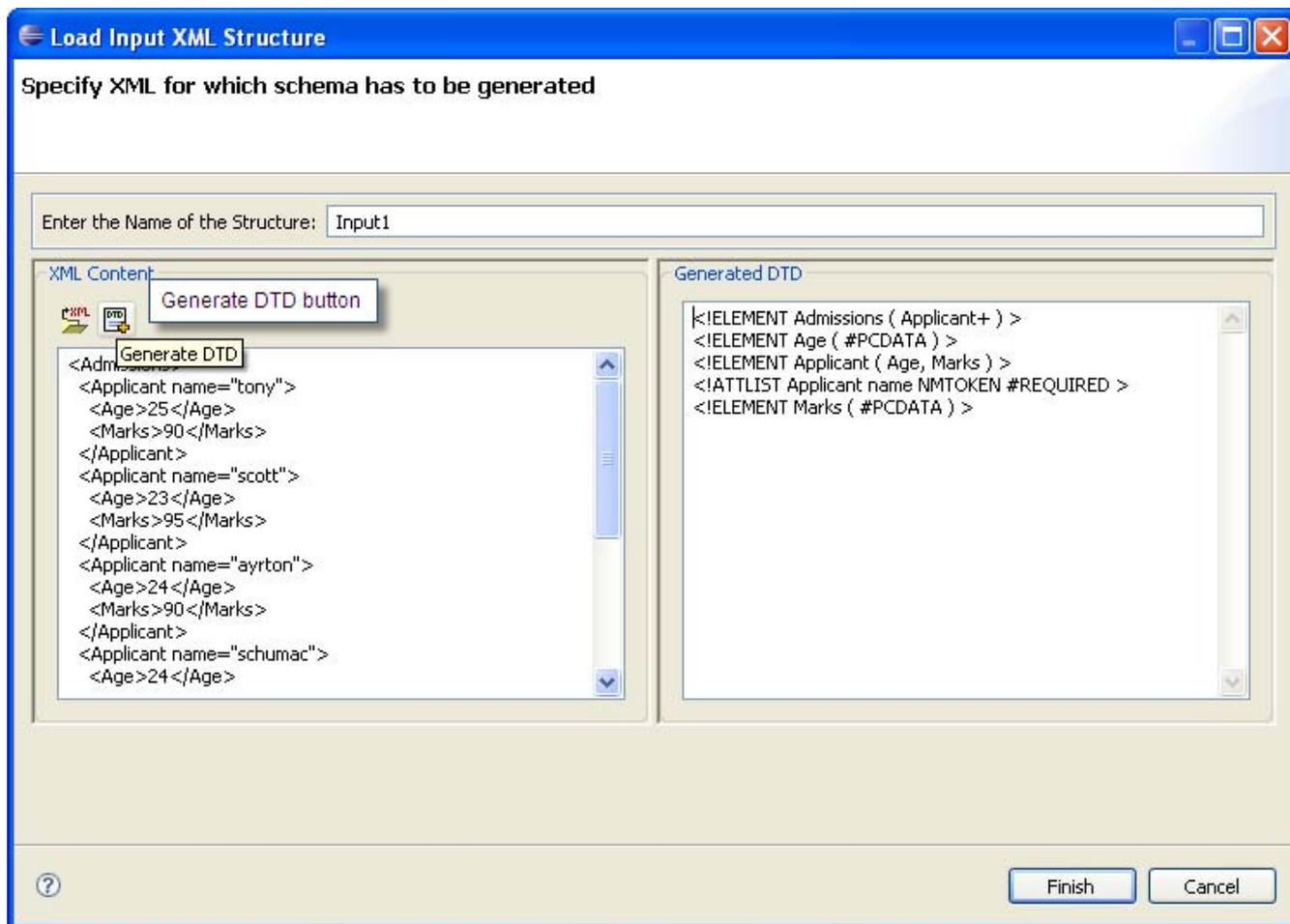
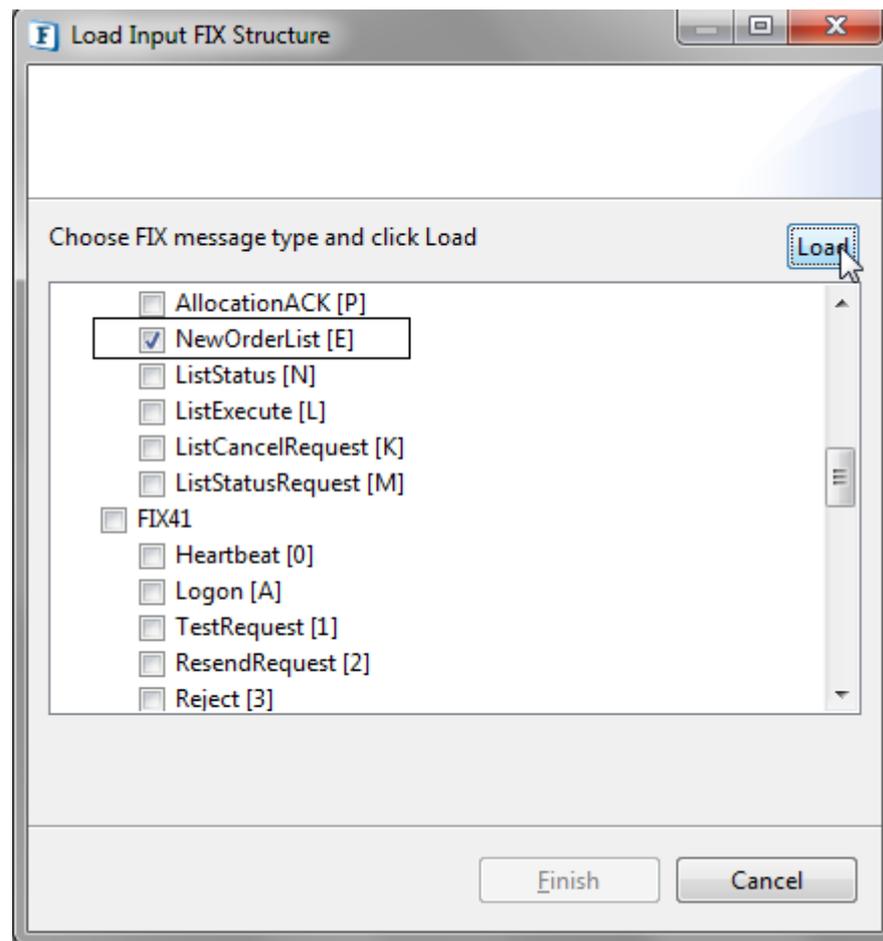


Figure 10.3.4: Load Input XML Structure

10.3.1.3 Load a FIX message as an Input/Output Structure

Select FIX from the **Add Structure** menu. A list of different FIX versions and message types is displayed where the messages to be loaded can be chosen. Select the message types and click the **Load** button to load the message types in to the workspace. Click **Finish** in the Add Structure wizard and the FIX message structure is loaded in Mapper.



More information on FIX-XML transformation is provided in [section 10.11](#)

10.3.2 Delete Structure

A structure can be deleted from the Input/Output Structure Panel by clicking the Delete Structure panel present in the structure panel's tool bar. This will delete the selected structure and will clear all the mappings associated with this structure.

10.3.3 Edit Structure

To edit Input/Output Structure:

1. Right-click the structure and click the **Edit Structure** option. The Edit Structure dialog is opens as shown in Figure 10.3.5.
2. The selected structure is shown in the text area. Modifications to the structure can be done here. The **Load From File** button can be used to load structure from a file.
3. Click the **OK** button to save the modifications done.

If the new structure is valid, it gets saved and loaded in its corresponding panel. Otherwise, an error dialog box is shown and the modifications are ignored.

Upon editing a structure, mappings defined to the affected elements/attributes are discarded.

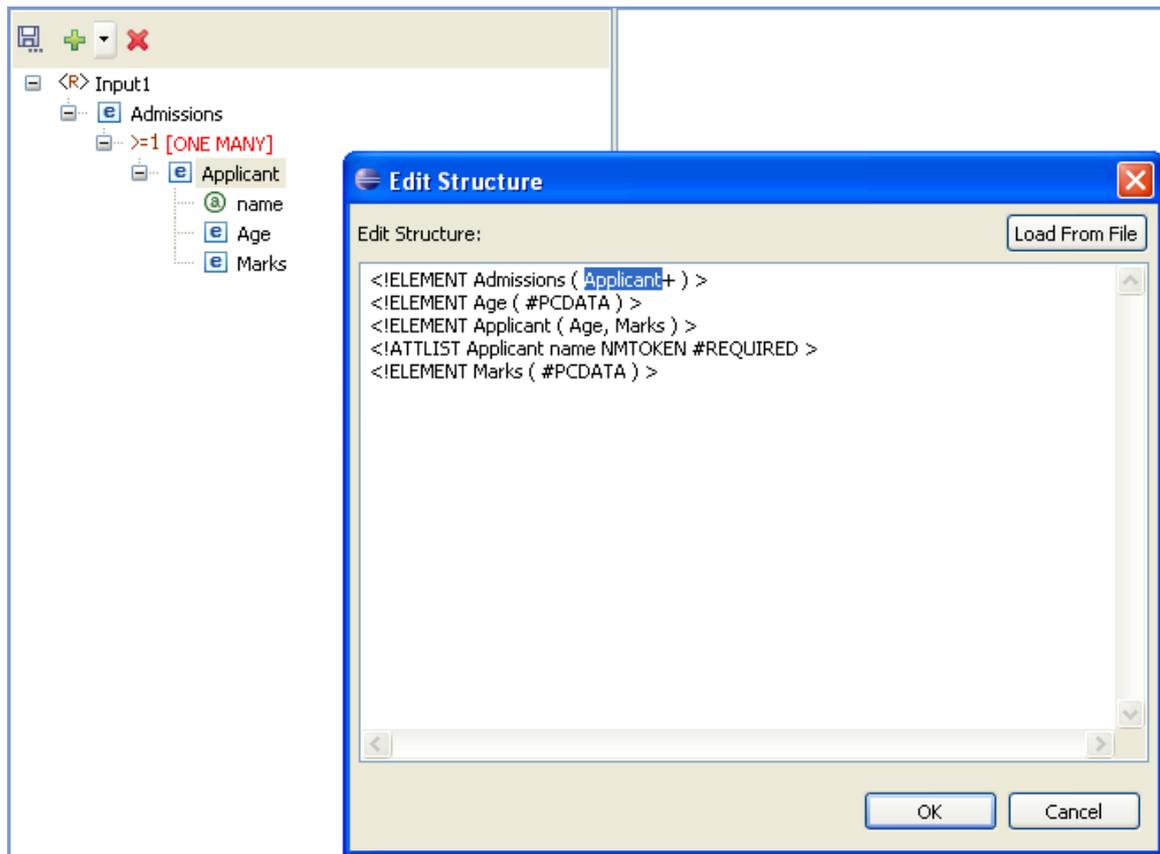


Figure 10.3.5: Edit Structure Dialog

10.4 Working with the Visual Expression Builder

Fiorano eMapper provides an easy to use graphical user interface – the Visual Expression Builder, used 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.

The Funcllet View provided in the Fiorano eMapper Perspective consists of the Visual Expression Builder. The Visual Expression Builder is shown automatically upon clicking on any node in the Output Structure.

The Visual Expression Builder consists of two areas:

Function palette

Funcllet easel

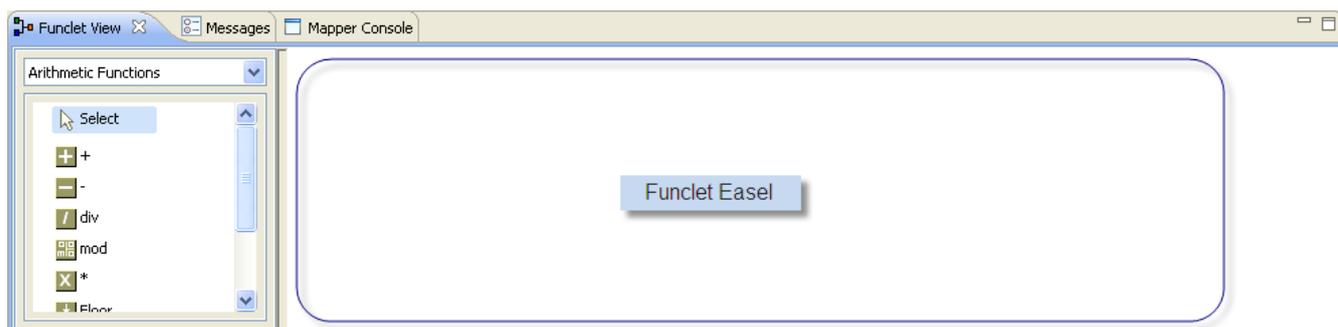


Figure 10.4 Funclet View

10.4.1 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 Functions
- User defined functions

Fiorano eMapper 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 ceiled 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.

10.4.1.2 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.

10.4.1.3 String Functions

Fiorano eMapper 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 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(" eMapperTool ")` returns "eMapper Tool".

White spaces before and after the string is removed and the white spaces between "eMapper" 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"`

10.4.1.4 Control Function

The following Control functions are available in Fiorano eMapper:

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 nodes 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.

10.4.1.5 Conversion Functions

Fiorano eMapper 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

10.4.1.6 Advanced Functions

Fiorano eMapper 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 <HEADERTNAME>
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

10.4.1.7 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 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 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--

Output Three-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--

Output: The 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.

10.4.1.8 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 a 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 a non-numeric value, then NaN is returned.

Input: A node set

Output: A numerical value

10.4.1.9 Boolean functions

The following boolean (logical) functions are available in eMapper 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: <ul style="list-style-type: none"> – 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 the example of Chat.dtd as Input Structure and Valid.dtd as Output Structure. Suppose we want to filter out mails that do not have a 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 10.4.1.



Figure 10.4.1: 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 10.4.2.

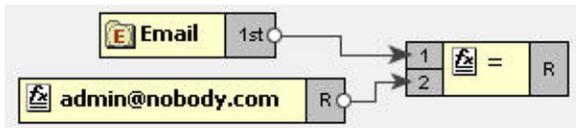


Figure 10.4.2: Linking the Email and Constant node outputs

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 10.4.3.

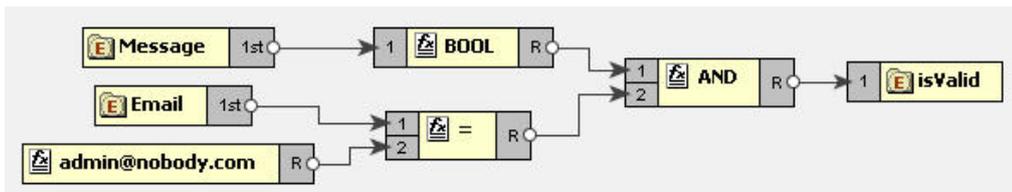


Figure 10.4.3: Linking the AND and = node outputs

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 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.

Input: BOOL (Object)

Output: Boolean value (TRUE/FALSE)

Example:

Consider the 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 10.4.4.

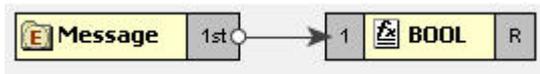


Figure 10.4.4: Linking Message and BOOL nodes

Link the output of the **BOOL** node to the input of the **isMessageExist** node, as shown in Figure 10.4.5.



Figure 10.4.5: 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 the 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 10.4.6.



Figure 10.4.6: 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 10.4.7.

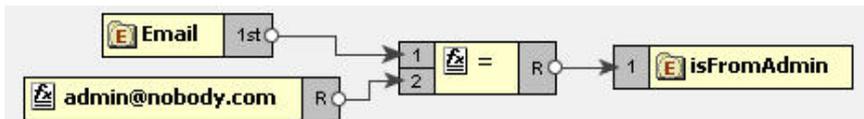


Figure 10.4.7: Linking = and **isFromAdmin** node

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 the 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 10.4.8.

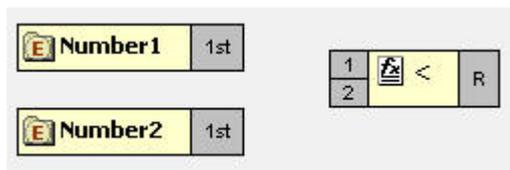


Figure 10.4.8: Placing a node on the Function Easel

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 10.4.8.

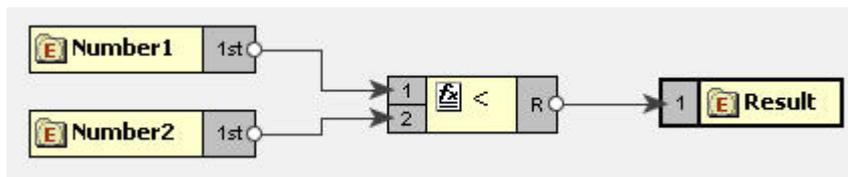


Figure 10.4.9: Linking < and Result node

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 the 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 10.4.10.

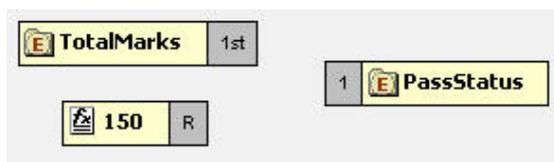


Figure 10.4.10: Setting the Constant building block to 150

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 10.4.11.

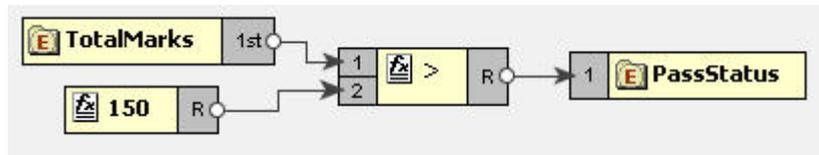


Figure 10.4.11: Linking the > and PassStatus node

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 the example of TotalMarks.dtd as Input Structure and Result.dtd as Output Structure. Suppose we want that the value of the **PassStatus** node as 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 10.4.12.

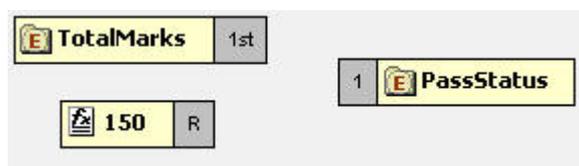


Figure 10.4.12: Setting the Constant building block to 150

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 10.4.13.

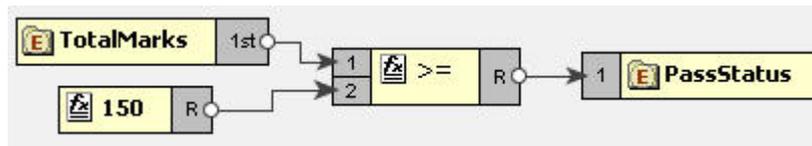


Figure 10.4.13: Linking the >= and PassStatus nodes

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 the example of Chat.dtd as Input Structure and Valid.dtd as 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 10.4.14.



Figure 10.4.14: Setting the Constant node value to aryton@nobody.com

Now place two = nodes on the Function Easel, and make links as shown in Figure 10.4.15.

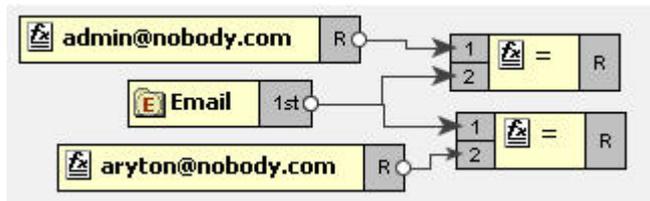


Figure 10.4.15: Placing two = nodes on the Function Easel

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 10.4.16.

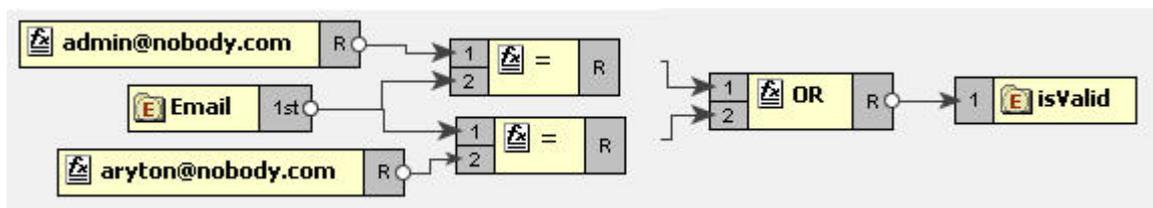


Figure 10.4.16: Linking the OR and isValid nodes

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 the example of Numbers.dtd as Input Structure and Results.dtd as Output Structure. Suppose we want that the **Result** node of Output Structure to 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 10.4.17:

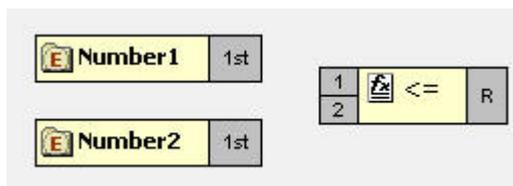


Figure 10.4.17: Placing <= node on the Function Easel

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 10.4.18:

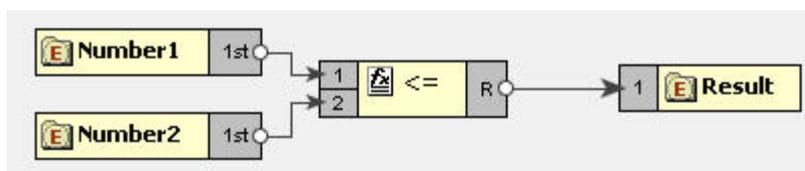


Figure 10.4.18: Linking outputs of the **Number1** and **Number2** nodes

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 the 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 10.4.19.

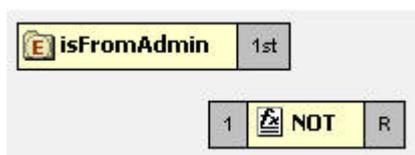


Figure 10.4.19: placing a **NOT** node on the Function Easel

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 10.4.20.

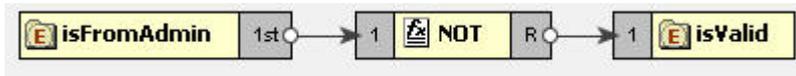


Figure 10.4.20: Linking the NOT and isValid nodes

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 the example of Chat.dtd as Input Structure and Valid.dtd as Output Structure. Suppose we want to filter out mails that do not have a 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 10.4.21.



Figure 10.4.21: 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 10.4.22.

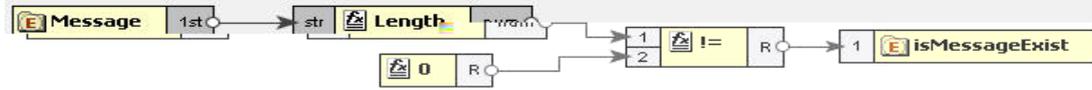


Figure 10.4.22: Linking the != and isMessageExist nodes

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)

10.4.1.10 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.

10.4.1.10.1 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 the values 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 eMapper classpath.
For example, if the lookup tables are in HSQL, include the path of **hsqldb.jar** in `<java.classpath>` of **eMapper.conf** present at `{FIORANOHOME}/esb/tools/eMapper/bin`.
2. To use this function in the eMapper tool, a system property **eMapper.lookup.dbconfig** has to be defined in **eMapper.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 10.4.23: Sample db properties file

3. For use in Route transformations, **eMapper.lookup.dbconfig** property has to be set in `{FIORANOHOME}/fps/bin/fps.conf`.
4. For use in XSLT component, **eMapper.lookup.dbconfig** property has to be included in JVM_PARAMS
For example: `-DeMapper.lookup.dbconfig=<path of db.properties>`

10.4.1.10.2 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`

Lookup functions can return value of multiple columns. To get multiple columns, use the format **<column3>,<column4>**.

For example, dvValueDA, dvDescription

3. Dates are expected in MM/dd/yyyy HH:mm:ss format
4. 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 eMapper classpath.

For example, if the lookup tables are in HSQL, include the path of **hsqldb.jar** in **<java.classpath>** of **eMapper.conf** present at **{FIORANOHOME}/esb/tools/eMapper/bin**.

10.4.1.11 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

10.4.1.11.1 Byte Content

Function: Byte Content

Description: The Byte Content function returns the byte content of a Fiorano document.

Output: Base64 encoded string value

10.4.1.11.2 Text Content

Function: Text Content

Description: The Text Content function returns content which is in text format from a Fiorano document.

Output: String value

10.4.1.11.3 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

10.4.1.11.4 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

10.4.1.12 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

10.4.1.12.1 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

10.4.1.12.2 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

10.4.1.12.3 myExt:compute

Description: This user-defined function can be used to compute all mathematical operations such as Addition, Subtraction, Multiplication and division of numbers. The function does not compute mathematical operations such as cos, sin, etc.

Input: A valid javascript expression

Output: A number

10.4.1.12.4 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

10.4.2 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 10.4.24.

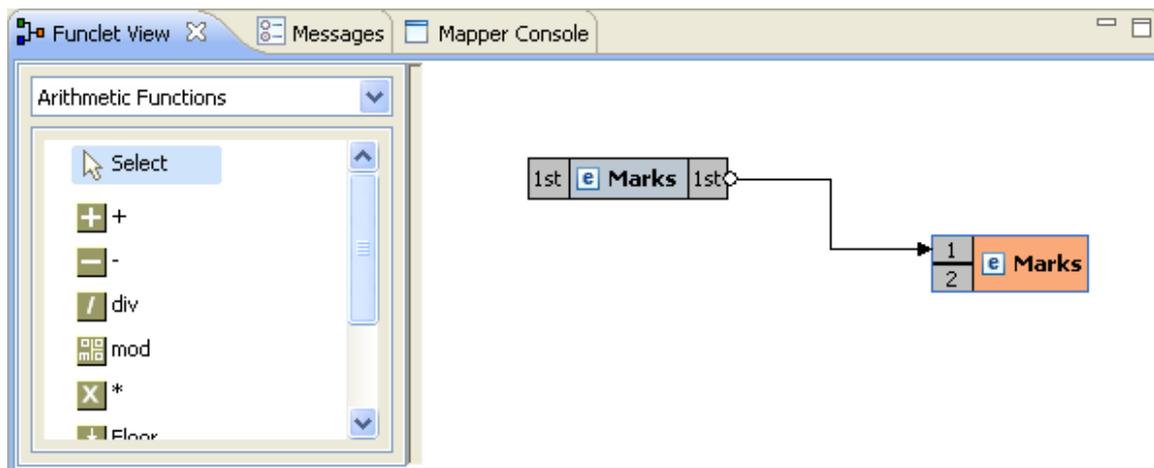


Figure 10.4.24: Funclet easel

10.4.2.1 Source Node

The Source node corresponds to a node in the Input Structure Panel. A Source node is shown in Figure 10.4.25.



Figure 10.4.25: Source Node

10.4.2.2 Destination Node

The Destination node corresponds to a node in the Output Structure Panel. A Destination node is shown in Figure 10.4.26.



Figure 10.4.26: 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 10.4.27. This represents the starting point of the link and the output box of the building block.



Figure 10.4.27: 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 10.4.28.

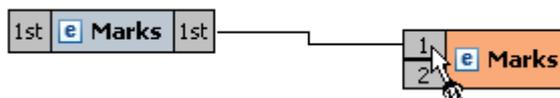


Figure 10.4.28: Linking the Source and the Destination node

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 10.4.28.

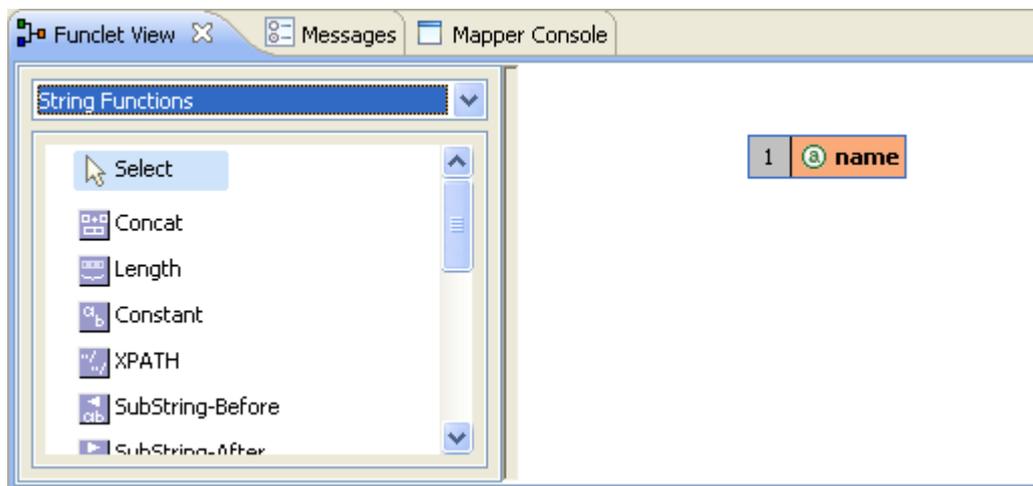


Figure 10.4.29: Adding Source node to Funclet easel

Add Function node to Funclet easel

Click the Function node on the Function palette that is to be placed on the Funclet easel, as shown in Figure 10.4.30.

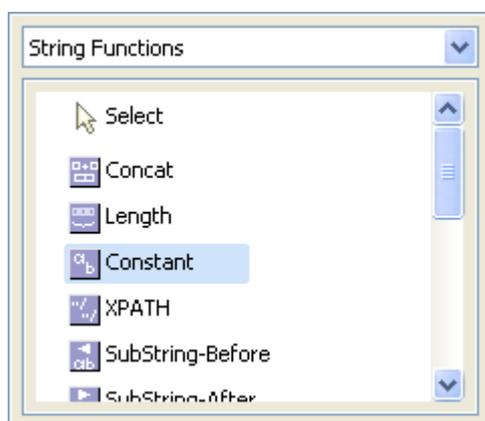


Figure 10.4.30: Selecting the Function node

1. Now, move the mouse onto the Funclet easel. This changes the mouse to a '+' sign, representing that the corresponding function node is selected.
2. Now click on the Funclet easel.
3. 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 10.4.31.

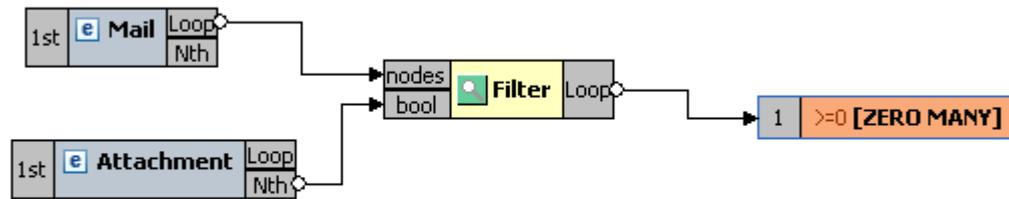


Figure 10.4.31: Funclet easel

Add Link between two nodes

To make a link between two nodes placed on the Funclet easel, follow the steps below:

Click on the gray box on the source building block. A small circle appears, as shown in Figure 10.4.32. This represents the starting point of the link and the output box of the building block.



Figure 10.4.32: Source node

1. 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 10.4.33.



Figure 10.4.33: Linking the Source and the destination node

2. Release the mouse. A link between the two nodes is created, as shown in Figure 10.4.34.

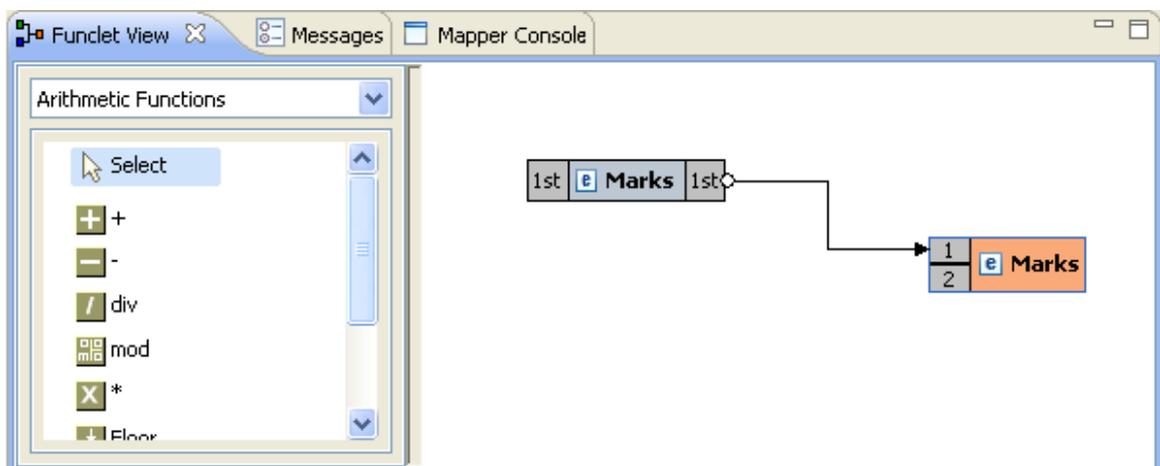


Figure 10.4.34: Linking Source and Destination nodes

Delete link between two nodes

To delete a link between two building blocks,

Click on the ending point of the link and drag it to an empty area in the Funclet easel, as shown in Figure 10.4.35.

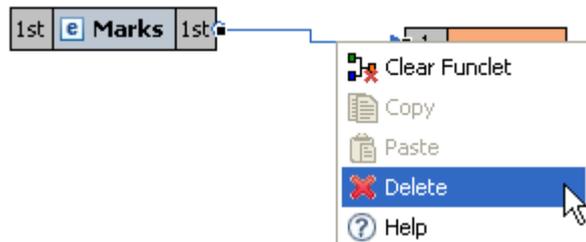


Figure 10.4.35: Deleting link

- Now, release the mouse. This removes the link between the corresponding nodes.

Delete node from Funclet easel

Select the corresponding building block and right-click on it. The shortcut menu appears as shown in Figure 10.4.36.

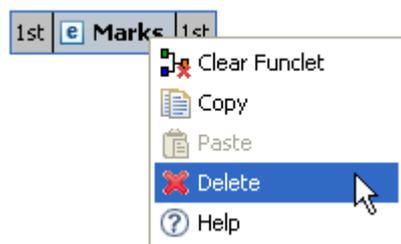


Figure 10.4.36: Pop-up menu

- Click **Delete** to delete the selected building block.

Open Function Help

The description for a predefined function can be viewed by clicking Help in the right-click menu of a Function node.

10.5 Creating Mappings

Mappings are defined between nodes of the Input and Output structures. The Structure is displayed in a tree form.

10.5.1 Understanding Types of Nodes

Mappings are defined between nodes of the Input and Output structures. These nodes can be divided into four types:

- Element Node:** This type of node contains an XML element.
- Text Node:** This type of node contains an XML element only.
- 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.

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 10.5.1 the Mail-List element can contain zero or many Mail nodes.

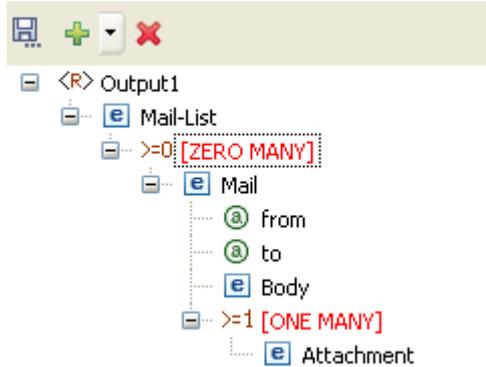


Figure 10.5.1: Example of Zero to Many control node

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 10.5.2 the Mail node can contain one or many occurrences of the Attachment node.

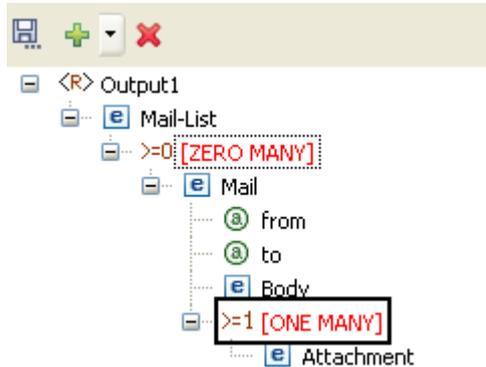


Figure 10.5.2: Example of One to Many control node

Control Node [Choice]: This Control node specifies that only one of the descendant nodes can exist in the parent node. For example in Figure 10.5.3 TifosiService node can have either Java node or Win32 node, but not both.

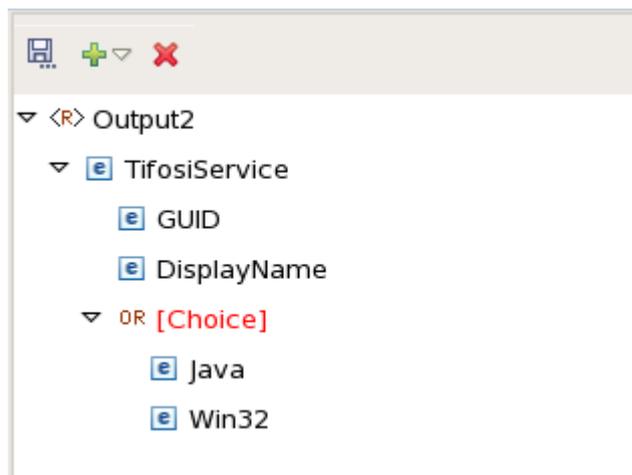


Figure 10.5.3: Example of Choice Control Node

A structure can also contain optional nodes. This type of node specifies that either zero or one occurrence of this node can exist in its parent node. For examples, in Figure 10.5.4, Student node can have either zero or one occurrence of the Nick-Name node. An Optional node (element/attribute) is displayed in green color.

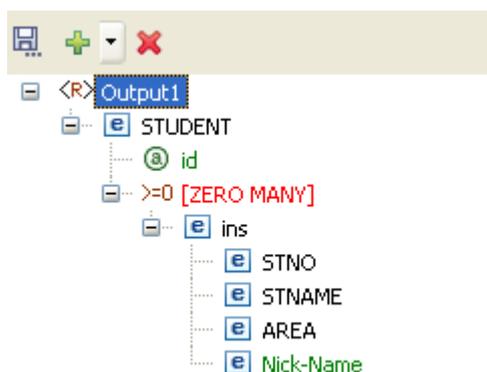


Figure 10.5.4: Example of Optional Node

10.5.2 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.

10.5.2.1 Name-to-Name Mapping

Now create mapping from Name-to-Name, as shown in Figure 10.5.6.

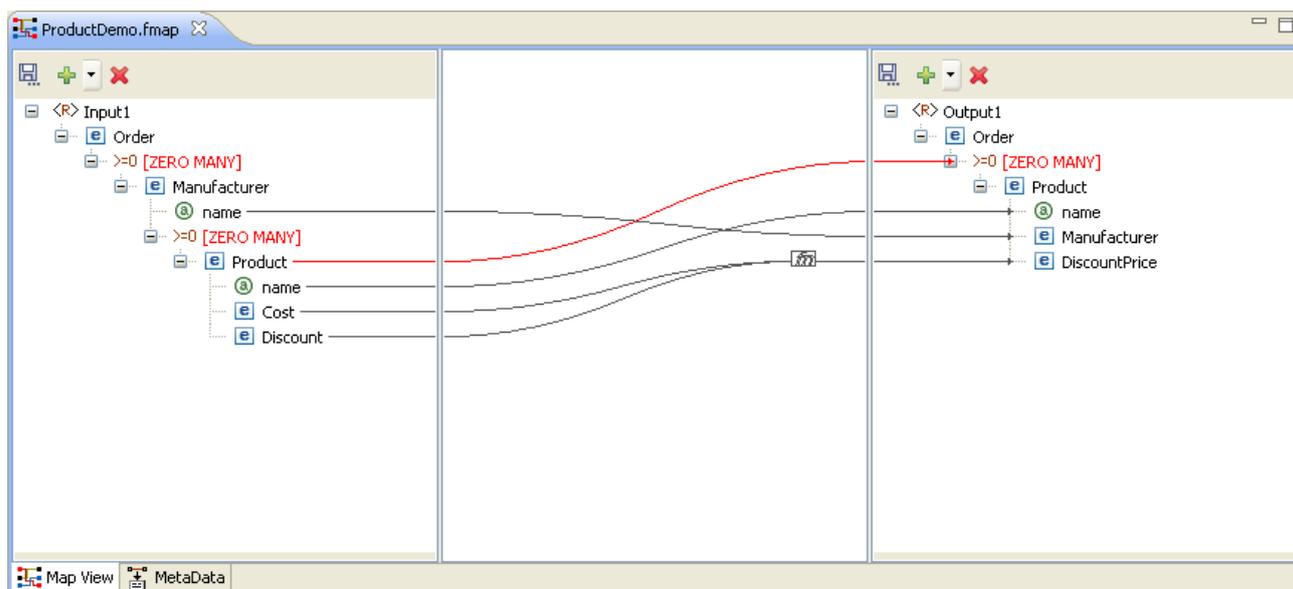


Figure 10.5.6: Name-to-name Mapping

The Funclet Wizard shows a link starting from the 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.

10.5.2.2 For-Each Mapping

When an Input Structure node can have multiple instances and the user wants 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 10.5.7 shows an instance of a For-Each mapping.

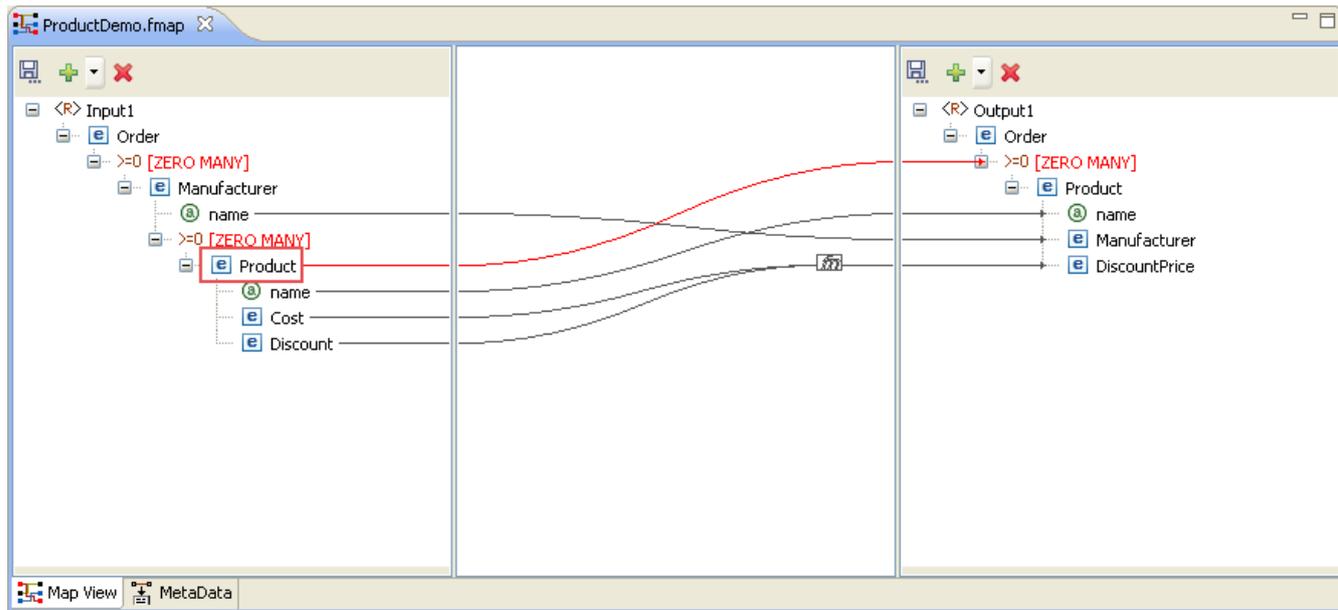


Figure 10.5.7: For-Each Mapping

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 the For-Each context.

So, in Figure 10.5.7 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.

10.5.3 Duplicating a For-Each Mapping

There may be situations in which one 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 the student element in the output structure DTD, as shown in Figure 10.5.8.

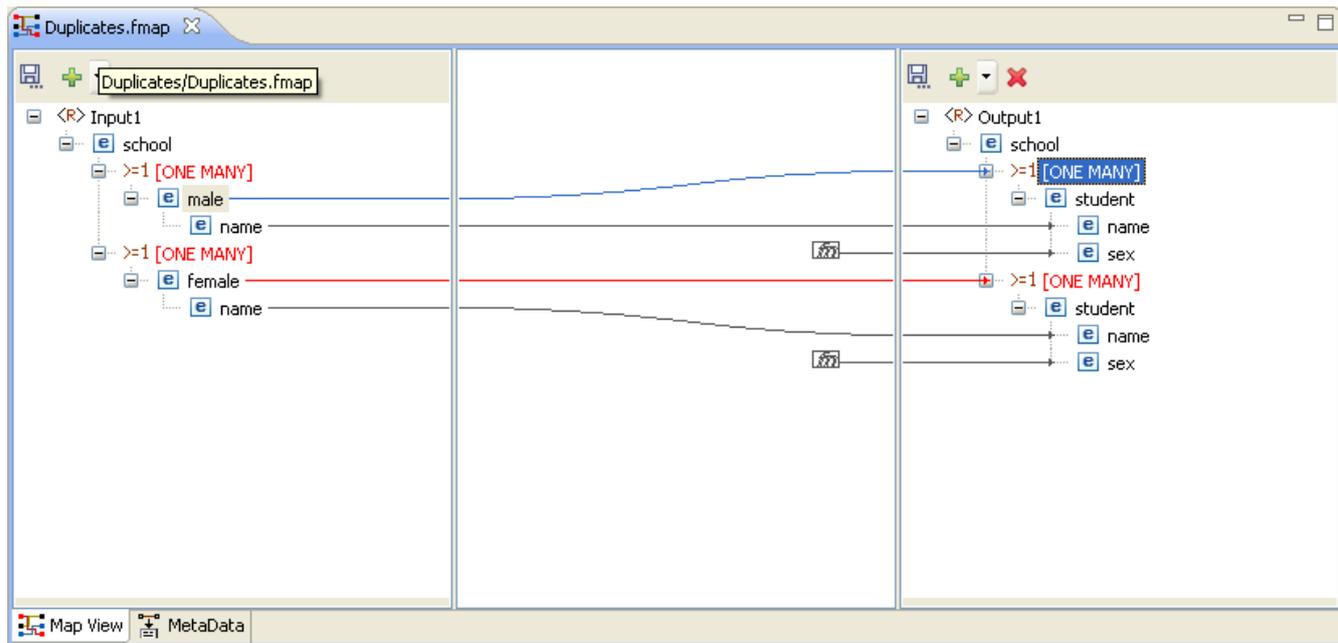


Figure 10.5.8: Mapping a node to One Many control node

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 message dialog box is displayed as shown in Figure 10.5.9.

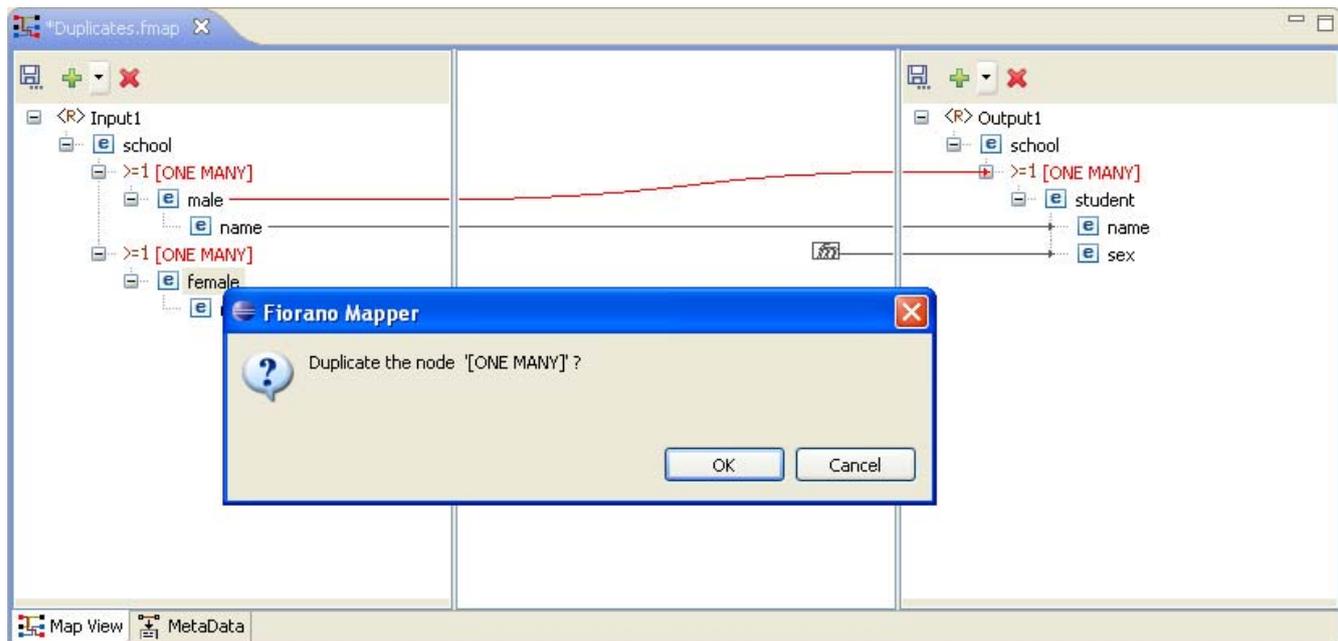


Figure 10.5.9: A shortcut menu prompts you to duplicate the node

Click **OK** in the message dialog box to create a duplicate node. A mapping is created as shown in Figure 10.5.10.

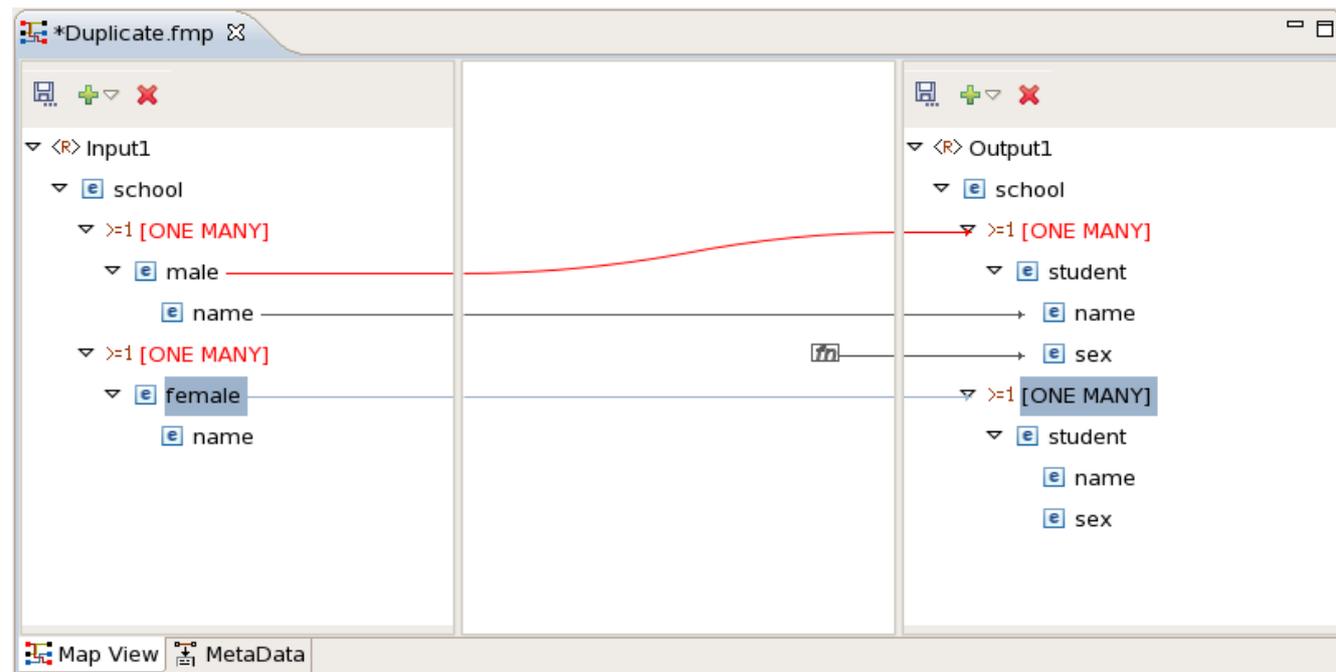


Figure 10.5.10: The One Many Node is Duplicated

10.5.4 Linking Nodes to Define Mappings

A Mapping is defined in the Fiorano eMapper 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

10.5.4.1 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 **Child to Child** option in the tool bar, as shown in Figure 10.5.11.

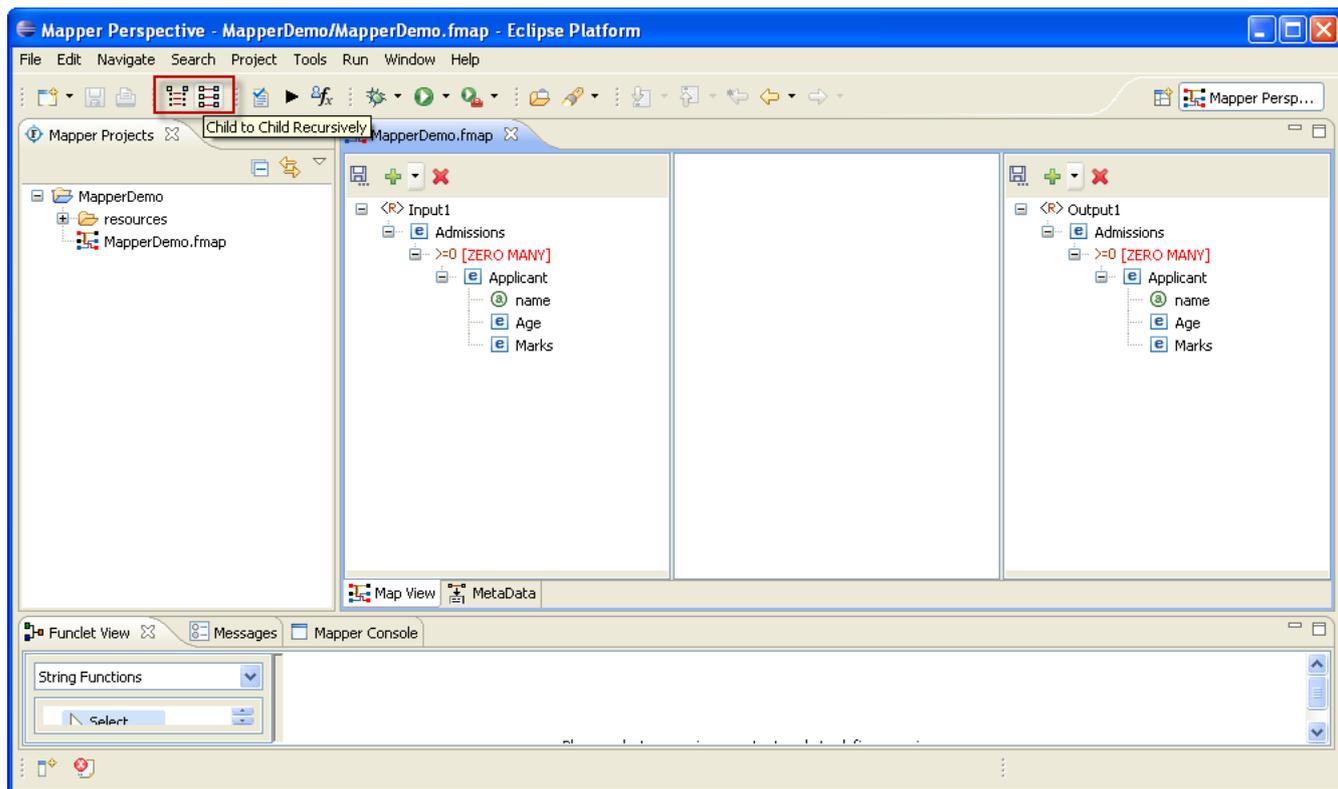


Figure 10.5.11: Creating Automatic Mapping between child nodes

10.5.4.2 Using the Visual Expression Builder to Define Mappings

The Visual Expression Builder (VEB) is a useful feature of the eMapper 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. Select the DiscountPrice output node, the Funclet View of the eMapper Perspective is displayed as shown in Figure 10.5.12.

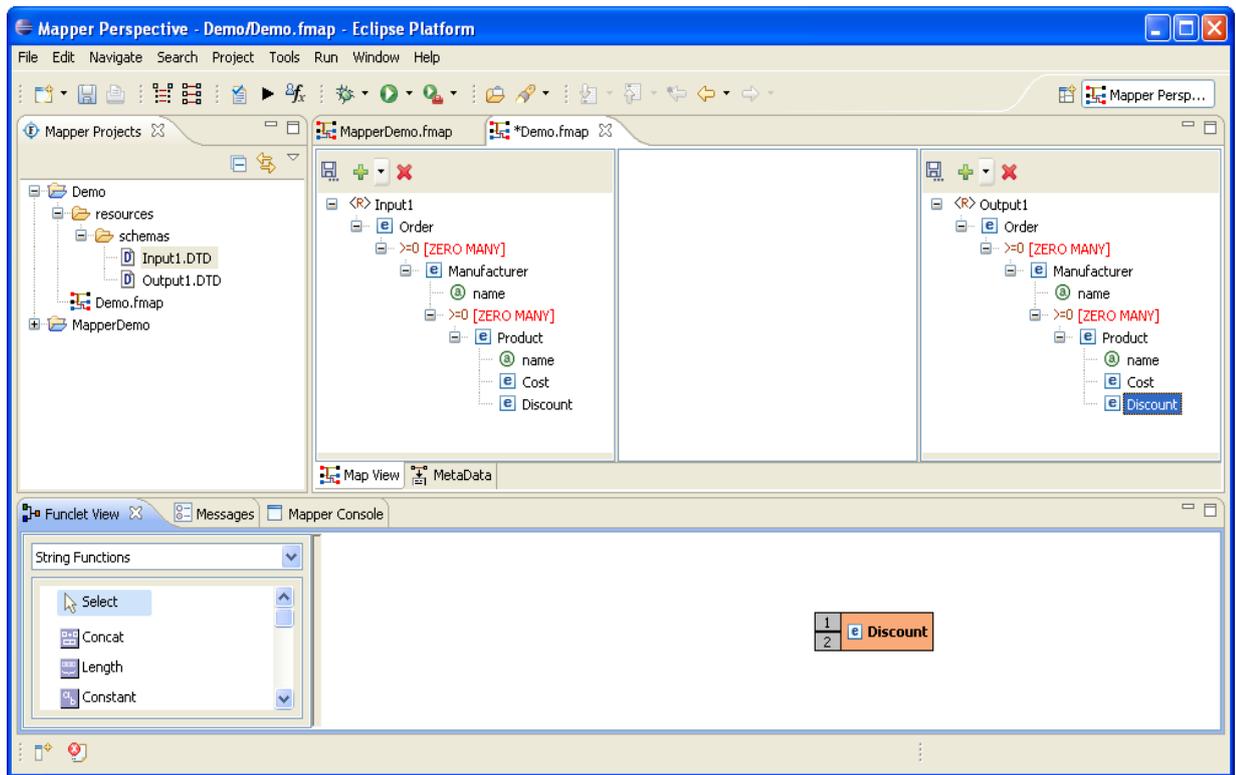


Figure 10.5.12: Selecting the Output Node for Mapping

2. The selected Output node is automatically displayed in the Function easel, as shown in the Figure 10.5.12. To add an input structure node to the mapping, drag it to the Funclet easel of the Visual Expression Builder. Here, 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 10.5.13.

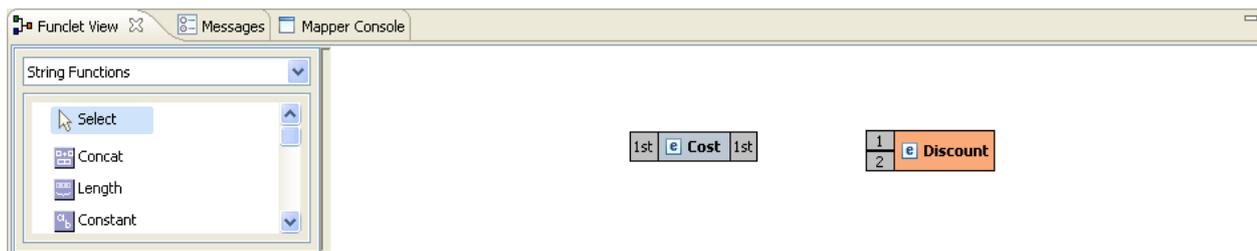


Figure 10.5.13: Dragging an Input node

3. To subtract the value of Discount input node, the subtract function from the Funclet Palette can be used. 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 10.5.14.

4. Select **Arithmetic Functions** from the list. The Arithmetic functions are displayed in the **Funclet 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 10.5.15.

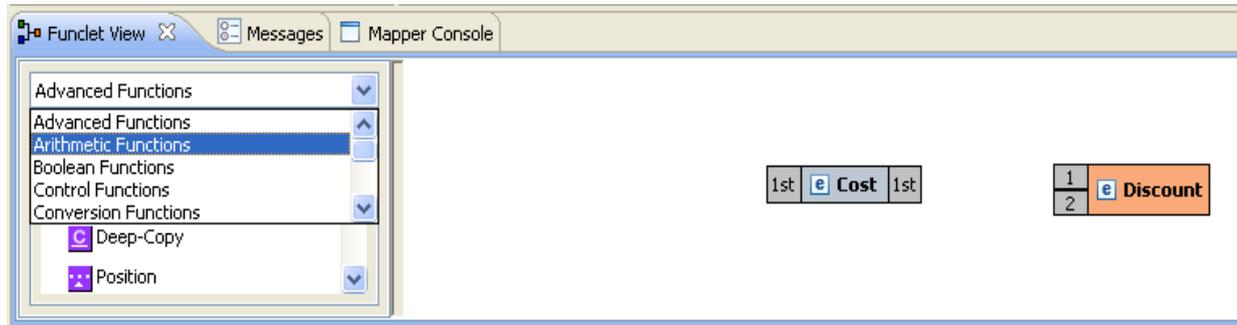


Figure 10.5.14: Selecting the Arithmetic Function Category in the Funclet palette

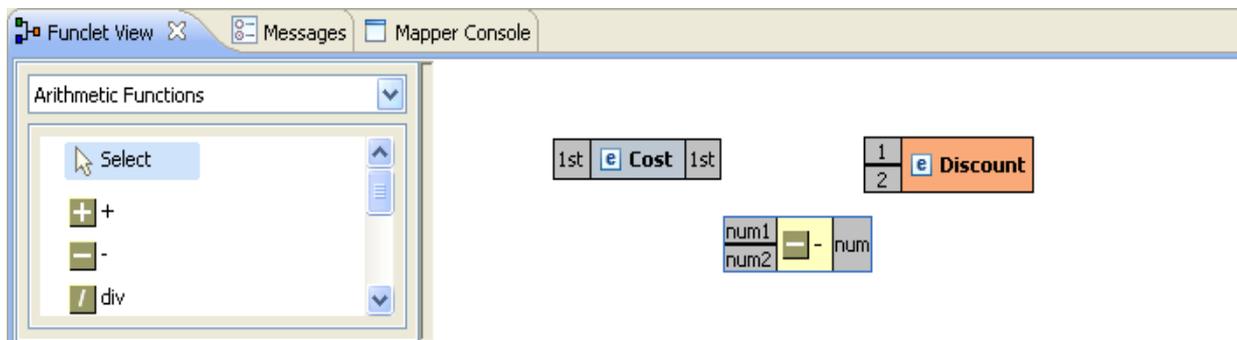


Figure 10.5.15: Adding the Subtract function

5. Next, add the Discount input node to the Funclet easel.

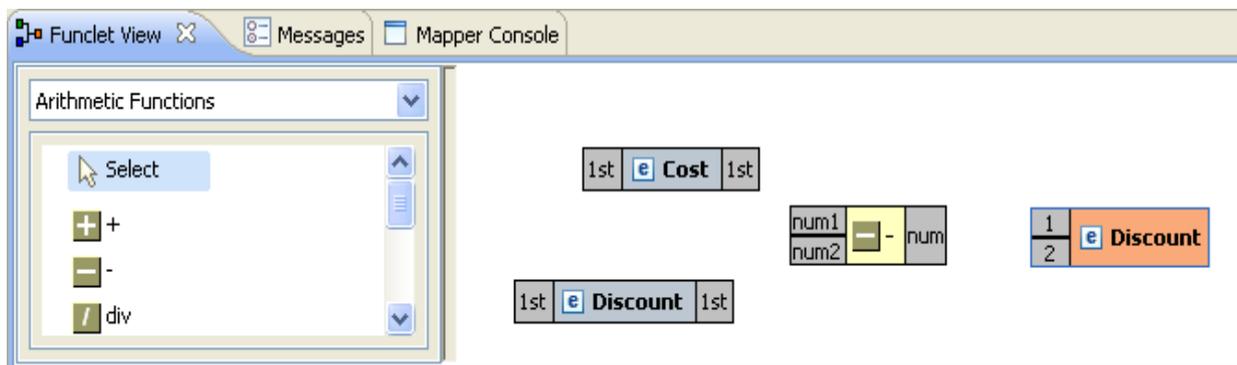


Figure 10.5.16: Adding another input node

- To define a mapping, links should be defined between these nodes. The Discount output is the difference between the Cost and Discount input nodes. To achieve this, the Cost and Discount nodes should be connected to the input pins (num1, num2 respectively) of the subtract function and its output pin should be connected to the input pin of the Discount output node.

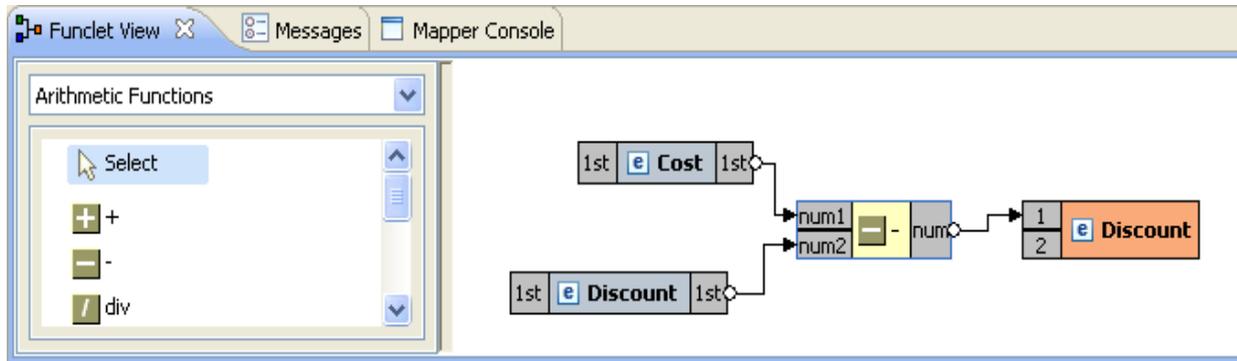


Figure 10.5.17: The final mapping is defined

- The required mapping is defined as shown in Figure 10.5.17.

10.5.5 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:

- Load the XML, DTD, or XSD input structure or structures.
- Load the XML, DTD, or XSD output structure.
- Link the Input XML Structure node(s) to the Output XML Structure node.

The following restrictions and conditions apply when mapping one XML format to another:

Nodes that do not have any content cannot be mapped. However, the child nodes of these nodes can be mapped provided they can contain content.

The SQL and advanced function categories are not available for XML to XML mapping

10.6 Adding User XSLT

eMapper also allows the user to customize the output of the transformation by adding custom xslt code to the generated XSLT. XSLT code snippets can be added 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, eMapper allows further refinement on the auto-generated output.

As an example, consider a case where the eMapper generates an output that contains elements not required by the user. In this example, the eMapper 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 eMapper 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, the User XSL feature can be used to customize the output and avoid the generation of unmapped tags. To provide a user defined xsl

Right-click the <unmapped element> in the output structure and select the **User XSL** option from the shortcut menu as shown in Figure 10.6.1.

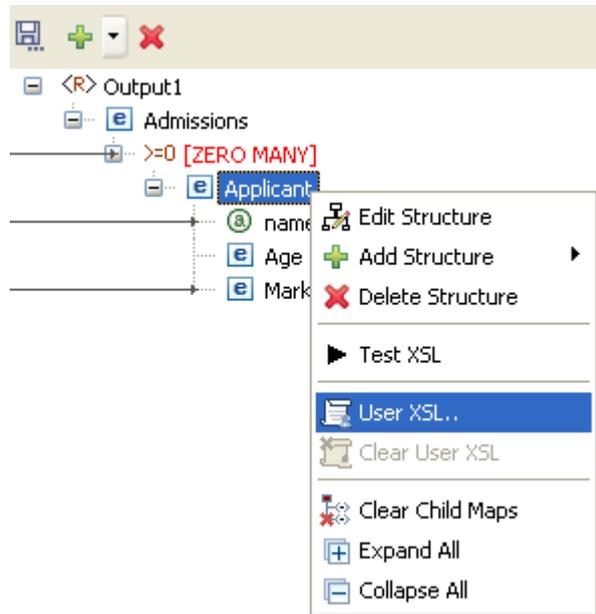


Figure 10.6.1: Selecting the User XSL option from the context menu

1. A dialog box appears which contains the xslt script. The xslt script displayed in this dialog box is partially editable. The editable regions, as shown in Figure 10.6.2, are marked by comments `<!--User code starts here-->` and `<!--User code ends here-->` at the beginning and ending respectively.

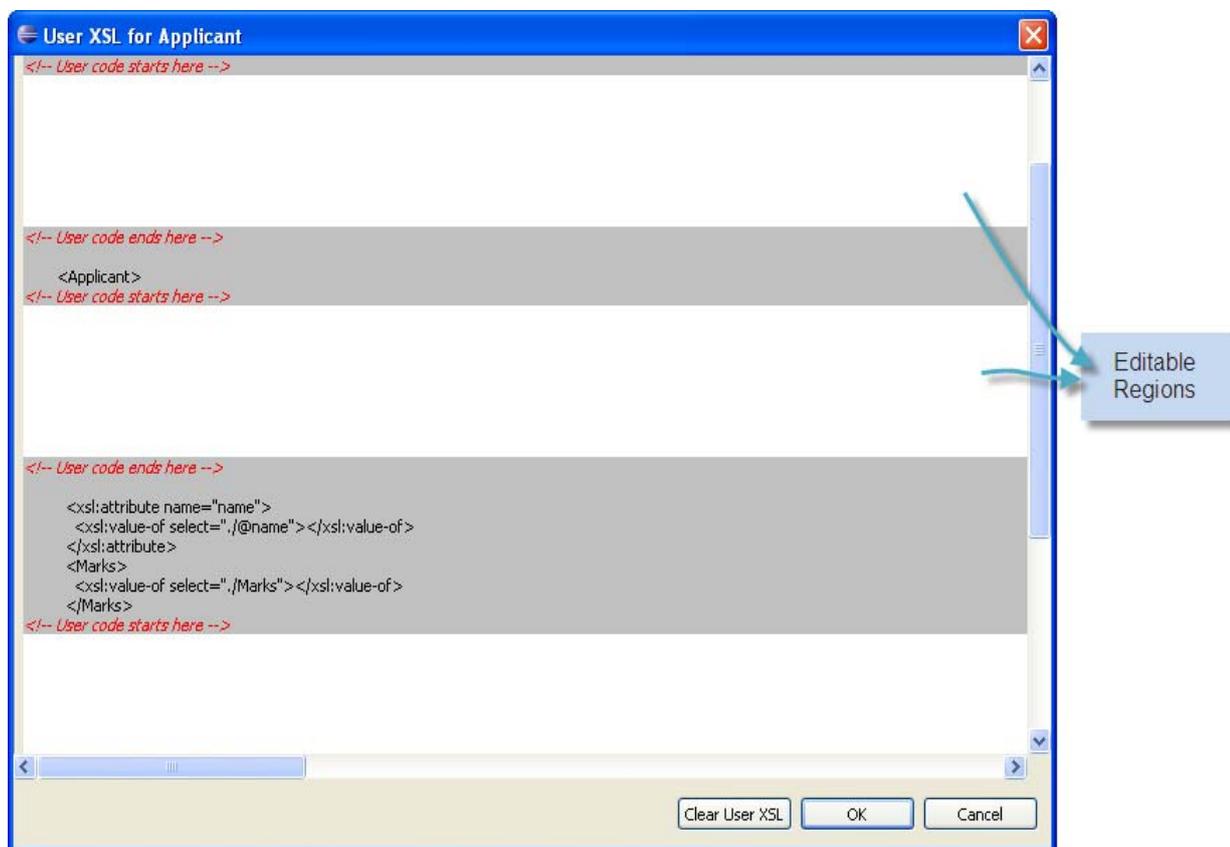


Figure 10.6.2: Editing the user xsl

As shown in Figure 10.6.2, XSL snippets can be added in the following four places:

- just above `<element>`
 - just below `<element>`
 - just above `</element>`
 - just below `</element>`
2. Add the required if code snippet in these regions.
 3. Click the **OK** button and the User XSL is saved for the element. It is denoted by the  icon next to the element/node in the structure as shown in Figure 10.6.3.

Applicant

Figure 10.6.3: Node with User XSL defined

4. The XSL can be tested it using **Test** option as described in the section [10.9 Testing the Transformation](#)

10.7 Working with derived types

When a complex type in an output/input structure has derived types, either by extension or restriction, the user can choose a derived type and the mappings can be defined using elements of selected derived type.

This is explained with an example. Screenshot of the sample schema used is shown below.

```
<?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:complexType name="Publication">
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="Author" type="xsd:string"/>
      <xsd:element name="Date" type="xsd:gYear"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="SamplePublication" type="Publication"/>
  <xsd:complexType name="BookPublication">
    <xsd:complexContent>
      <xsd:extension base="Publication" >
        <xsd:sequence>
          <xsd:element name="ISBN" type="xsd:string"/>
          <xsd:element name="Publisher" type="xsd:string"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="PaperPublication">
    <xsd:complexContent>
      <xsd:restriction base="Publication">
        <xsd:sequence>
          <xsd:element name="Date" type="xsd:gYear"/>
          <xsd:element name="Location" type="xsd:string"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
```

Figure 10.7.1: Schema with derived types

The schema provided in Figure 10.7.1 contains an element **SamplePublication** of type **Publication**. The type **Publication** has two derived types: **BookPublication**(extension) and **PaperPublication**(restriction).

When the schema is loaded in Mapper, the element **SamplePublication** is shown in Mapper.

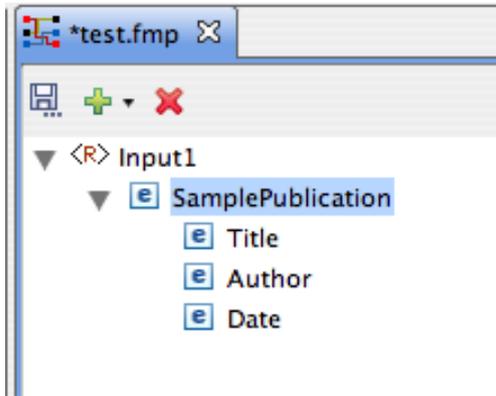


Figure 10.7.2: Sample Publication with default type

Since the type Publication has derived types, user can change the type of the element SamplePublication. All the derived types will be shown when Right-clicked on the SamplePublication element and the user can select the required derived type as shown in Figure 10.7.3.

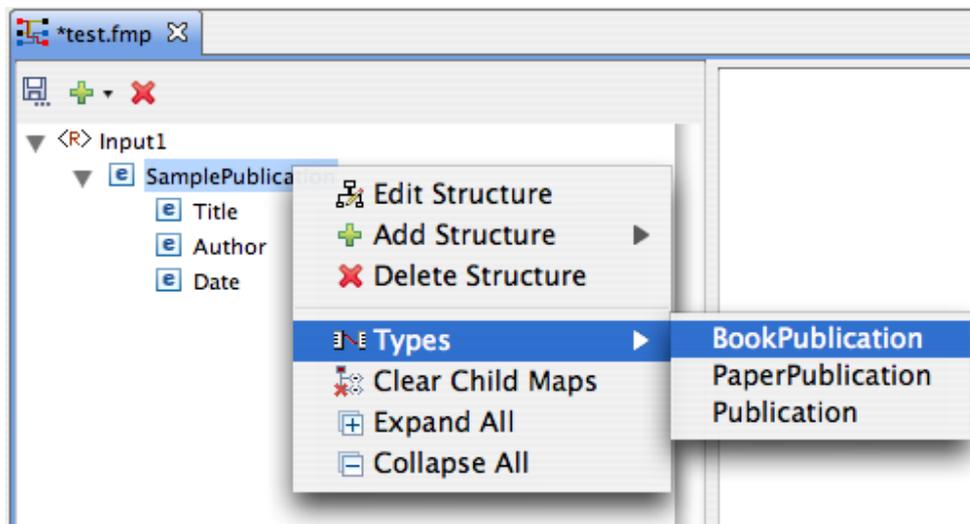


Figure 10.7.3: Available derived types

When a different type is selected, the structure will be refreshed to show the selected type.

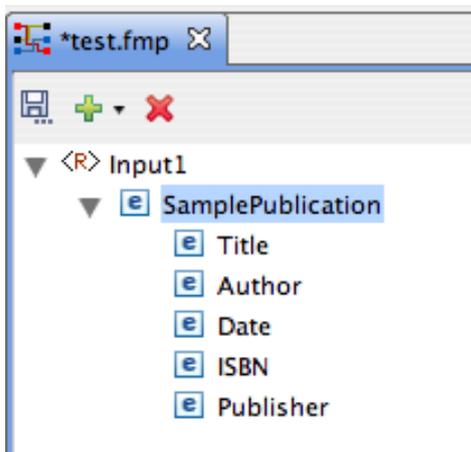


Figure 10.7.1: SamplePublication element when BookPublication type is chosen

Mappings can be defined assuming that the element SamplePublication is of type BookPublication.

Note: When derived types are used, the input/output must comply with the type used.

10.8 Create/Edit User Defined Function(s)

Custom Functions can be added to the User Defined Functions category of the Function Palette. Functions can be created by performing the following steps.

- Go to Tools menu in the eMapper perspective and click Create/Edit User Defined Function(s). The **Extensions Dialog** is shown as shown in the Figure 10.8.1.



Figure 10.8.1 Extensions Dialog

- This dialog has a list of all extensions that are defined.
- To create a new extension, type the name of the extension to be created in the text area provided in this dialog and click **OK**.
- To edit one of the existing extension, click on the extension and then press **OK**.
- The **Script Function Wizard** will appear as shown in Figure 10.8.2. The wizard has two pages viz **Script Information Page** and **Function Page**.

Script Information Page:

- Extensions can be defined either in **Javascript** or **Java** language. The language of the extension being added can be specified from the **Language** combo present in the **Script Information Page**
- The Javascript or the qualified name of the Java class, depending on the language of the extension, needs to be provided in this page.
- To add Javascript functions, provide the Javascript and click **Next**. The script will be processed and the list of functions will be populated in the **Function Page**.
- To add Java Functions, provide the qualified name of the Java class and click **Next**. The list in the **Function Page** will be populated with all the **public static** functions defined in this class.

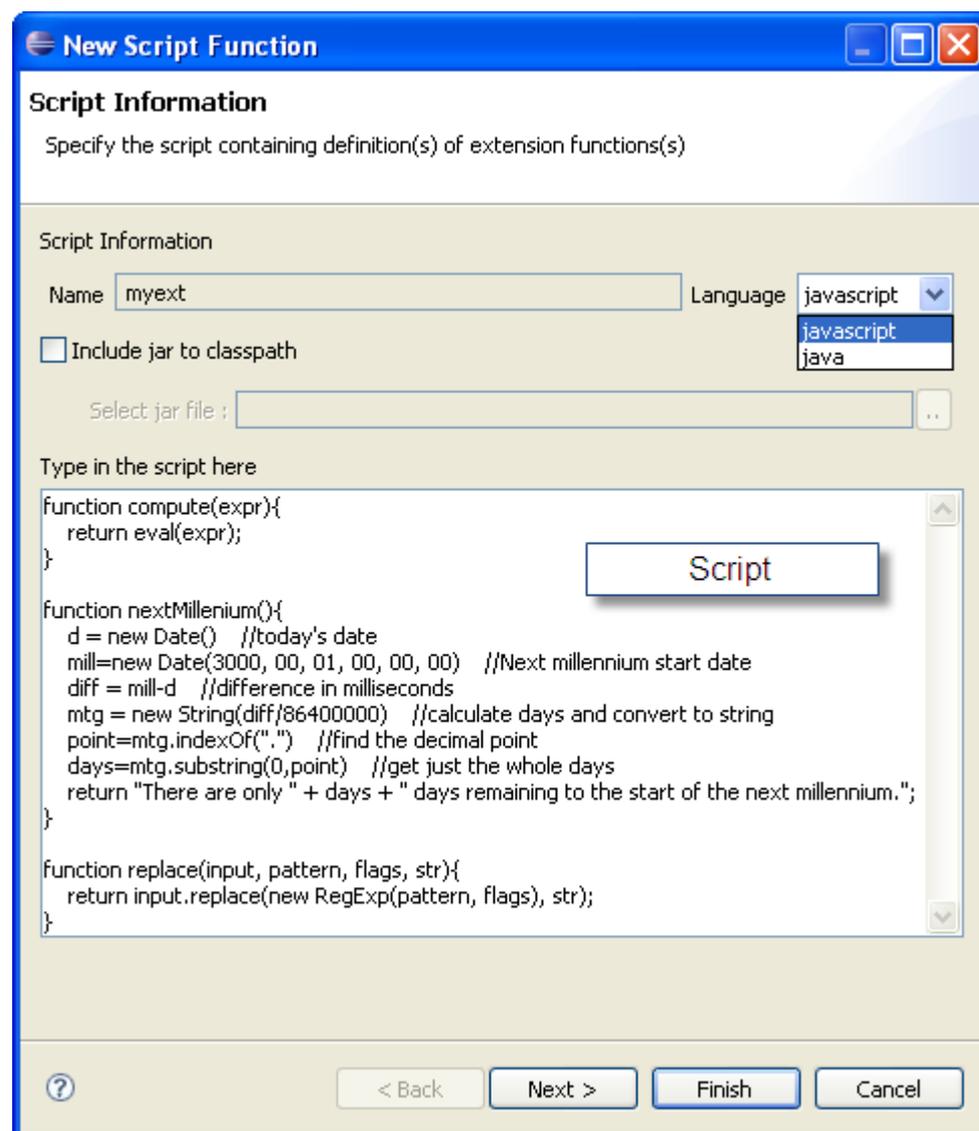


Figure 10.8.2 Script Information Page

Function Page:

- The **Function Page** shows the list of functions that were defined in the **Script Information Page**. The user can select the desired functions and the selected functions will be added to the Function palette under the **User Defined Functions** category.

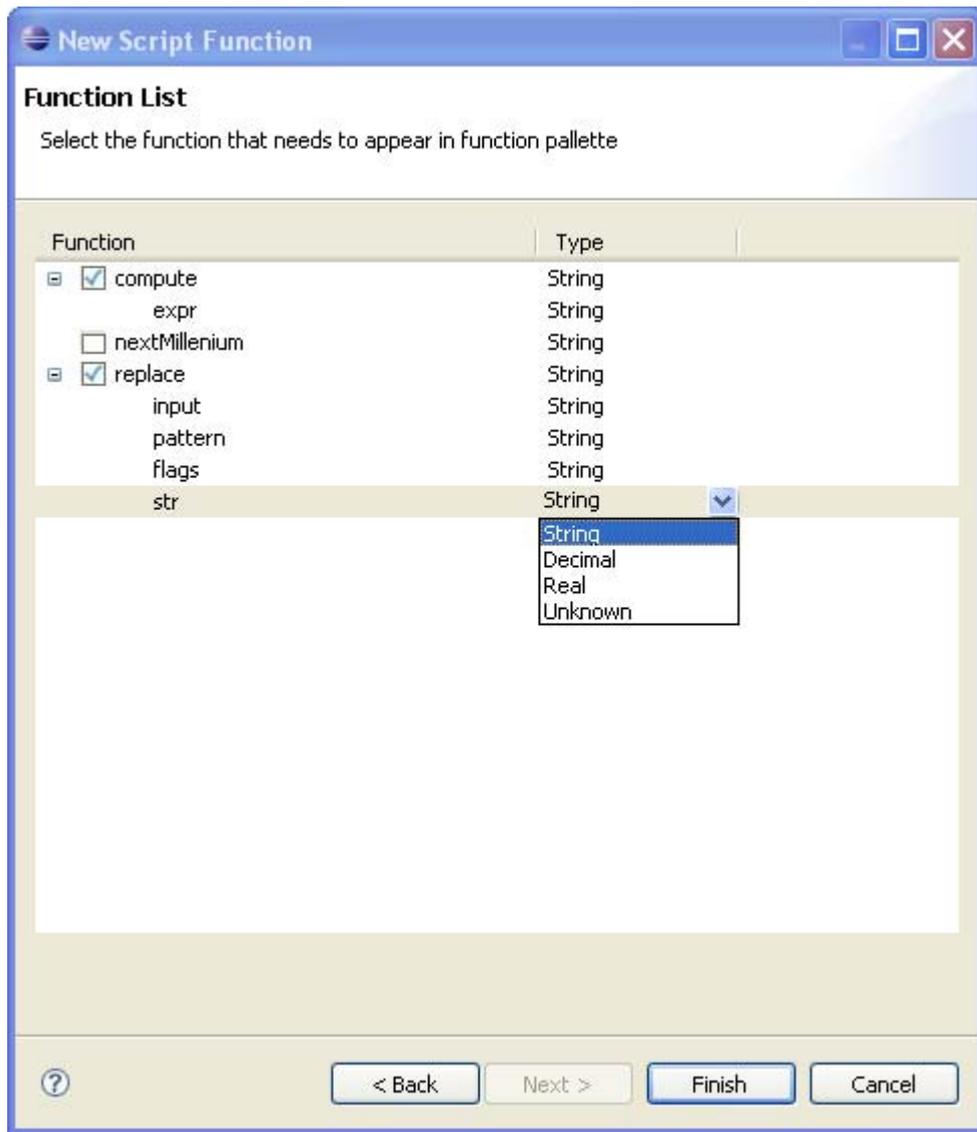


Figure 10.8.3 Function Page

Note: While adding Java functions, the user might have to add a .jar file to the classpath in order to fetch the list of functions. This can be done through the Include jar to classpath option provided in the Script Information Page. Click the browse button and add the required jar file.

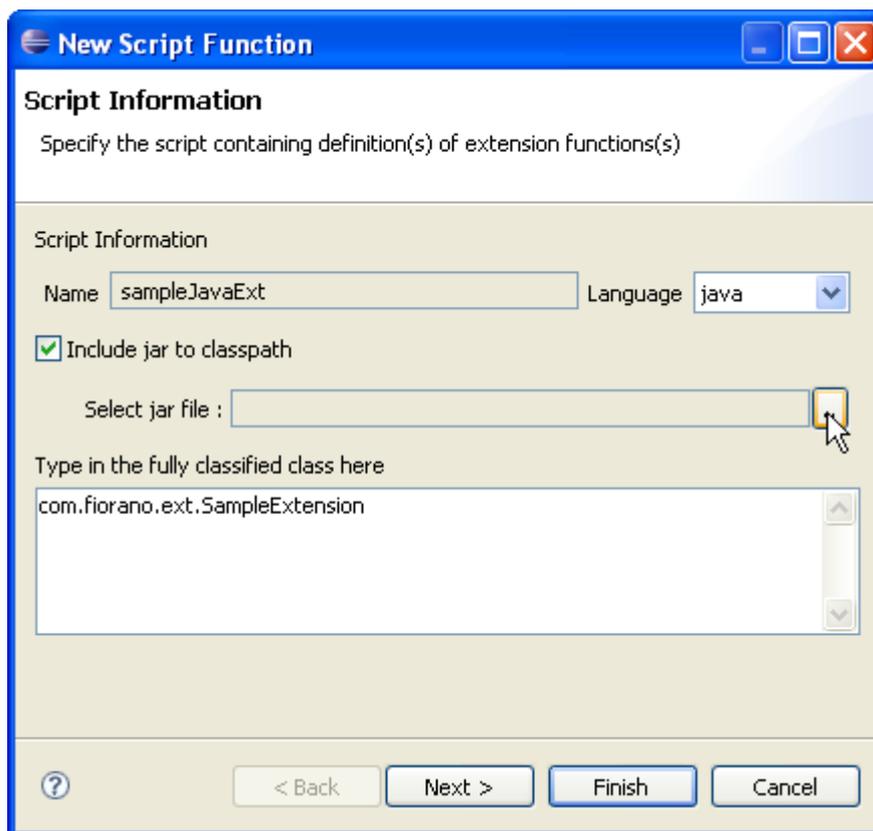


Figure 10.8.4 Adding Jar to classpath

10.9 Testing the Transformation

The transformation created in a eMapper project can be tested by performing the following steps:

Click **Tools > Test Mapping** in the Fiorano eMapper's menu bar, as shown in Figure 10.9.1 or click the **Test** button in the tool bar

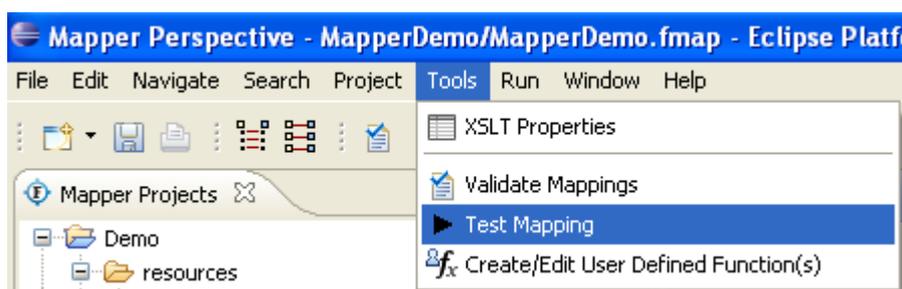


Figure 10.9.1: Invoking the Test option

The Test XSL wizard is displayed, as shown in Figure 10.9.2. The Transformation can be tested by following these steps:

Providing MetaData

- This wizard has two pages, the **MetaData** page and the **Test Mappings** page.
- The output structure for which the transformation is being tested can be chosen from the combo provided at the top of the MetaData page.

- The text area, by default shows the transformation generated automatically by the eMapper for the specified output structure. This transformation can also be modified by deselecting the **Always Load From eMapper** button.
- This allows the user to modify the XSL. Specify the XSL and move to the next page to perform the transformation.

Input XMLs

- The Test Mappings page has two tabs, **Input XML** tab and **Output XML** tab.
- The Input XML tab, as the name suggests, is used to provide the input XMLs. This tab in turn has sub tabs for each input structure loaded in the eMapper.
- A sample XML can be generated from the corresponding structure by clicking the **Generate Sample XML** button present in the tool bar of an input tab.

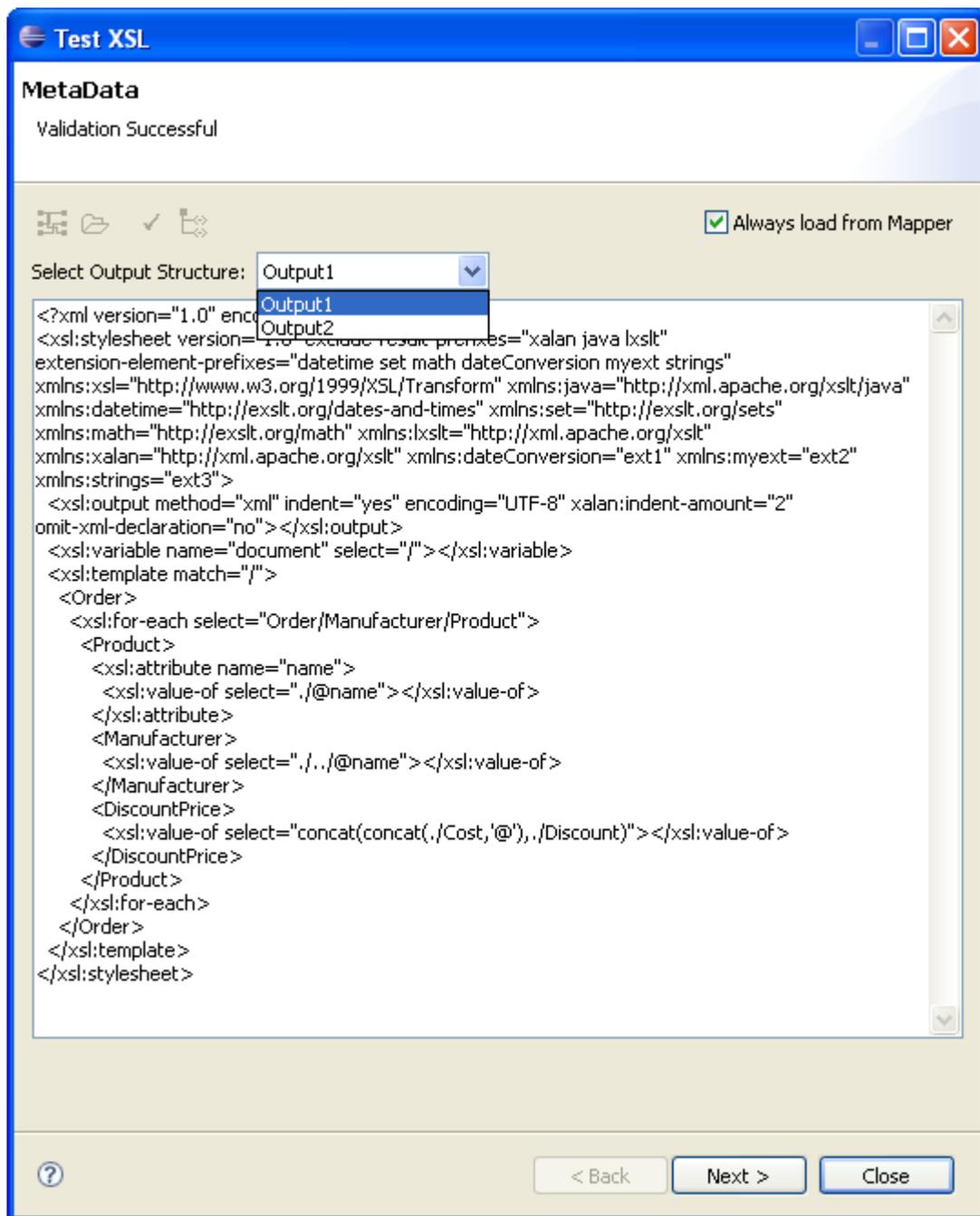


Figure 10.9.2: Metadata Page

- The **Generate Sample XML** dialog box is displayed, as shown in Figure 10.9.3. The default values are appropriate in most situations. Provide the desired values and click OK to generate a sample XML.

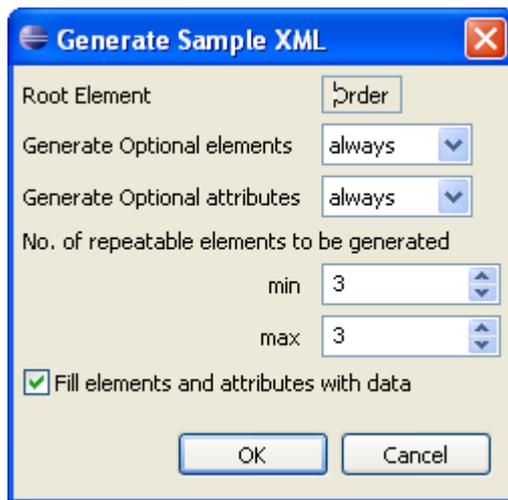


Figure 10.9.3: Selecting the sample Input XML generation options

- The sample XML is generated in the Input XML tab as shown in Figure 10.9.4.

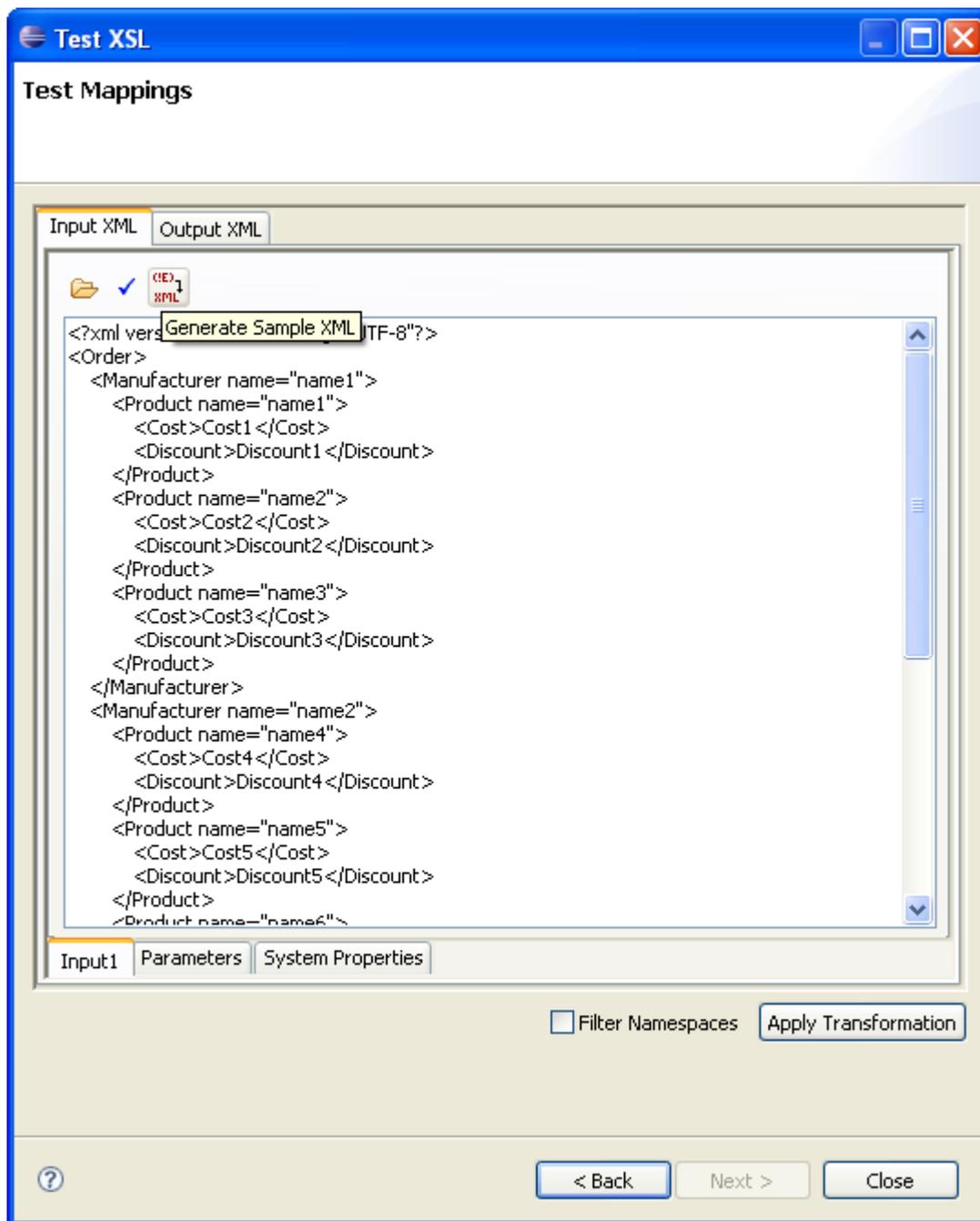


Figure 10.9.4: A Sample Input XML

- Options to load input XML from a file and validate the input XML are provided in the tool bar. Validation errors if any will be displayed at the top of the wizard.
- The Input XML tab also contains a **Parameters** tab that can be used to define parameters to be used while transformation. The required parameters can be added to the table provided in this tab.
- System Properties, if needed, can be defined from the **System Properties** tab. For example, while using Lookup Functions, a system property needs to be defined pointing to the `db.properties` file which holds data for oracle data base

Testing the transformation

- Click the **Apply Transformations** button to test the defined transformation.
- The output XML is displayed in the Output XML tab, as shown in Figure 10.9.5.

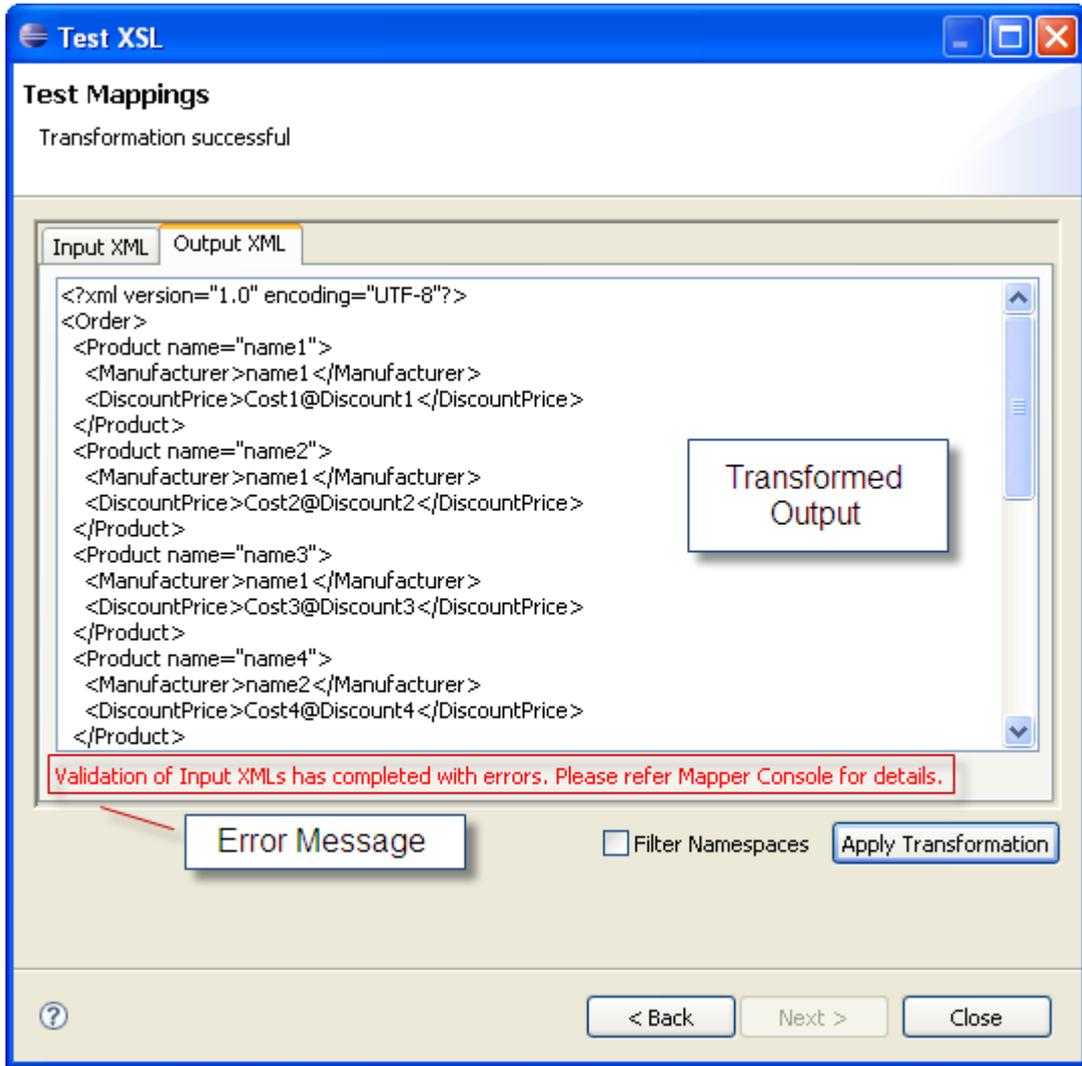


Figure 10.8.5: The Output XML resulting from the Transformation

10.10 Managing Mappings

Creating mappings is as simple as dragging an input node and dropping it on a target structure node. The eMapper also provides few other options to manage mappings.

10.10.1 Exporting eMapper Project

To export the eMapper project, right-click on the project node and select **Export**. Select the destination file and the project will be exported as an archive.

10.10.2 Importing Project from the File

A eMapper project can be imported either from an existing .tmf file or from another eMapper project.

To import the project from an existing project:

1. Right-click on the **Mapper Projects** node and choose **Import**. The Import wizard is shown.

2. Choose the zip file from which the projects are to be imported. The Import wizard is opened with a list of all available projects. Choose the required projects and click **Finish** to import the projects.

To import mappings from a .tmf file:

1. Right click on **Mapper projects** node and click **New**
2. In the New eMapper Project wizard, provide a valid project name and select the Load from tmf file option.
3. Load the tmf file using the browse button provided and click Finish.
4. The new eMapper project with the Mappings from the provided .tmf is created in the workspace.

10.10.3 Copying functions in a Mapping

You can copy functions within a mapping project and across mapping projects. To copy a function

Select the function in the funcelet view and click **Copy** from the right-click menu as shown below in Figure 10.10.1.

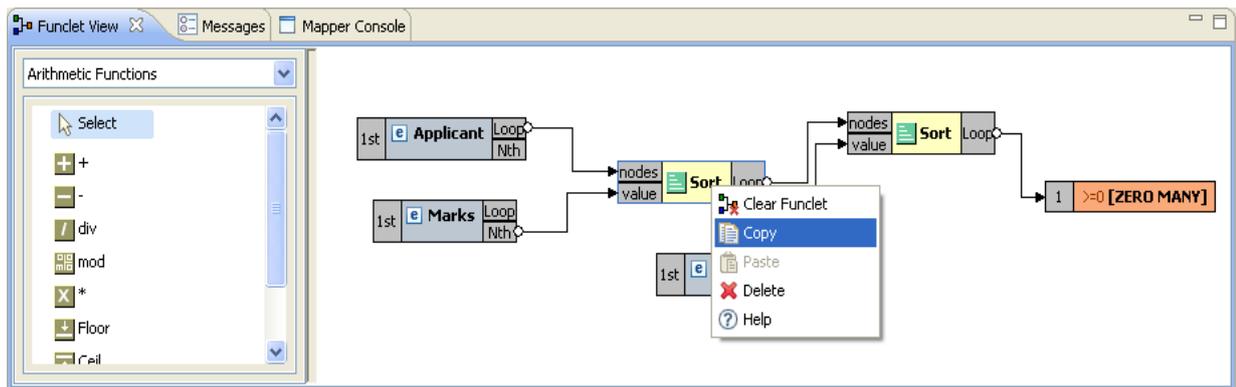


Figure 10.10.1: Copying a function

Click Paste and the function is pasted in the funcelet view and can be reused within or even across mappings.

10.10.4 Clearing All Mappings

To clear all the mappings between the Input and the Output Structure,

1. 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.

10.10.5 Managing XSLT Properties

You can also manage the XSLT properties of the output XSLT. To do this:

Click **Tools> XSLT Properties**. The XSLT Properties dialog box is displayed as shown in Figure 10.10.2.

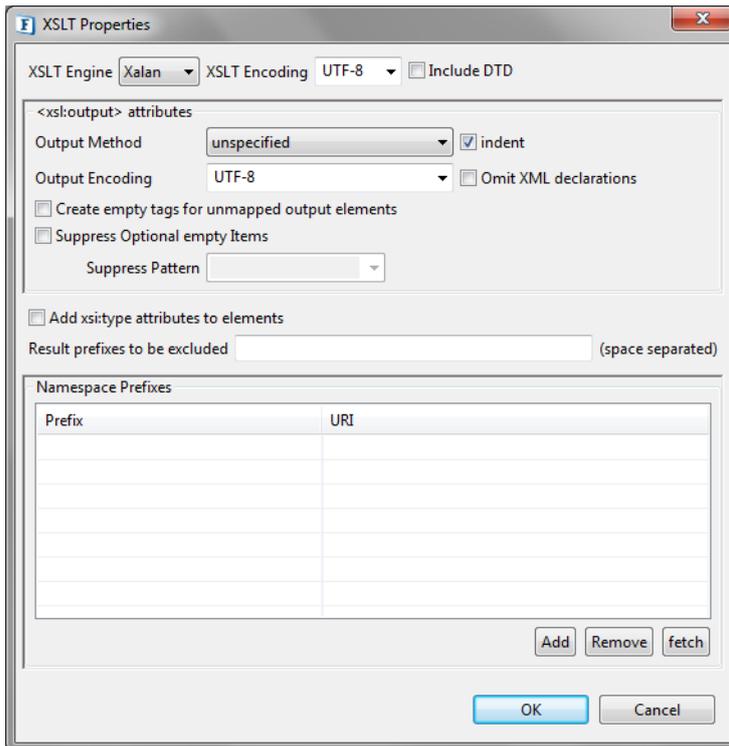


Figure 10.10.2: Viewing XSLT Properties

This dialog box contains the following components:

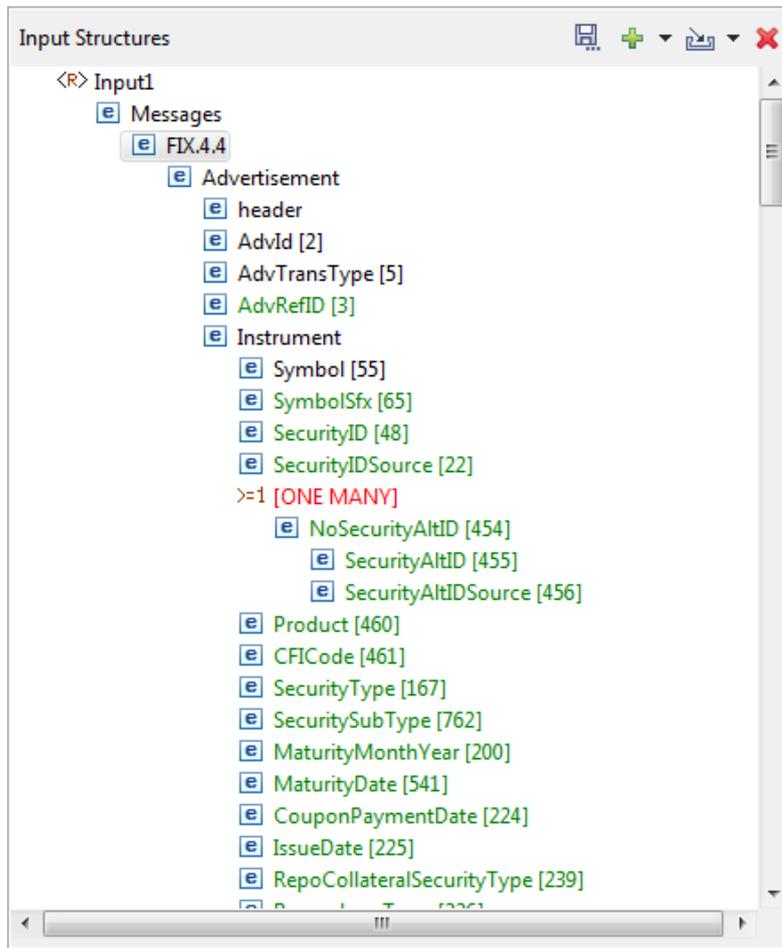
1. **XSLT Engine:** The XSLT Transformation Engine to be used for performing the transformation. The available implementations are Xalan, Saxon and XSLTC
2. **XSLT Encoding:** Specifies the encoding of the generated XSL.
3. **Include DTD:** Select this option to include the internal specified DTD in the transformation output. This option is disabled by default.
4. **<xsl output: attributes>**
 - a. **Output Method:** Select the method of output after transformation from the drop-down 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.
5. **Omit-xml-declaration:** Specifies whether the output XML generated should contain XML declarations or not.
6. **Create empty tags for unmapped output elements:** Select this option to create empty tags for all unmapped elements of the output structure.
7. **Add xsi:type attributes to elements:** Selecting this option will add xsi:type attribute to all the elements in the output XML.
8. **Suppress optional empty items:** Select this option for defining a mapping to an output node, always generate the output nodes in output xml since event input xml has no matching nodes. It is sometimes desirable not to generate optional output nodes if no input matching node is found in input xml. This requires using conditional mapping. You can specify such conditional mapping by using "User XSL" feature. eMapper can generate such conditions automatically for optional elements if this option is selected.

10.11 FIX-XML Transformations

10.11.1 Loading FIX message in Mapper

An option to load FIX message is provided in the eMapper tool. A FIX message structure can be added by choosing **FIX** after clicking the Add Structure button (+) in the Input / Output Structures panel.

A list of different FIX versions and message types is displayed where the messages to be loaded can be chosen. Select the message types and click the **Load** button to load the messages types in to the workspace. Click **Finish** in the Add Structure wizard and the FIX message structure is loaded in Mapper as shown in the fig.

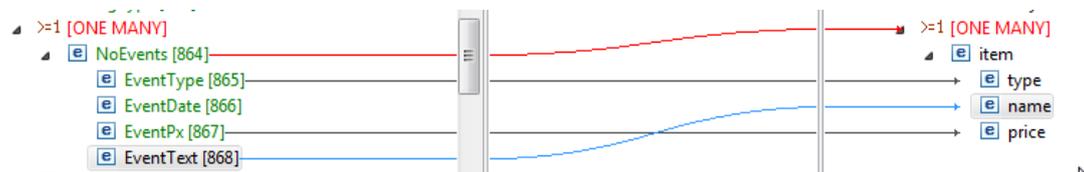


Each field in a message is represented by a node. Repeating groups are represented using a **[ONE MANY]** node. The first child of a **[ONE MANY]** node is the NumInGroup node, representing the number of times the fields of the group are repeated, and the repeating fields are added as children to this node. A Component type in the message is denoted by a node with the component's fields, groups and sub-components added as its children.

To map FIX message fields to an XML message, load the input FIX message type in the Input Structure and the output schema in the Output Structure panel. A mapping can be created between two nodes by dragging the input node and dropping it on the output node. Define mapping from the input FIX message to the output XML.

To map a repeating group to an output node that has [ZERO MANY] or [ONE MANY] cardinality, map the NumInGroup field to the control node ([ZERO MANY] or [ONE MANY]) node and map the child nodes to their corresponding output nodes.

This generates a for-each mapping. In the example below, a for-each mapping is defined between the Events group in the input FIX message and the **item** element in the output xml message which means that for each repetition of the group fields in the input, an output node <item> will be created.



Similarly, while mapping XML to FIX, the repeating element in the input has to be mapped to the [ONE MANY] node representing the group to generate a for-each mapping.

10.11.2 XSL generation

XSL is generated automatically when mappings are done. The generated XSL can be viewed in the **Metadata** tab of the editor.

10.11.3 Test Mappings

To test the XSL generated, click the Test button in the toolbar (), provide the input FIX message in the space provided and click **Apply Transformation**. The output xml will be generated.

Note: Since the mapper tool uses XSL to perform the transformation, and XSL does not allow non-XML messages as input, the input FIX message is modified by adding <Message> and </Message> tags at the beginning and ending of the message. Also, the non-printable SOH character will be replaced by '^' as Unicode characters are not allowed in an XML message.

10.11.4 Mapping XML to FIX

Mappings can be defined to transform an XML message to a FIX message by loading the XML structure in the Input panel and the FIX message type being generated in the output panel.

If a required field in the message is not mapped to any element in the input, an error message is shown in the Metadata Messages view.

The values for fields BeginString(8), BodyLength(9), MessageType(35) and Checksum(10) will be generated automatically for the output FIX message. Any mappings defined for these fields will be ignored.

10.11.5 Defining FIX to XML or XML to FIX transformation on a Route

XML to FIX or FIX to XML transformations can also be defined on a route in an Event Process. The following points have to be noted while using FIX-XML conversions on a route.

- Generated XSL uses java functions from the fix-xmlconverter.jar to fetch the value of field from the input FIX message string, calculate the body length / checksum of an output FIX message. This jar is present in the folder \$FIORANO_HOME/eStudio/plugins/com.fiorano.tools.mapper.runtime_1.0.0/jars folder. This file needs to be added to the classpath of the peer server so that the peer server can access the functions while performing transformations. This can be done by adding the relative path to this file under <java.classpath> in fps.conf file present in \$FIORANO_HOME/esb/fps/bin/ folder and in server.conf file present in \$FIORANO_HOME/esb/server/bin/ folder.
- As mentioned previously, the input document for an XSL should be an XML. While defining a route transformation from FIX to XML, the input FIX message needs to be modified by adding <Message> and </Message> tags at the beginning and ending of the message and by replacing the non-printable SOH character with '^'. This can be achieved by adding a java-script component which gets the text body of the JMS-Message, performs the required string operations (append and replace) and then set the resulting string back on the JMS-Message.
- Similarly for XML to FIX messages defined on a route, the output message will contain '^' instead of the SOH character. A Java-script component here can replace all the occurrences of '^' with the required component.

Chapter 11: Working With Multiple Servers And Perspectives

11.1 Active Server Node

In Online Event Process Development perspective, the user can add as many Enterprise Servers and Log into them and create, deploy, and run Event Processes on them.

Active Enterprise Server in context of eStudio means that states of Event Processes and other repositories will be shown corresponding to this particular server. Only one server is shown as an active server at any point of time but user can still work on all other servers by switching the active server. An active server switch can be made explicitly by selecting Activate option from the Enterprise Server node context menu or an inactive server will be automatically made active whenever the user wants to perform any action (Open Event Process, delete Event Process, CRC, Launch, Import, Export, and so on) on the inactive Enterprise Server.

The active server is displayed in Green color and the inactive servers are displayed in the default black color.

For instance, in the Figure 11.1.1 the server **EnterpriseServer_1** is Active and **EnterpriseServer** is inactive. All other views will be in accordance to the Active Enterprise Server (that is, EnterpriseServer_1). For example, Service repository and Service palette will show the services present in EnterpriseServer_1.

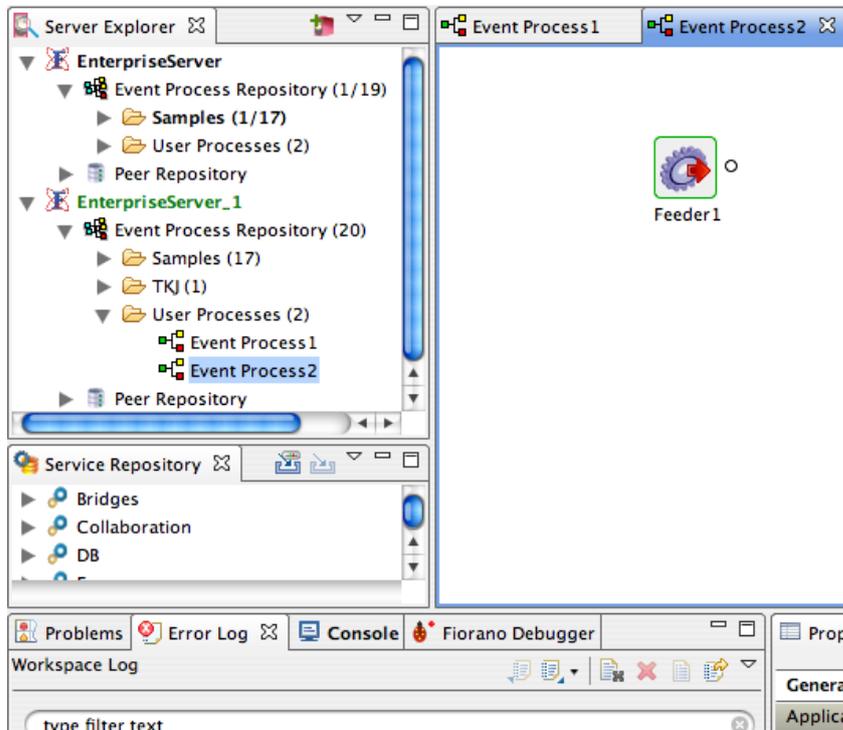


Figure 11.1.1: Multiple Enterprise Server login

If the user tries to perform any action on inactive server, a confirmation dialog is shown (user can set a preference to avoid the dialog each and every time) saying that the active server will be switched. When the switch happens, all the editors belonging to EnterpriseServer_1 are closed and editors corresponding to Enterprise Server are restored.

11.2 Switching of Active Server

With multiple servers alive there can be application deployed on different Enterprise Servers. But only event processes deployed on the Active Enterprise Server will be shown to the user.

On changing the Active Enterprise Server, editors for all the event processes deployed on to the previous Active Enterprise Server will be closed and these editors will be restored when that server becomes active (when the user selects any node in that server).

The following steps describe the active server switch:

1. Login into the two servers. The Service Palette and Service Repository shows the services present in the server to which the user has logged in recently (**Enterprise Server_1** in this case). Create some event processes in **Enterprise Server_1**, and keep the created event process editors open.

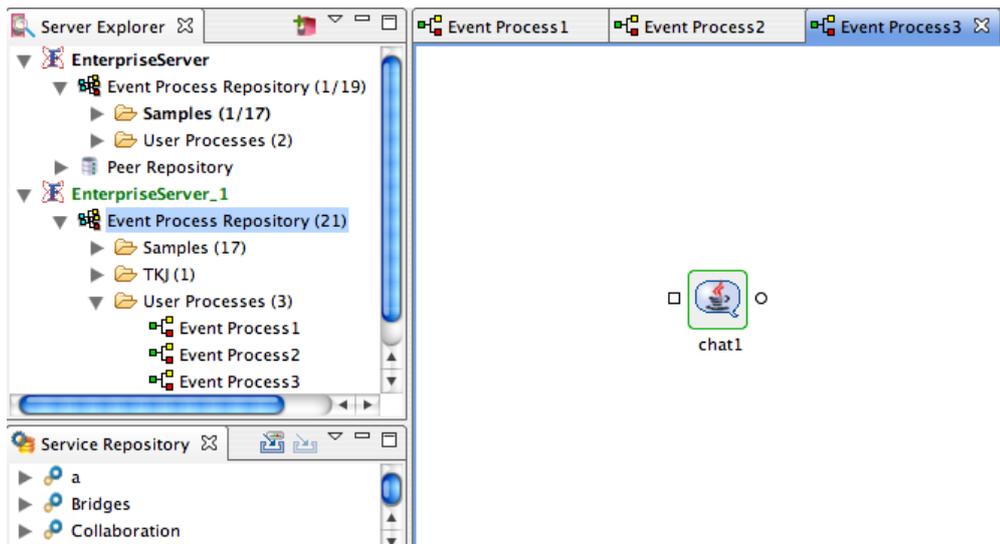


Figure 11.2.1: Active Enterprise Server

2. Now try to perform any action (say Open Event Process) on the inactive server (i.e. **Enterprise Server**). A confirmation dialog box is shown saying that the action requires the active server switch.

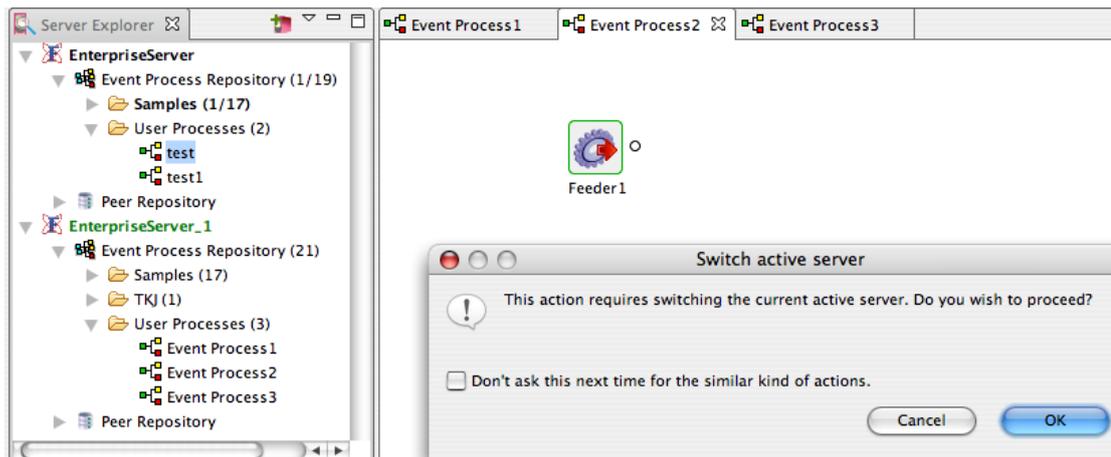


Figure 11.2.2: Switch Active Server

- When the user clicks the Ok button, the editors related to **Enterprise Server_1** will be closed and the editors corresponding to **Enterprise Server** will be opened. Also the service palette and service repository show the services present in the **Enterprise Server**. The Active server can be identified by the color green.

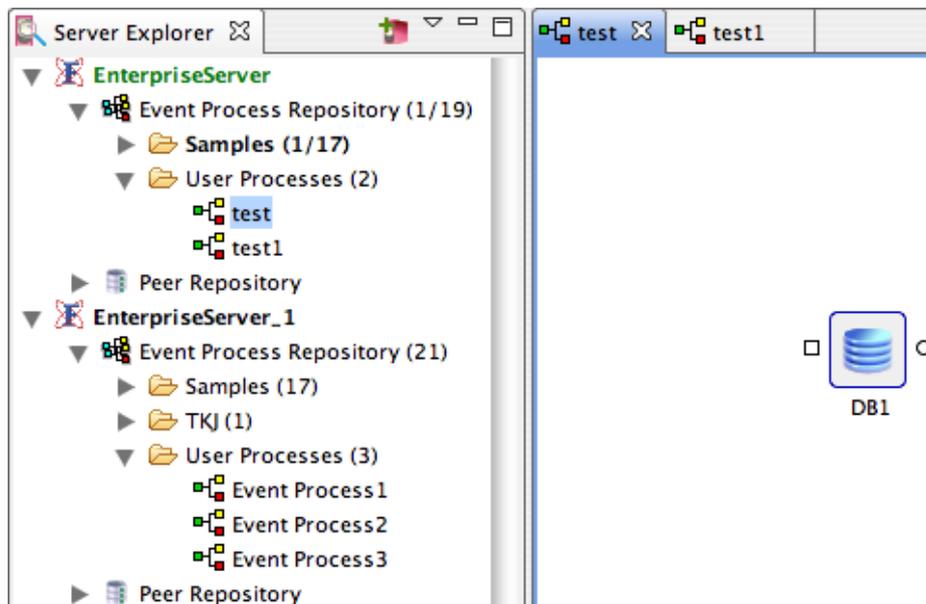


Figure 11.2.3: Event Processes present in Enterprise Server

- To switch back to **Enterprise Server_1**, perform any action on the **Enterprise Server_1** node or right-click and select the **Activate** option. All the editors which are opened previously are restored and the editors corresponding to previously active server are closed.

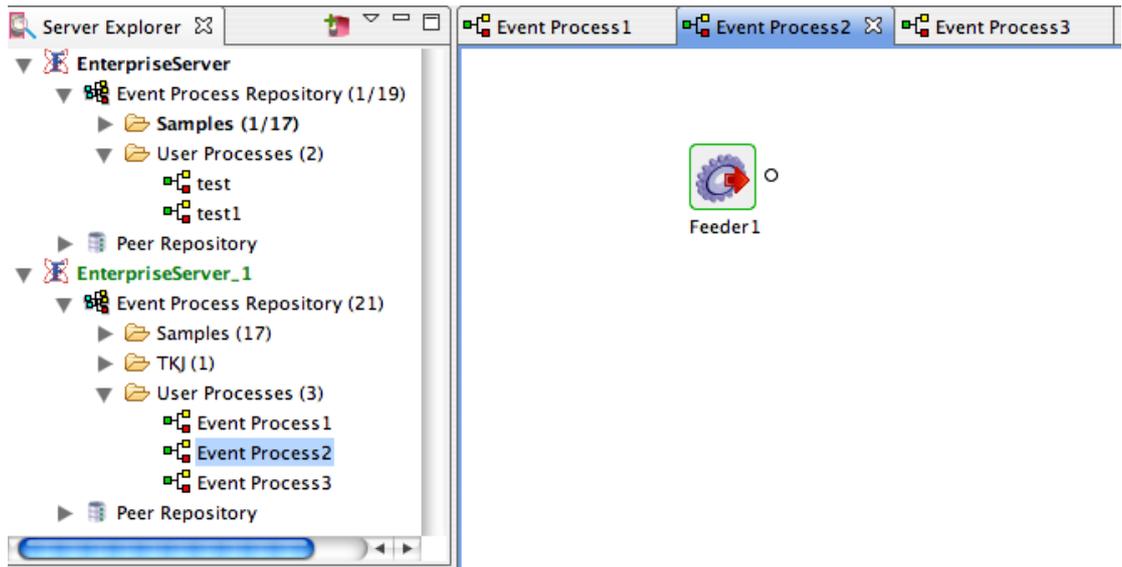


Figure 11.2.4: Services restored in Enterprise Server_1

During the active server switch, if the user tries to switch from a server containing any unsaved editors, a dialog box containing the unsaved editors will be prompted where the user can select the editors to be saved.

Note: The User can select appropriate option to save or discard changes in editors but there is no option to veto the switch.

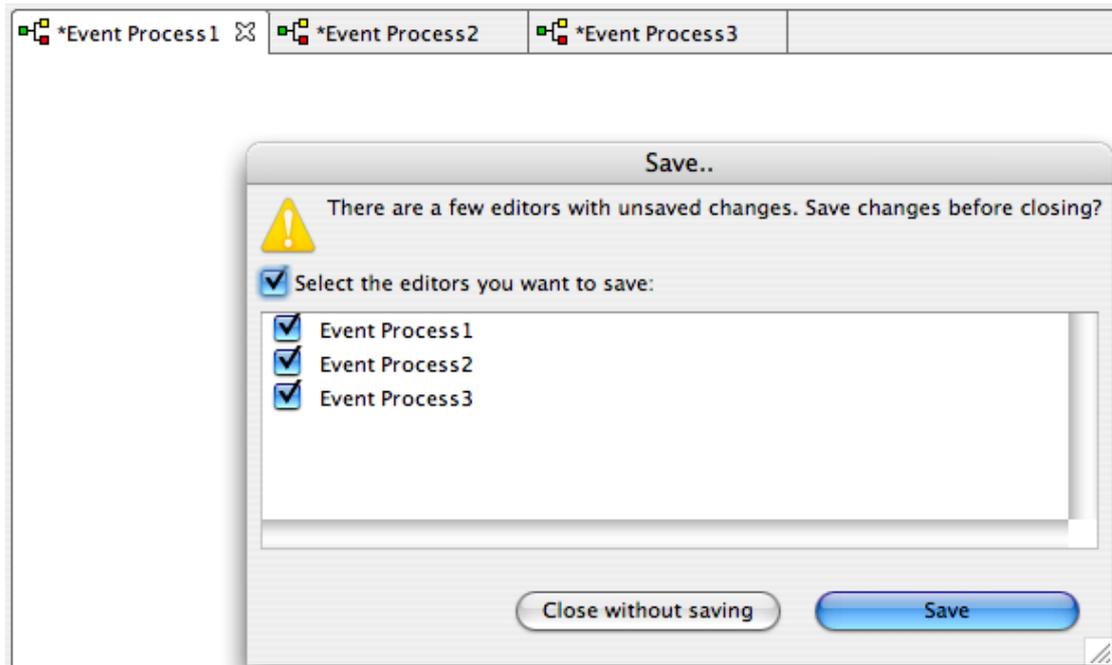


Figure 11.2.5: Unsaved editors dialog

11.3 Switching Between Perspectives

eStudio has three perspectives; Offline Event Process development, Online Event Process development and Mapper

To change the perspective, perform the following steps:

1. Click the **Open Perspective** option from Window -> Open Perspective -> Other or from the shortcut bar on the left-hand side of the Workbench window.

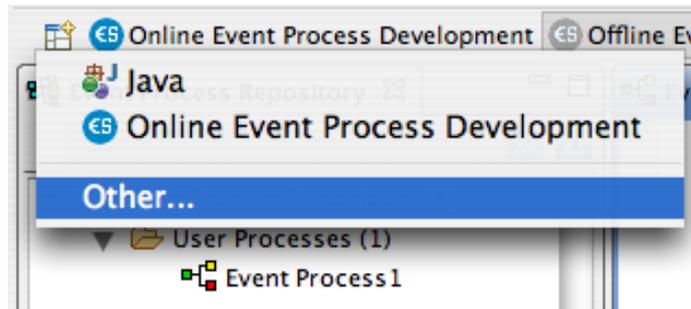


Figure 11.3.1: Selecting the Other.. option from perspective button

2. Select the **Online Event Process Development** to open the online perspective.

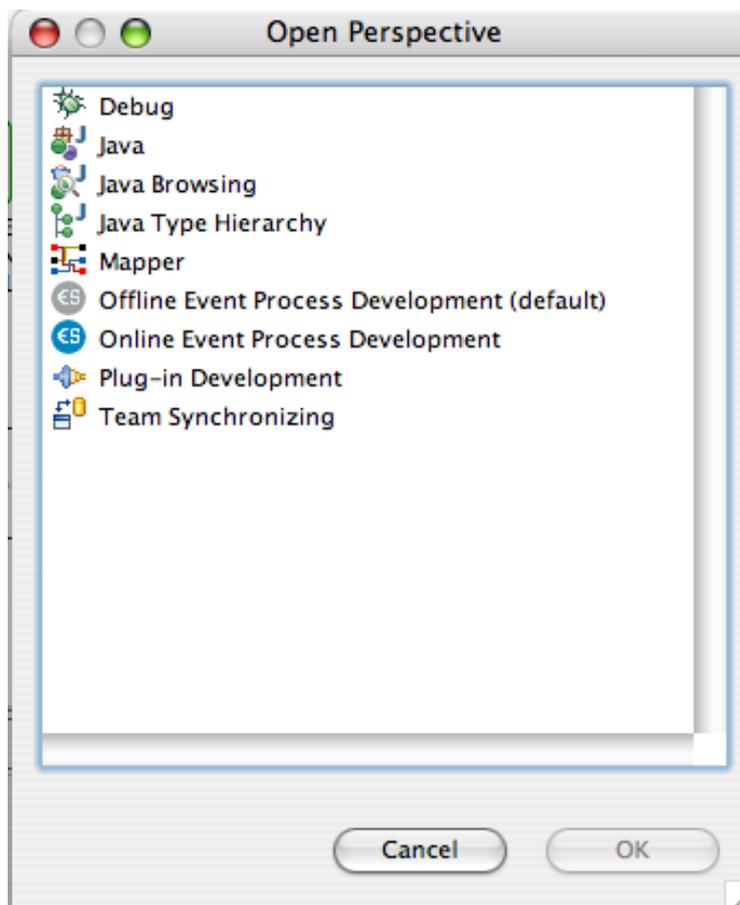


Figure 11.3.2: Selecting Online Application Development perspective

The Online perspective shows all the views and editors customized for the online application development as shown in the figure. During online application development, application development takes place after logging in to the server.

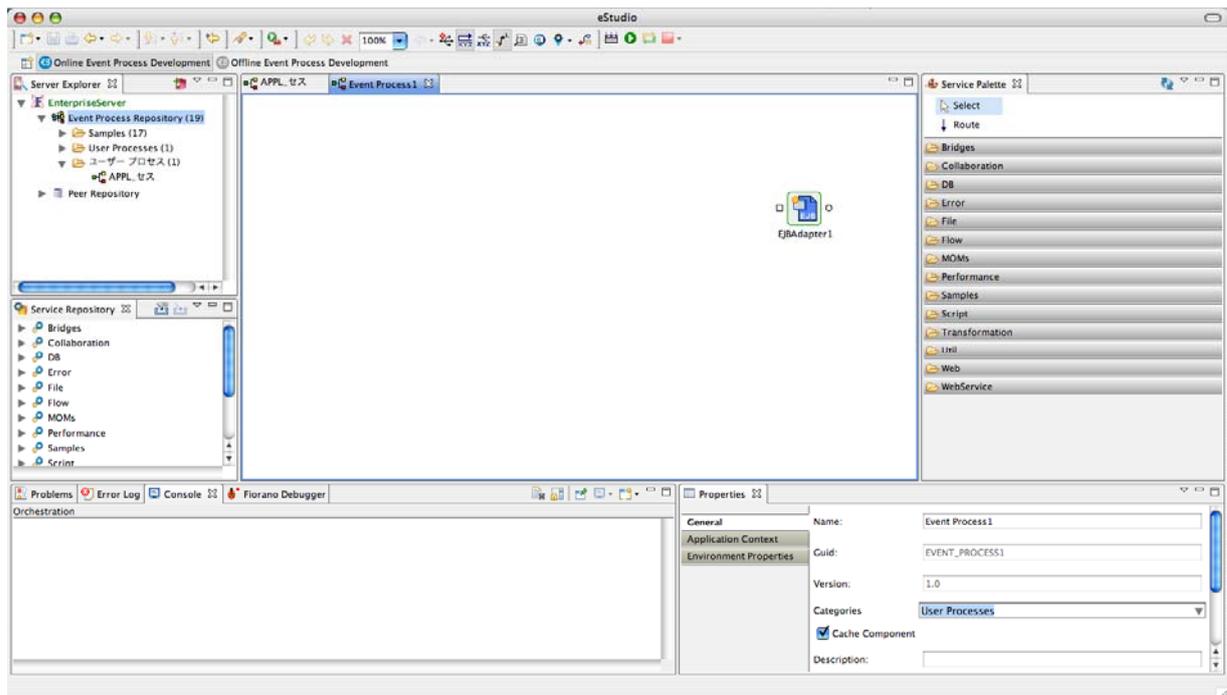


Figure 11.3.3: Online Application Development Perspective

Now, switch to Offline perspective by clicking on the Open Perspective button on the shortcut bar on the right-hand side of the Workbench window, select Other... from the drop-down menu. Select the Offline Event Process Development to open the Offline perspective. (User can also switch to different perspectives like java etc.)

The Offline perspective shows all the views and editors required for the offline application development as shown in the figure 11.3.4. During offline application development, there will not be any interaction with the server.

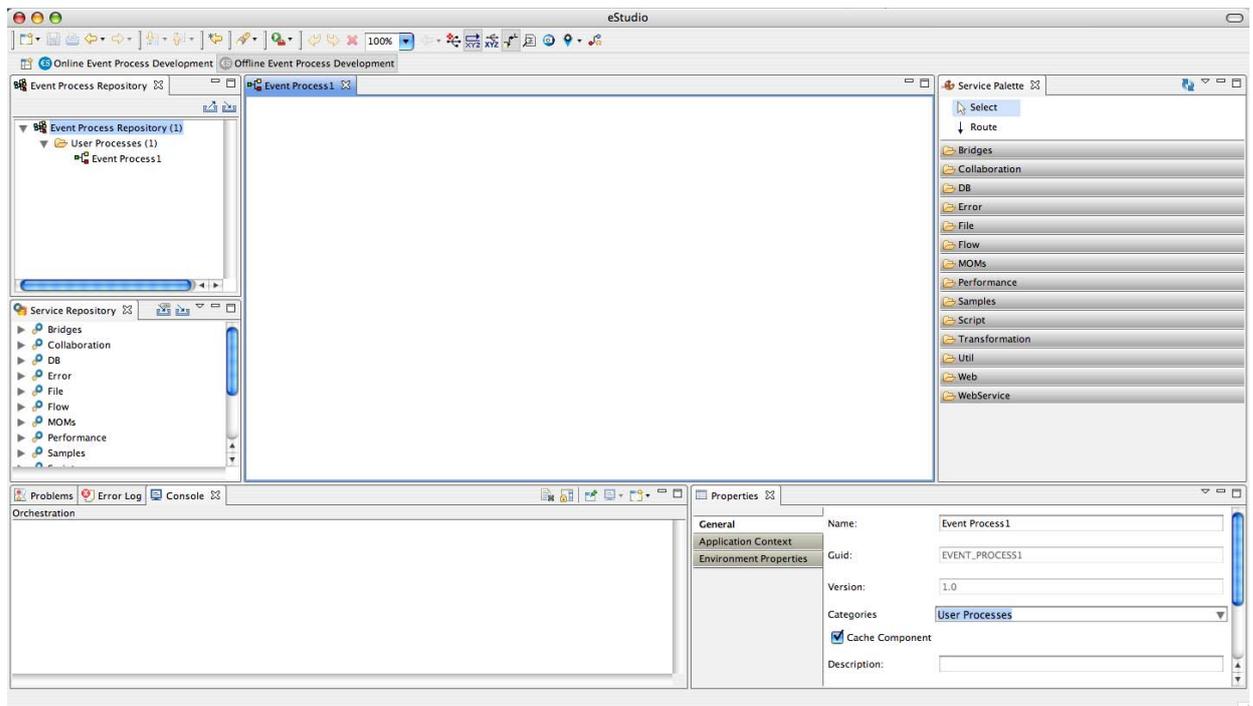


Figure 11.3.4: Offline Application Development Perspective

Chapter 12: Fiorano Preferences

Fiorano Preferences are available under Window > Preferences -> Fiorano. Various sections in Fiorano Preferences are explained in the following sections.

12.1 ESB Connection Preferences

Enterprise Server configurations can be defined here. List of Enterprise Servers can be added and the server details such as IP address, port and security credentials can be provided. These servers configuration is used in Offline Event Process Development perspective for actions (export Event Process to server, import Event Process from server etc.) that require a Server connection.

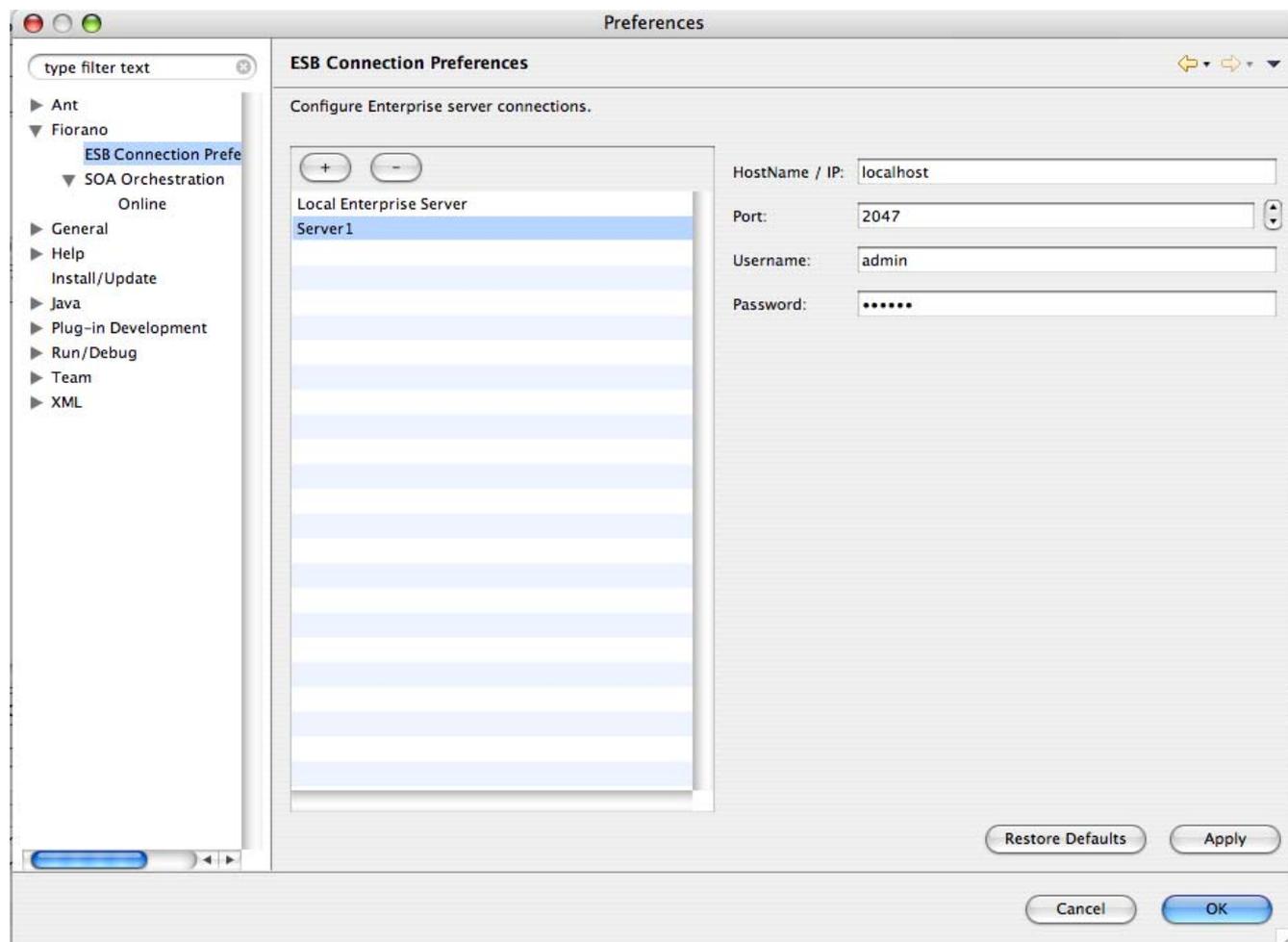


Figure 12.1.1: Enterprise Server Configurations

Restore Defaults button is used to restore the preferences to default values.

12.2 SOA Orchestration

SOA orchestration preferences are grouped into General options, Workflow options, Service options and CPS options.

12.2.1 General Options

General Options contains preferences for Error Port and Routes Color and Route Shape. The preference chosen here will be applied in orchestration editor.

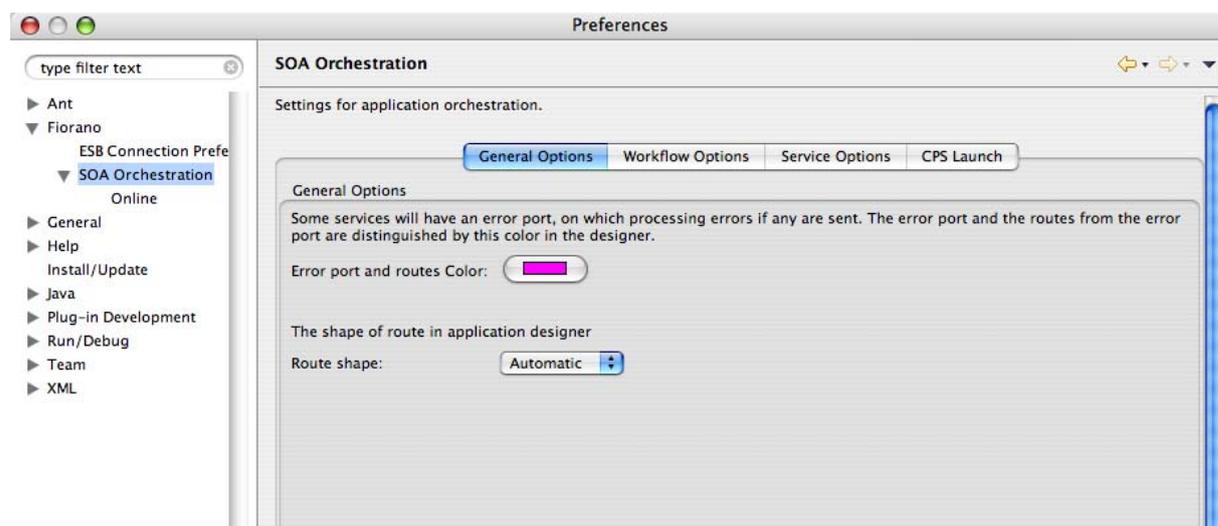


Figure 12.2.1: SOA Orchestration Preferences

12.2.2 Workflow Options

Workflow options contain Workflow color information. Workflow Item color and Workflow End color used in Document tracking can be configured here.

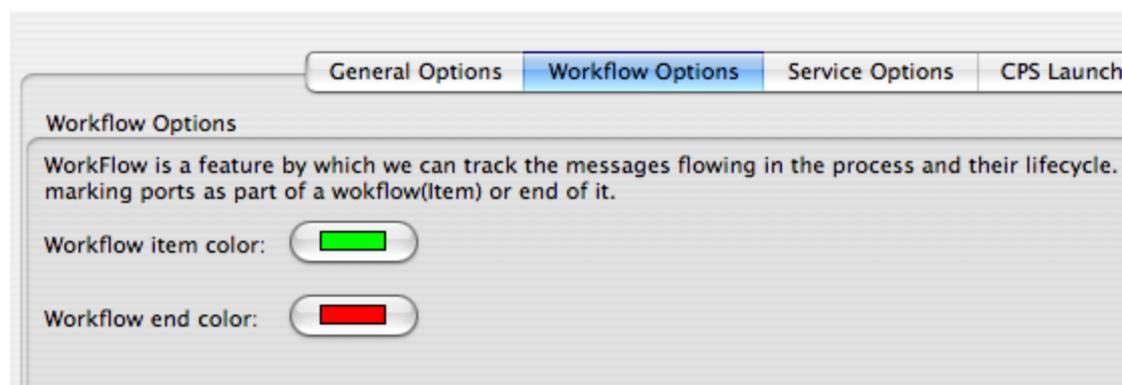


Figure 12.2.2: Workflow options

12.2.3 Service Options

Service instance default configurations can be provided here. These default configurations are set on a service instance when a new service instance is created.

12.2.3.1 Default JVM Configurations

JVM configurations like classpath, System properties, memory options etc. can be defined. These options are used while launching the component in Separate Process launch mode.

These are the default configurations that are applicable to all the newly created service instances. Service Instances can also overwrite the default configurations set on them by making modifications in properties view.

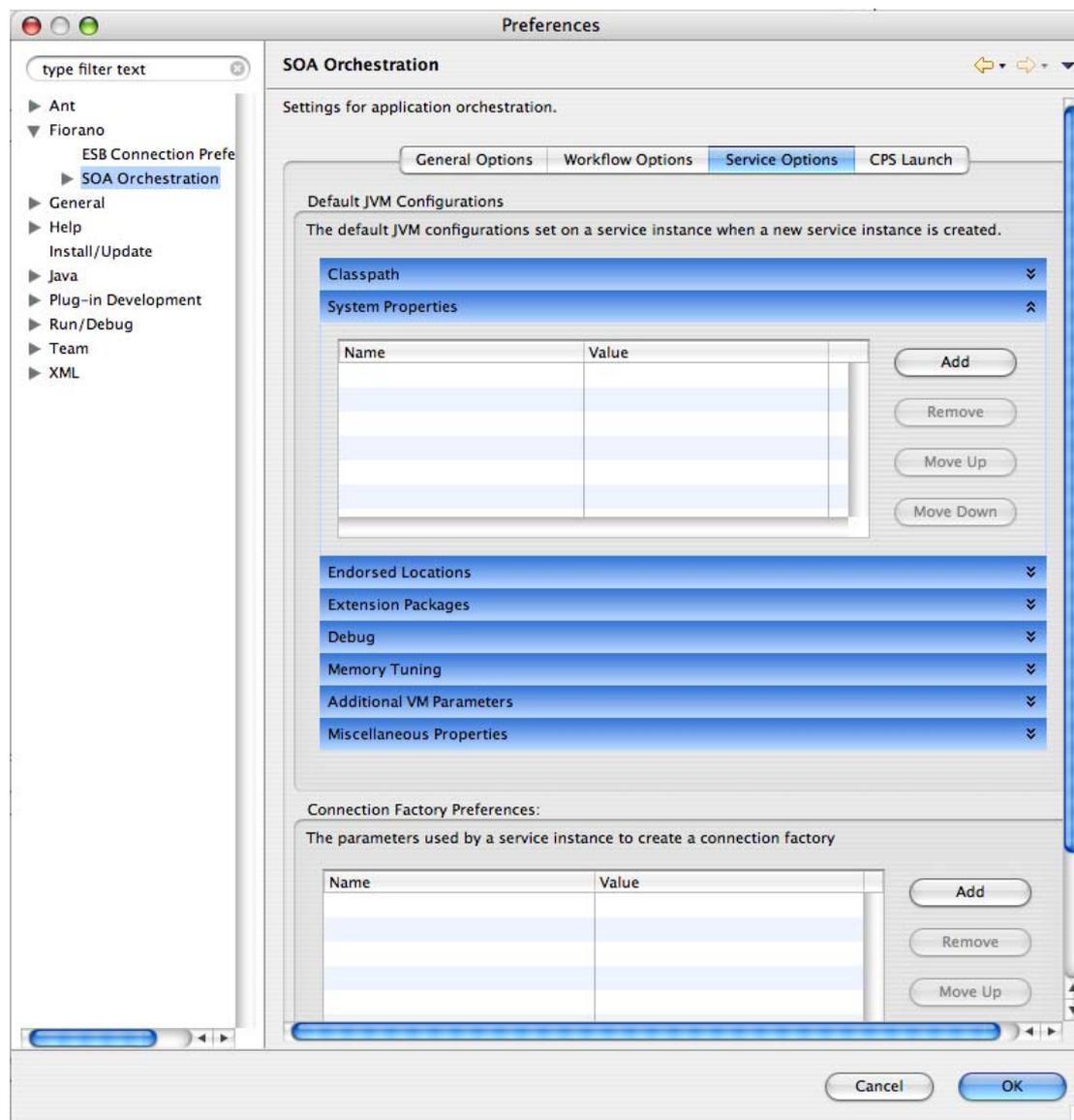


Figure 12.2.3: Service options

Configurations defined here are set on the Service Instance in Runtime Arguments section of the properties view. For example if the user wants to change the heap memory settings, he can provide the values for memory tuning properties as shown in Figure 12.2.4

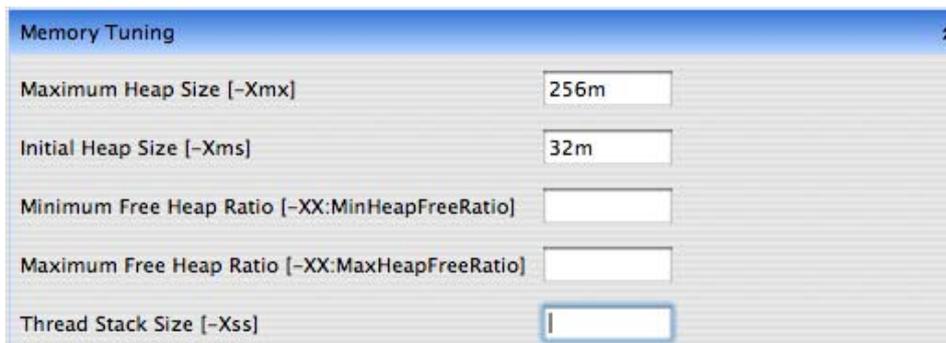


Figure 12.2.4: Memory tuning options

After defining these configurations, the default values are set when a service instance is drag-and-dropped in Orchestration editor and can be seen in Runtime Arguments section as shown in Figure 12.2.5.

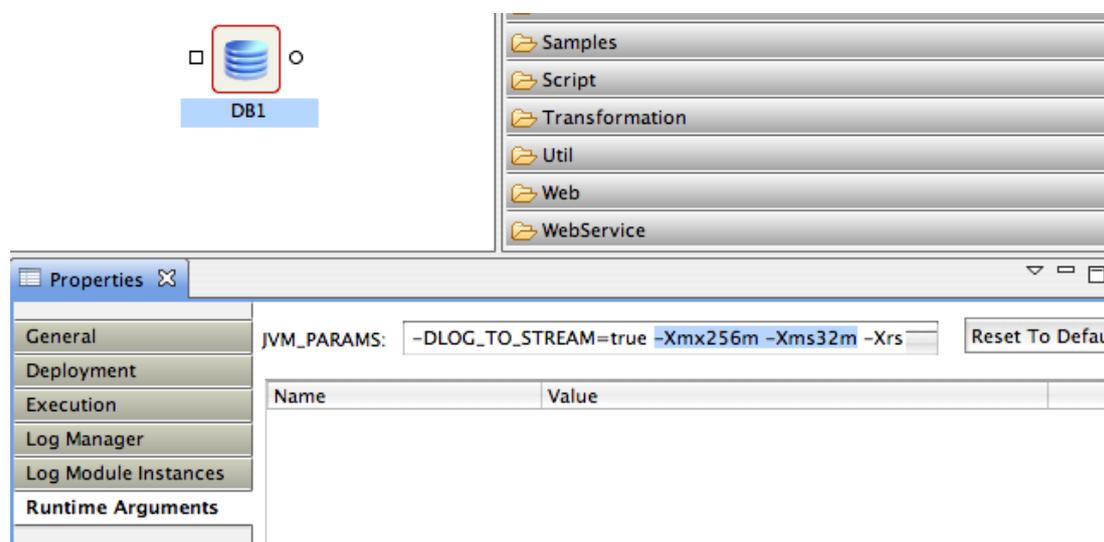


Figure 12.2.5: Runtime Arguments

These properties are set on the JVM on which the service instance will be launched.

12.2.3.2 Connection Factory Preferences

Configurations used by service instances while creating Connection factories can be defined here. The connection factories are created internally by using default configuration. To overwrite the defaults, user can set the properties here.

The properties defined here are available in Execution section in service instances properties view.

12.2.4 CPS Options

These options are used by external CPS launch components where the CPS is launched as a separate JVM process. The following components CPS is launched in separate process JVM: SapR3, XMLSplitter, SapR3Monitor, Aggregator, CBR, Join, CompositeBC, JMSIn: 5.0, JMSOut: 5.0 and JMSRequestor: 5.0.

Apart from these prebuilt components, custom components CPS will also be launched in a separate process JVM.

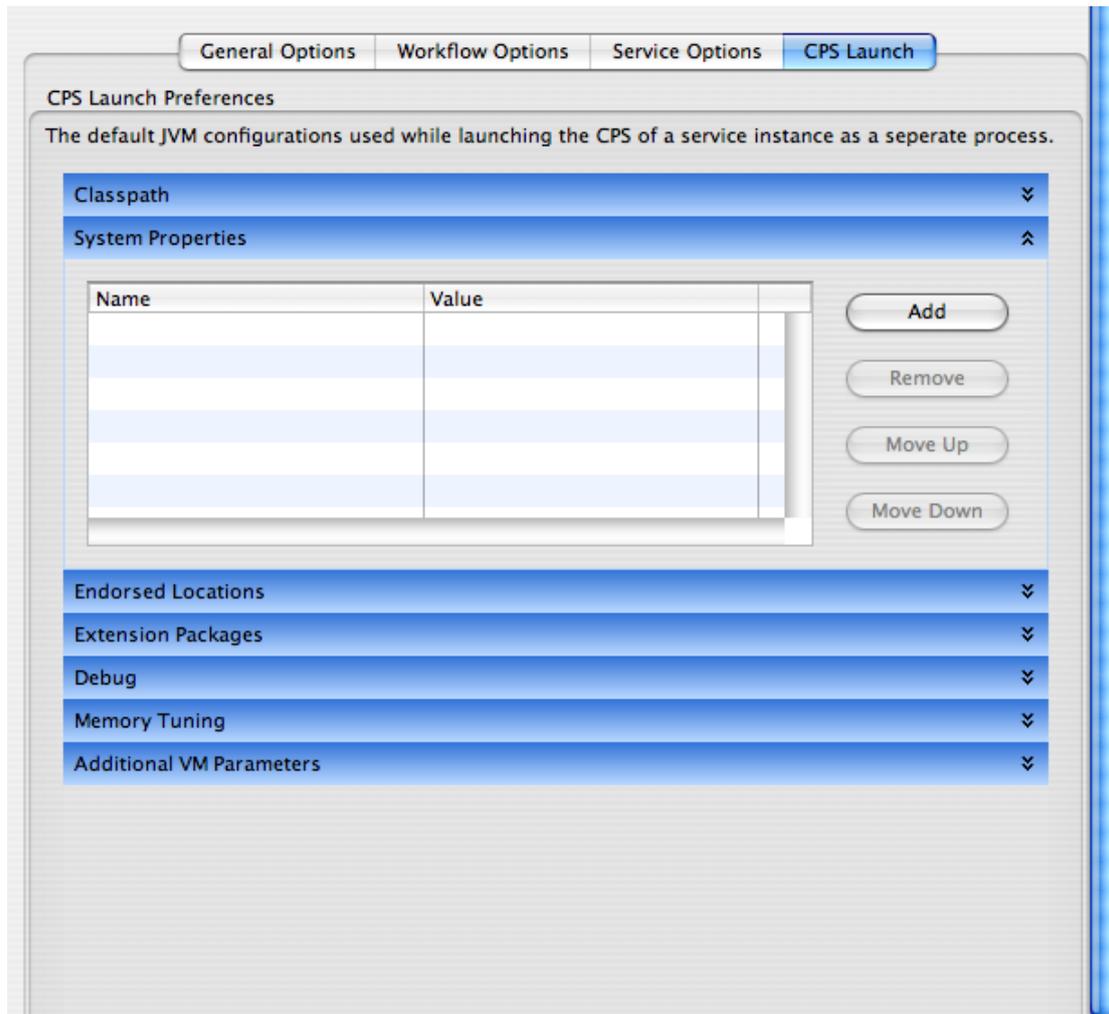


Figure 12.2.6: CPS launch options

12.3 SOA Orchestration Online

This section contains configurations for online Event Process orchestration.

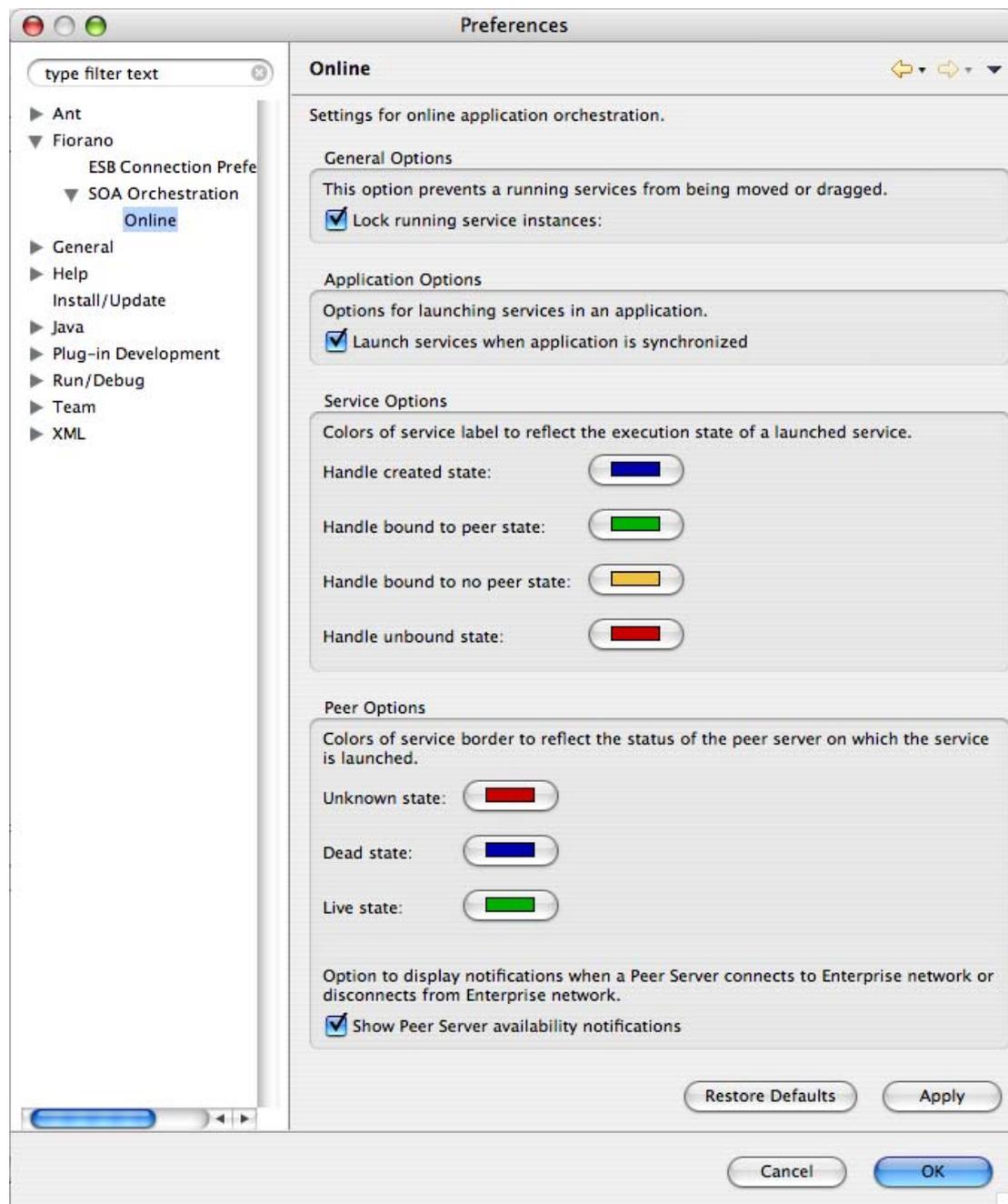


Figure 12.3.1: Online Orchestration preferences

12.3.1 General Options

Lock running service instances: This option prevents the service instances from being moved or dragged when an Event Process is running.

12.3.2 Application Options

Launch Services when application is synchronized: If this option is enabled, in a running Event Process, service instances in stopped state will be started if the user clicks the Synchronize button in an Event process.

12.3.3 Service Options

The color of the Service Instance label name at different execution status can be configured from here, that is, when a Service Instance is running, stopped and so on.

By default when a Service Instance is dragged and dropped the instance name color is Black. The states and corresponding Service instance label name colors are explained below.

- **Handle Created State:** This color is shown when the service instance handle is created. This happens before the component is launched completely.
- **Handle Bound to Peer state:** This color is when the service instance is running.
- **Handle Bound to no peer state:** This color is shown when the peer server on which the component is running is stopped.
- **Handle unbound state:** This color is shown when the component in a running Event Process is stopped.

12.3.4 Peer Options

These are the colors applied to service instance border to reflect the status of the peer server on which the service instance is configured to launch.

- **Unknown State:** The peer server configured is unknown. i.e. the peer server configured is not running and is not present in peer repository node under Enterprise Server node.
- **Dead State:** The peer server configured is not running but it is present in peer repository node under Enterprise Server node.
- **Live State:** The peer server configured is present in Peer repository and is running.
- **Show Peer Server availability notifications:** Whenever a peer server connects to the Enterprise Server or disconnects from Enterprise network, a notification dialog will be shown as shown in Figure 12.3.2.

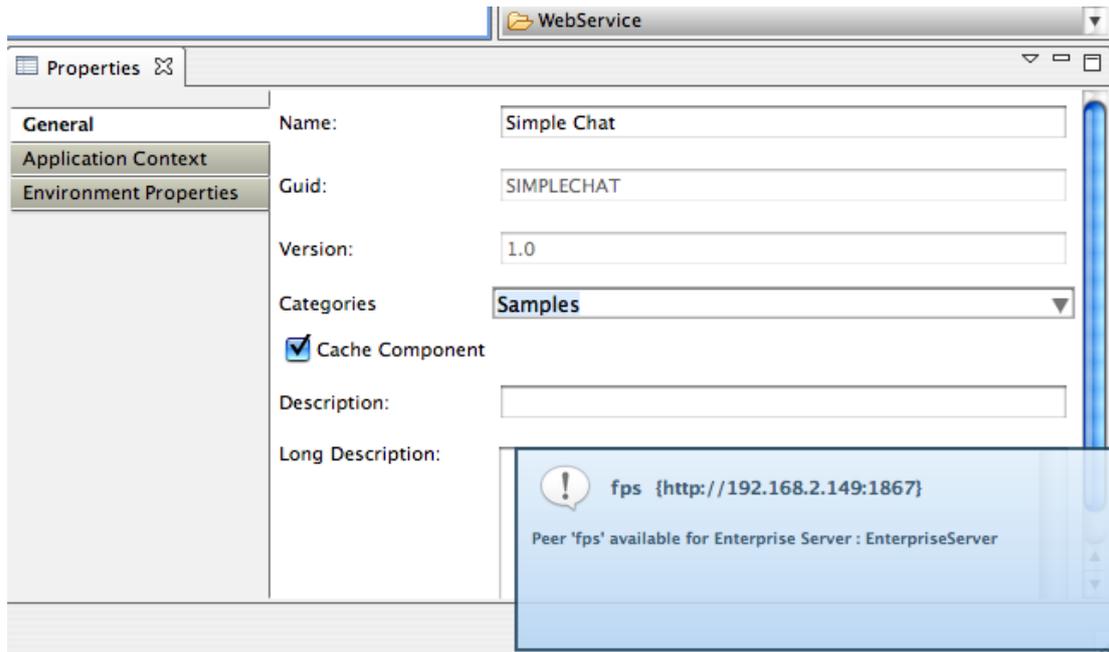


Figure 12.3.2: Peer server notification

This option is to enable or disable the notifications.

12.4 Key Board Short Cut Preferences

Before using Key Board shortcuts Fiorano scheme has to be set in Preferences (Window -> Preferences -> General -> Keys).

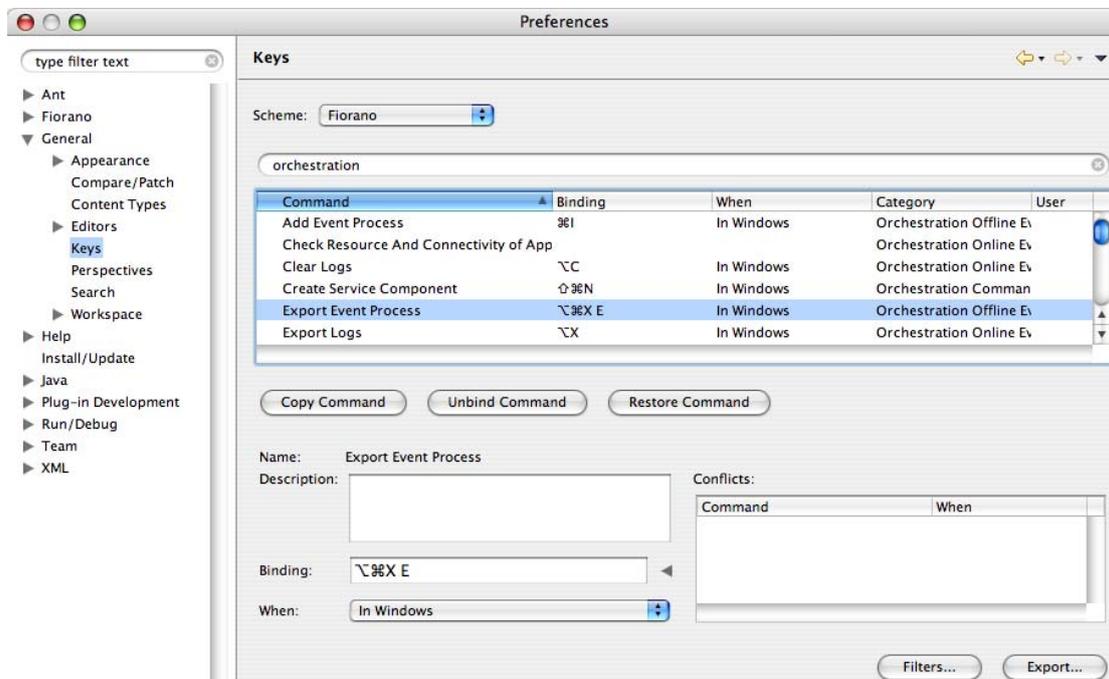


Figure 12.4.1: Key-binding preferences

The default Key Board shortcuts for various actions are listed below.

Help F1

Rename F2

Undo Ctrl + Z

Redo Ctrl + Y

Add Event Process CTRL + I

Open Event Process CTRL + O

Import Event Process CTRL + ALT + I E

Import Event Process (nStudio) CTRL + ALT + I N

Import Service (from Local Disk) CTRL + ALT + I L

Import Service (from Server) CTRL + ALT + I S

Export Event Process CTRL + ALT + X E

Insert

1. Service Instance CTRL + ALT + A S

2. Event Process CTRL + ALT + A E

3. Remote Service Instance CTRL + ALT + A R

CRC ALT + Shift + C

Run Application ALT + Shift + R

Synchronize ALT + Shift + S

Stop Application ALT + Shift + K

View

1.View Debugger CTRL + ALT + V D

2.View Properties CTRL + ALT + V P

3.Logs CTRL + ALT + V L

4.View Error ports CTRL + ALT + V E

5.View Route Names CTRL + ALT + V R

Clear Logs ALT + C

Export Logs ALT + X

Toggle Lock Mode ALT + Z

Schema repository CTRL+Shift+S

Create Service component CTRL+Shift+N

The option to edit keyboard shortcuts is also available under General -> Keys section in the preferences dialog. The list of Fiorano Orchestration commands can be viewed by entering Orchestration in the filter box provided above the available keys. The shortcut for any of the action/command can be changed by editing the Binding text field available below the keys table section.

Chapter 13: 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.

1. Navigate to Tools -> Schema Repository. This opens a Schema Repository editor as shown in Figure 13.1.1 using which schemas can be added to schema repository.

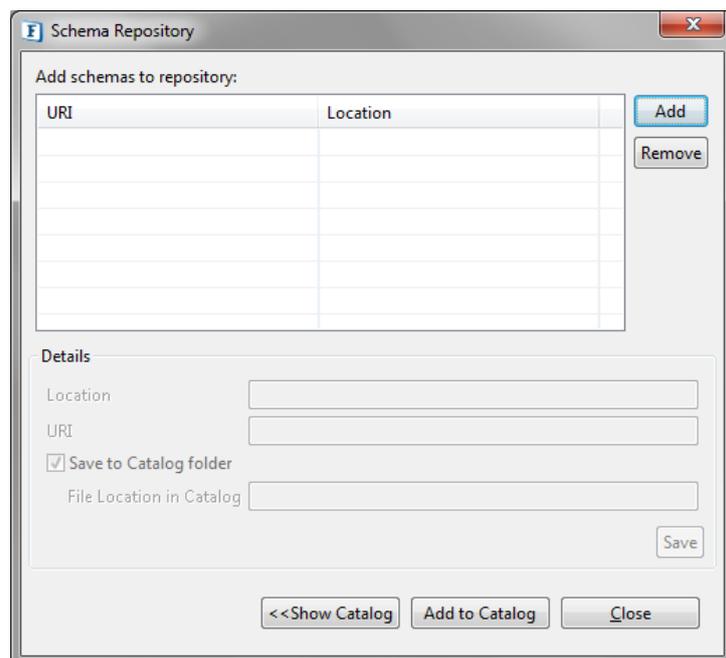


Figure 13.1.1: Schema Repository

2. To add schema files to the catalog, click **Add** button and select the **Add File(s)** option as shown in the figure 13.1.2. Folders containing multiple schemas can also be added to the catalog using **Add Folder** option. Select the suitable option and choose the required file(s) or folder from the browser window opened.

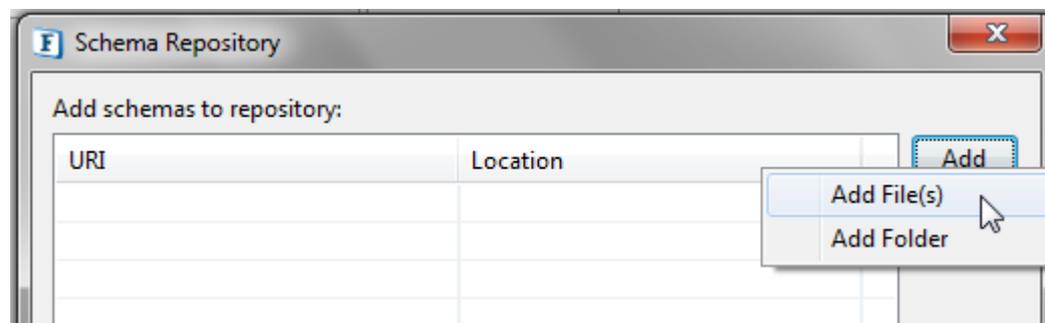


Figure 13.1.2: Adding XML files to Catalog

3. A confirmation dialog will be opened asking if the files need to be saved to the Catalog folder. If answered with **Yes**, the schema files will be saved in **<FIORANO_HOME>/xml-catalog/user** folder upon clicking the **Add to Catalog** button. Otherwise the absolute path of the files' original location will be used to refer the schema.
4. The Namespace **URI** and **Location** of all the files chosen will be populated in the table as shown in the figure 13.1.3. Selecting a particular entry will show its details in the **Details** panel.
5. Any changes to the **URI** or **File Location in Catalog** can be modified from the **Details** panel and the changes can be saved using the **Save** button.

Note: If an entry has empty **URI** string or if two or more entries have the same **URI**, such entries will be shown in red and an error message describing the problem with the entry will be shown in its **Details**.

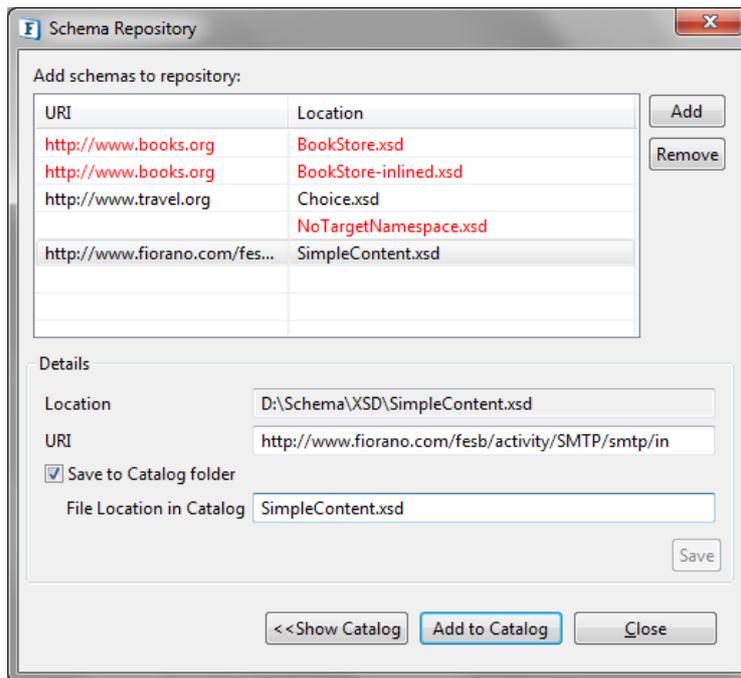


Figure 13.1.3: Adding schema to Schema Repository

6. Make sure all the entries are added without any errors and click **Add to Catalog** button to add the schemas to Schema Repository. The schema files for which the user chooses the **Save to Catalog Folder** option will be added to **<FIORANO_HOME>/xml-catalog/user** directory and directories added will be copied along with all the sub-folders and files.
7. If the **Save to Catalog Folder** option is not chosen, the files will be referred from their original locations.
8. Click **Show Catalog** button to view the schema files added to the catalog.
9. A new row specifying the URI and Location of each XSD will be added to the table displayed at the bottom of the dialog as shown in the figure below.

Note: 1. Schema files which are not saved in the catalog folder will be referred by their absolute paths. Refer Entry 3 in the figure below.

2. If a folder is added, relative paths will be used to refer them in the catalog. Refer Entries 4 and 5

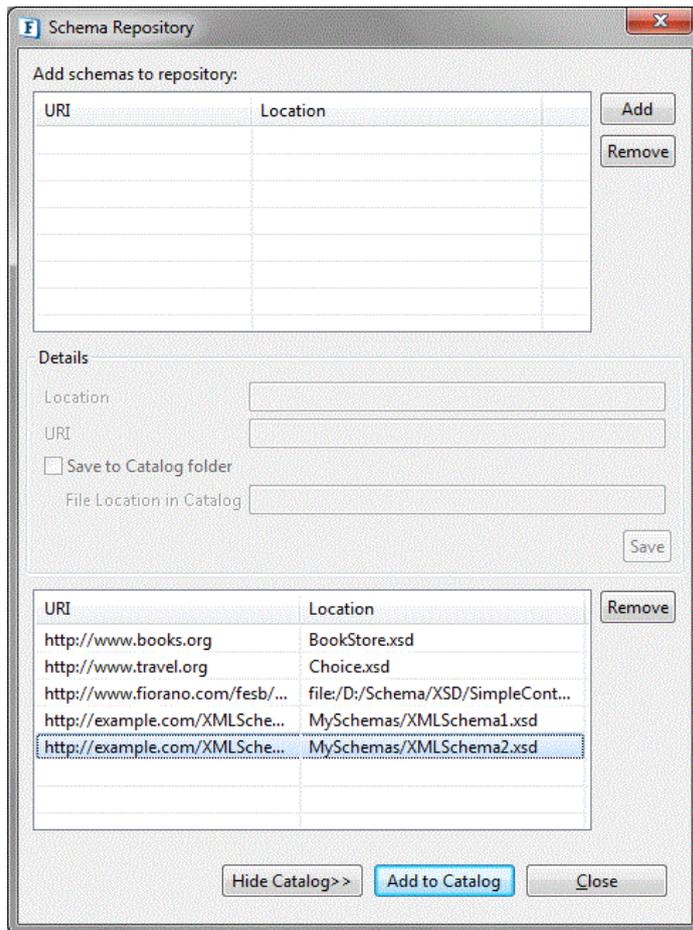


Figure 13.1.4: Table displaying the schema files added to catalog.

- To remove a schema from the schema repository, select the row from the table and click Remove.



Figure 13.1.5: Deleting schema from 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.

Chapter 14: SCM Integration

The Event Processes developed in Fiorano environment can be shared directly into from the version control system in eStudio. The steps below explain the processes to be followed to configure and use the version control support in eStudio. For demonstration purpose steps for SVN integration are mentioned.

14.1 Downloading and integrating SVN into eStudio

To integrate SVN, subclipse has to be downloaded from the eclipse update site.

1. In eStudio, navigate to **Help → Install New Software**. Paste the URL http://subclipse.tigris.org/update_1.6.x in the text field beside **Work With** property and click the **Add** button.
2. A dialog box is displayed asking to provide a **Name**. Provide a name and click **OK**.
3. All subclipse related libraries are shown. Select all libraries except the **Subclipse Integration for Mylyn 3.x** option, as shown in Figure 1. Deselect the option **Contact all update sites during install to find required software** (this speeds up the installation process) and click **Next**.

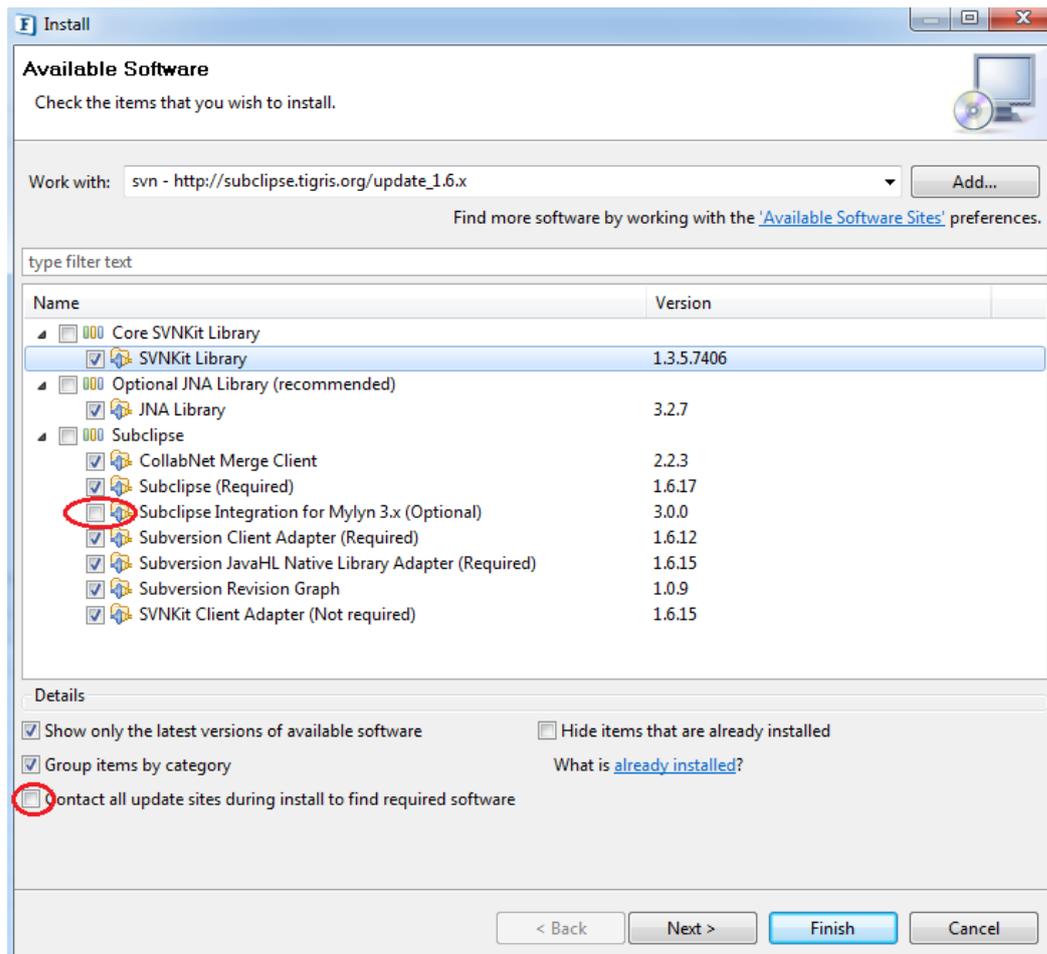


Figure 1: Install the New Software wizard

4. In the next page the items selected in step c are listed. Click **Next**.
5. In the Review Licenses page, accept the terms and click **Finish**.
6. Subclipse libraries will be installed. During installation a security warning will be displayed as shown in Figure 2. Click **OK**.



Figure 2: Security warning during installation

7. Once the installation is done, eStudio displays a prompts for a restart. Restart the eStudio and the version control plugins are integrated and ready to be used.

14.2 Using SVN with eStudio

Event Process can be committed to SVN either from the Event Process Repository view or from the project explorer view **Window->Show View->Other->General**. Closed projects cannot be shared in the version-control system. Event Process should remain open so as to be able to share then in SVN.

1. Right-click on an Event Process project and select **Team→Share Project** as shown in Figure 3.

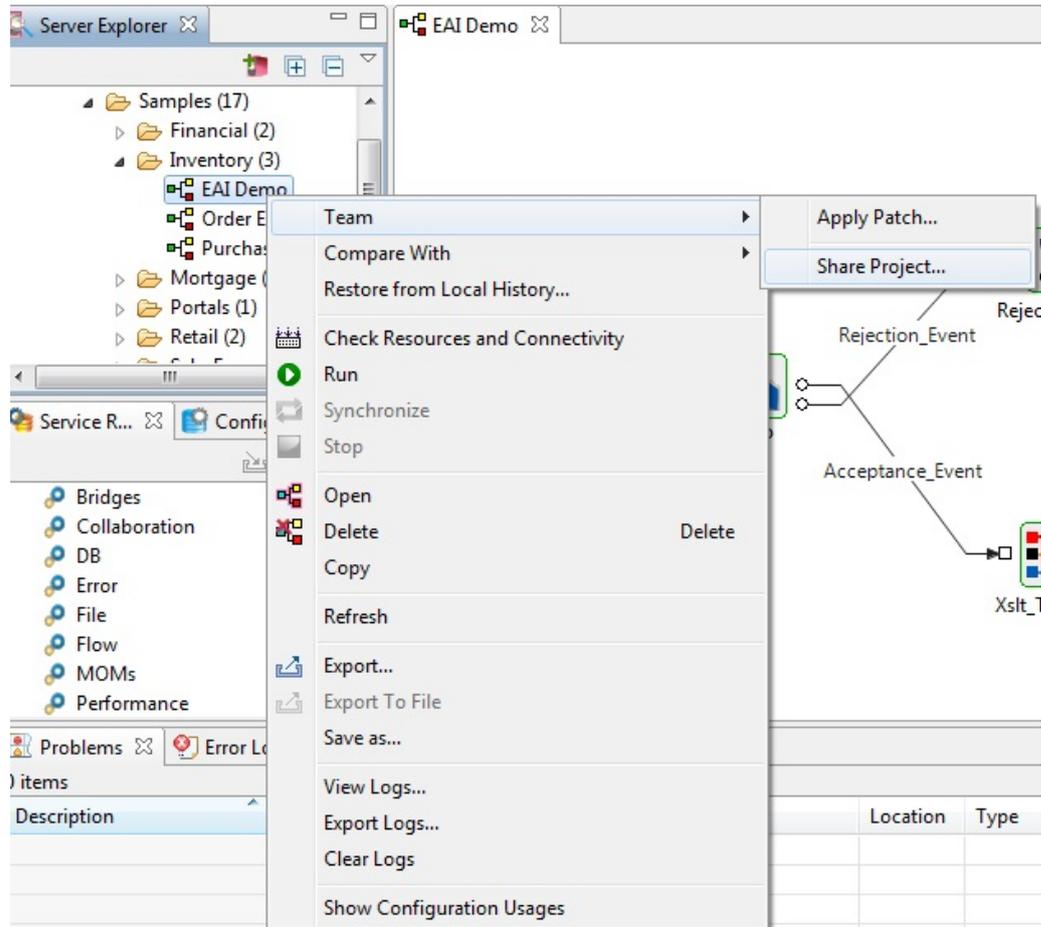


Figure 3: Sharing Event Process project

- Specify the SVN URL location where the project is to be stored as shown in Figure 4.

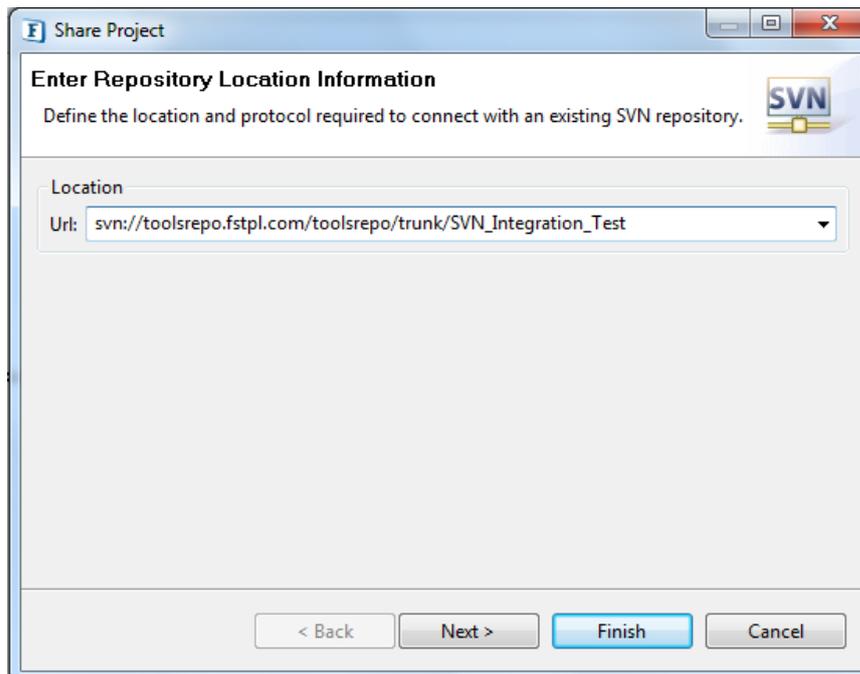


Figure 4: Providing svn repository URL

- Provide additional details and **Finish** the wizard. A dialog will open prompting for a perspective change.
- Click **Yes** as shown in Figure 5. Clicking **Yes** automatically transfers the User to the **Team Synchronizing** perspective.

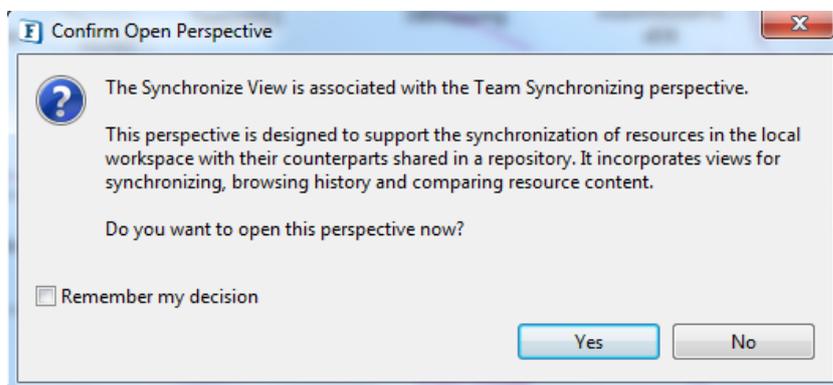


Figure 5: Dialog prompting for perspective change

- Right-click on the project in the Team Synchronizing view and select the Commit option to commit the project into the repository as shown in Figure 6.

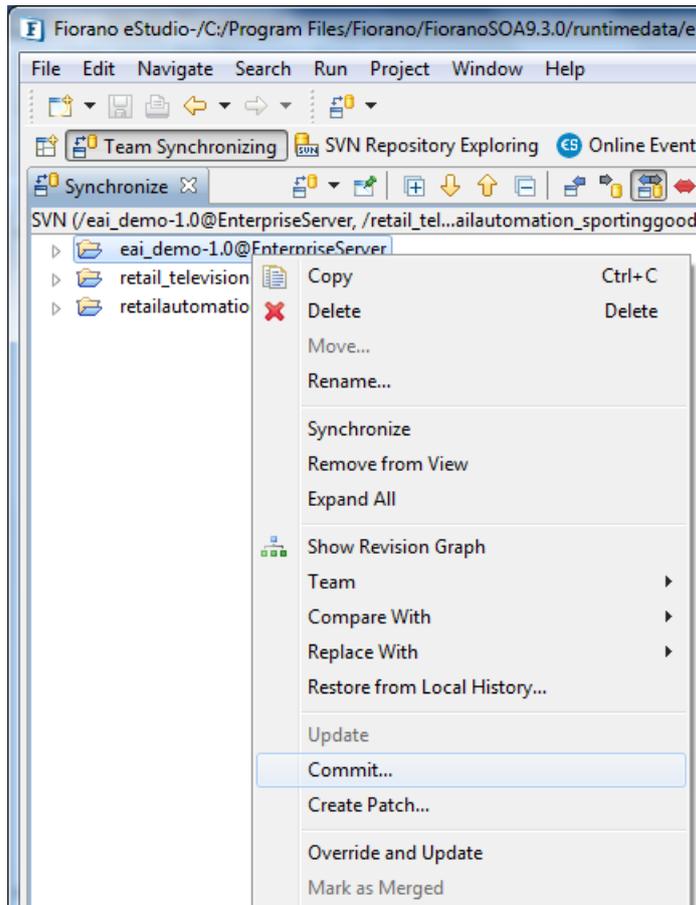


Figure 6: Committing project to SVN

Note: Perspective change mentioned in step (d) is not mandatory. Commit can also be done by right-clicking on the Event Process.

Once the Event process is shared in the repository, all SVN related options are available through the Event Process right-click menu, as shown in Figure 7.

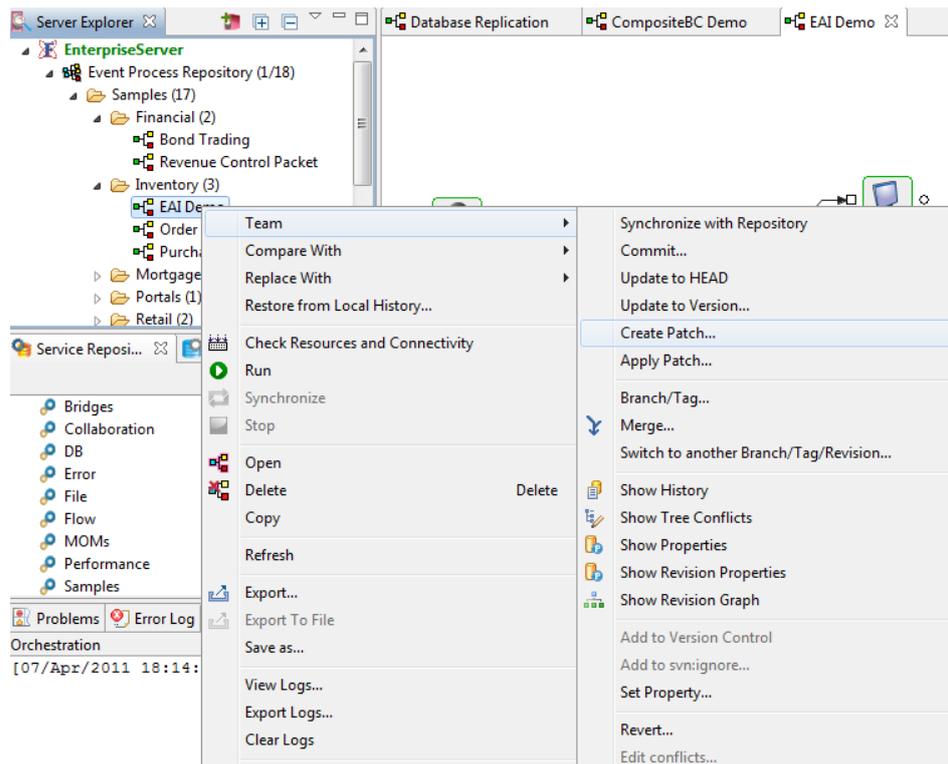


Figure 7: The SVN option in the Event Process context menu

14.3 Checking out and Updating Event Processes

To checkout Event Processes from the SVN repository,

1. Go to the SVN Repositories View **Window**→**Show View**→**Other**→**SVN**→ **SVN Repositories**.
2. In the SVN Repositories view, right-click on the Event Process to be checked and select **Checkout** option, as shown in Figure 8.

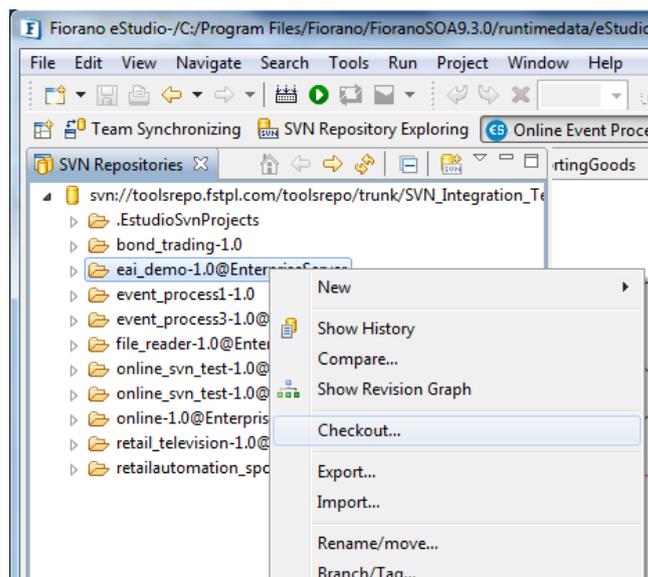


Figure 8: Checking a project from SVN

3. A dialog box opens asking for the project name, as shown in Figure 9. By default the Event Process name is shown with the Enterprise Server name appended. It is strongly advised that project names should not be changed.

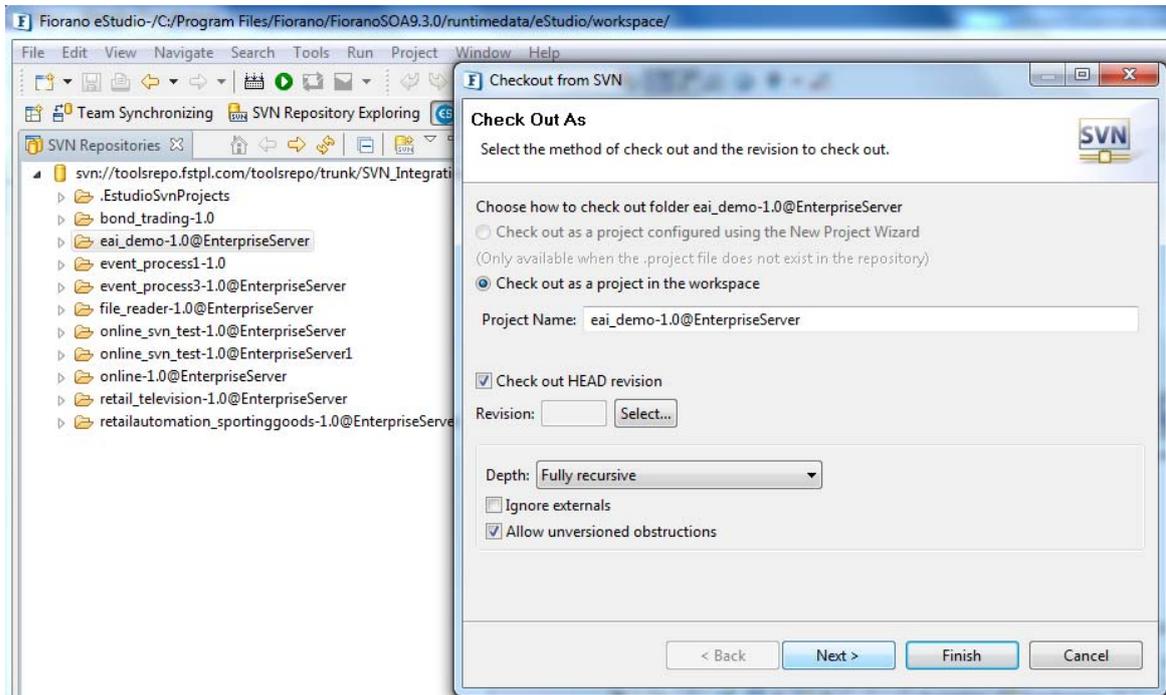


Figure 9: Checkout As dialog

Note: By default, the latest version from the repository will be checked out. To checkout an earlier version uncheck the option **Check out Head revision** and click the **Select** button to select the revision to be checked out.

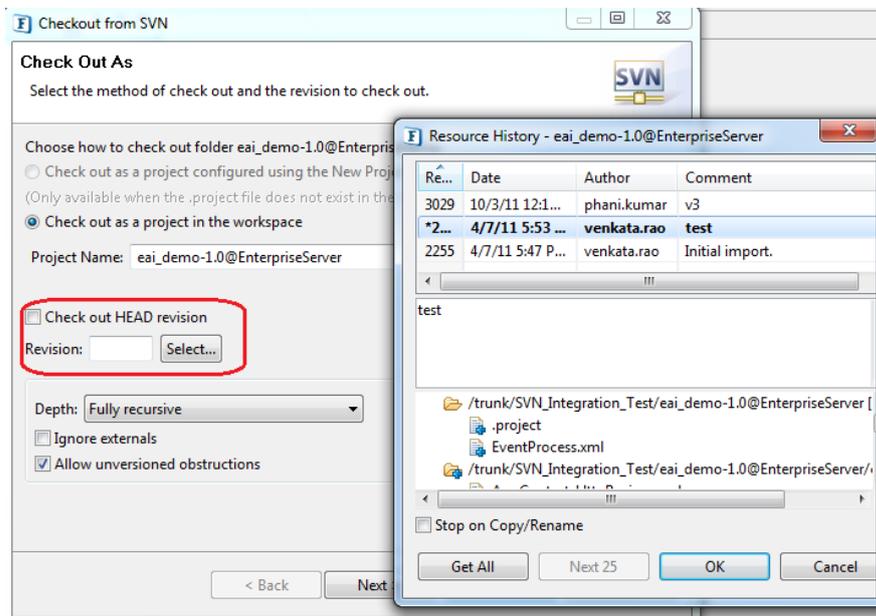


Figure 10: Checking out a selected version

4. Click **Next** button and the following page shows the workspace selection field. Use the default workspace and click **Finish** to checkout the Event Process.

The Event Processes will be checked out. Refresh the Event Process Repository node to see the newly checked out Event Process, as shown in Figure 11.

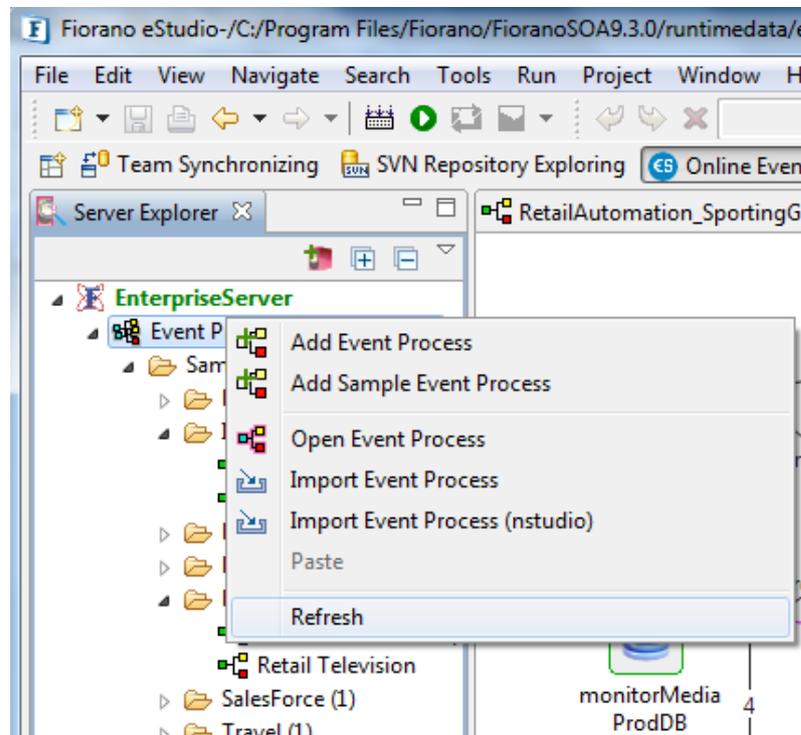


Figure 11: Refreshing Event Process node

Points to Note:

- During checkout, if the Event Process with the same name already exists in the workspace, it is always advised to delete the existing project from the Event Process Repository before checking out the project from SVN.
- Project names should not be changed during checkout unless it is required.

In Event Process project names, the name of the Enterprise Server node is used internally. The Event Processes checked in from one Enterprise Server node cannot be checked out in another Enterprise Server node if it does not have the same name.

For example if there are two Enterprise Server nodes - **EnterpriseServer** and **EnterpriseServer2**, if the user checks in an event process (say **Test**) from **EnterpriseServer** node into a version control system, then the project **Test@EnterpriseServer** is checked in into the repository.

If the user checks out the same Event Process to **EnterpriseServer2**, then the Event Process is not visible even though the checkout is successful. In eStudio Online mode, if a project name doesn't end with the current Enterprise Server node name then it is ignored. So in this particular scenario user has to rename the project name to **Test@EnterpriseServer2** during checkout.

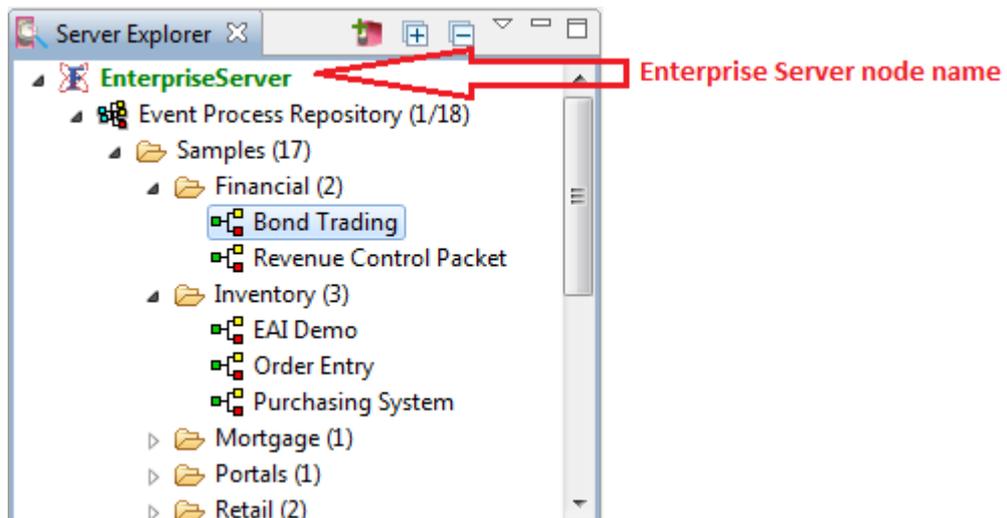


Figure 12: Enterprise Server node name

- To maintain svn information, the SVN related directories (**.svn**) are deployed into the Fiorano Enterprise Server along with the other Event Process data.
- **.svn** folders are ignored during Event Process export into the local disk or from Offline Perspective to Online Perspective.
- It's always a good practice to revert/commit the current changes in an Event Process whenever trying to update it with a copy from the repository to avoid any conflicts.

Chapter 15: Named Configurations

Named configurations are predefined configurations which are assigned a name and stored for later reuse. For example, if a particular routing logic (CBR component) or a transformation (XSLT component) is reused in multiple Event Processes, each such Service Instance has 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. With named configuration support, required configurations can be predefined and the name of the predefined configuration will be associated with all Service Instances. Since the actual configuration is stored in only one location and reused by multiple Service Instances, making changes to the named configuration will affect all Service Instances without the need to reconfigure them individually.

A view is provided in eStudio for managing Configurations. This view has consists of the types of configurations sorted in alphabetical order, as shown in Figure 1.

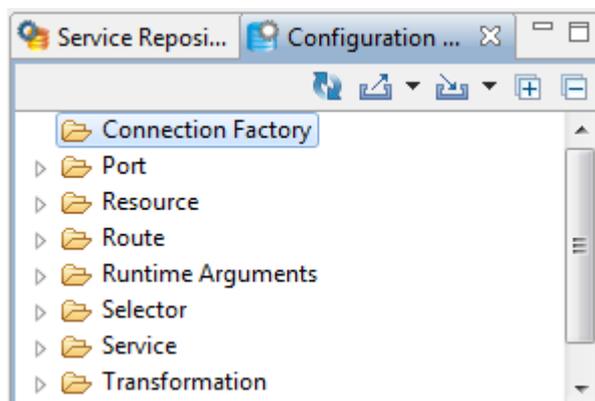


Figure 1: Configuration Repository View

Configurations can be of the following types:

- Connection Factory
- Port
- Resource
- Runtime Arguments
- Selector
- Service
- Transformation
- Workflow

15.1 Connection Factory

To add a Connection Factory configuration, right click on **Connection Factory** and select **Add Configuration** as shown in Figure 2.

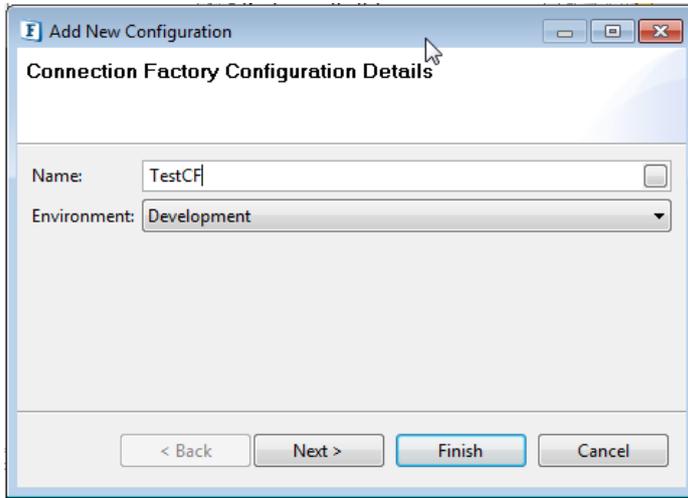


Figure 2: Connection Factory configuration details

Provide a name for the configuration in the details page. Click on **Next** to add Connection factory configurations. Add the desired properties using the **Add** button provided in the configuration page and click **Finish** to save the configuration, as shown in Figure 3.

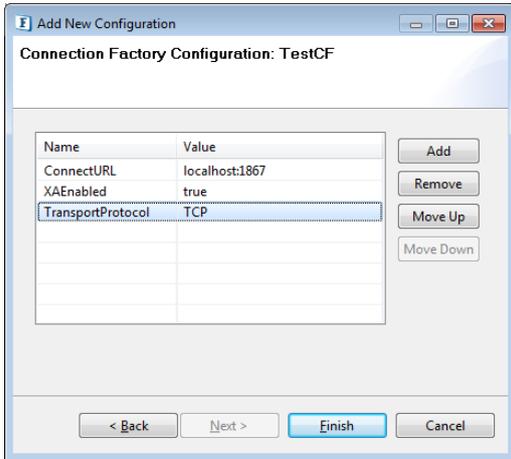


Figure 3: Connection Factory Configuration

Once the configuration is saved in the repository, it can be used to provide **Execution Details** of a Service Instance. To use a named configuration for the Connection Factory properties, select a Service Instance and go to the **Execution** tab in the properties view. The property **Configuration** lists the names of all available Connection Factory configurations. Select the configuration to be used and the corresponding Connection Factory properties will be fetched from the named configuration selected.

These properties can be viewed in the Connection Factory Properties table as shown in Figure 4.

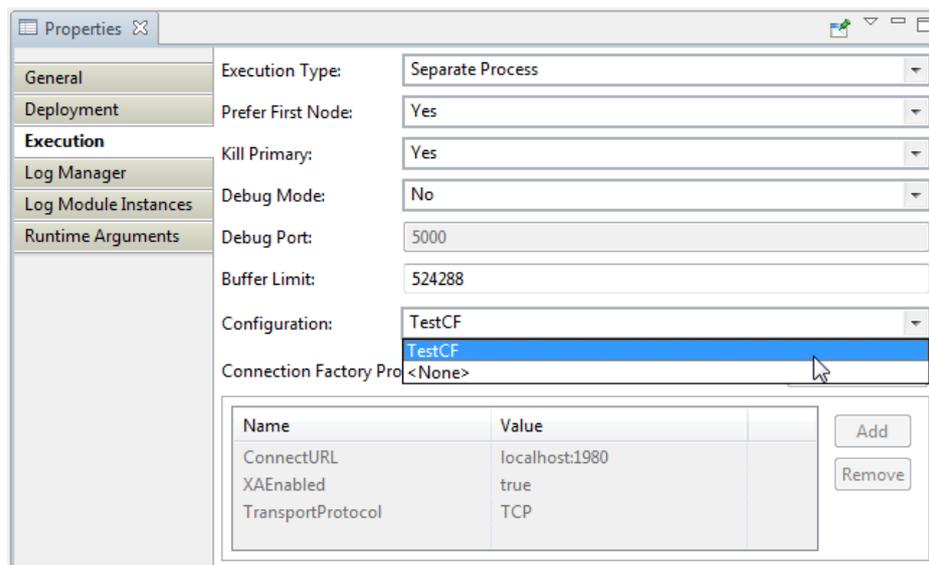


Figure 4: Service Instance Execution properties

Note: When the Connection Factory named configuration is used, the properties in the named configuration are grayed out to prevent editing. To modify these properties, edit the configuration from the **Configuration Repository** view if the changes need to be applied to all the components using that particular configuration or use the default configuration (with the required values).

15.2 Port

In the Configuration Repository view, right click on **Port** and select the **Add Configuration** option to define a named configuration port, as shown in Figure 5.

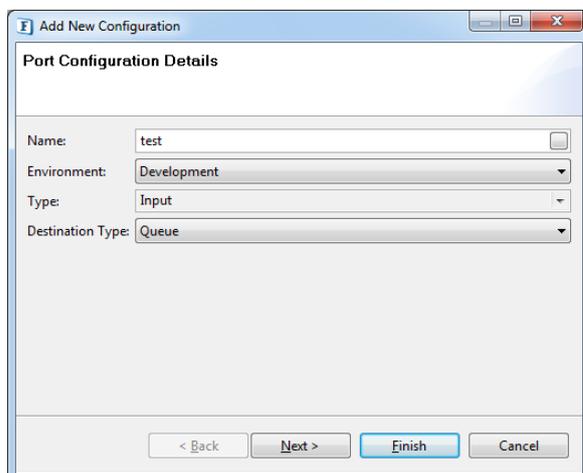


Figure 5: Port Configuration details

Provide details of the configuration name, environment, port type and destination type and click **Next**.

Note: The configuration defined for a particular environment can be assigned to elements in an Event Process only when the Event Process is in the same environment as that of the configuration.

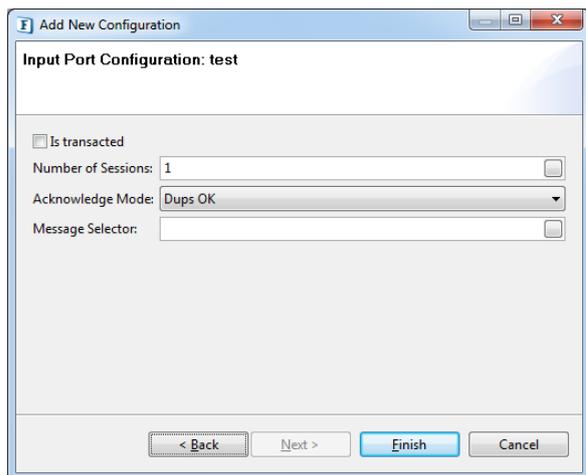


Figure 6: Input Port Configuration

Provide the port messaging configurations such as transaction details, Number of Sessions, message selector and so on and click **Finish** to save the configuration as a Named Configuration as shown in Figure 6.

Once Named Configurations are added to the repository, a User has the choice to either define Port properties or to refer to an existing Port Named Configuration of the same type. This option is provided in the Messaging tab of Port properties.

In Figure 7 when the input port of a component (chat2) is selected, the property **Configuration** lists all the available input port Named Configurations. By default no Named Configuration is used and the property is set to **<None>**.

When a configuration is selected, the configuration details are loaded and the property fields are then grayed out to prevent further editing.

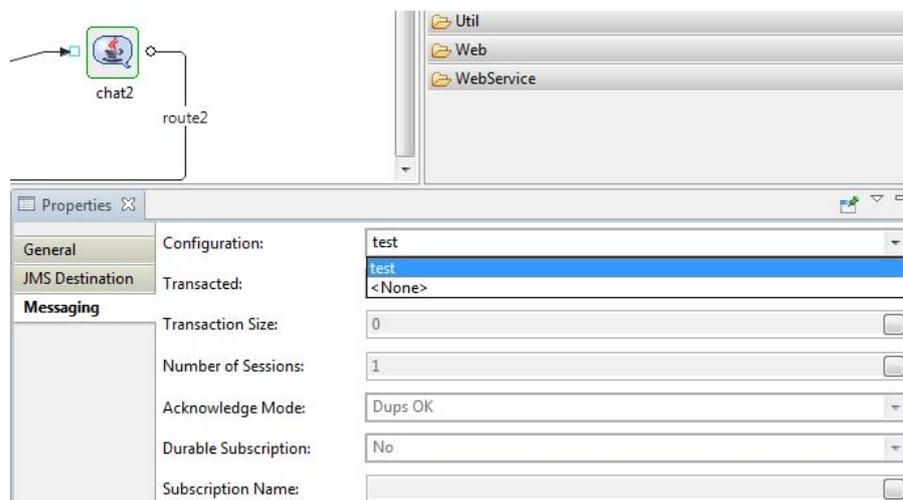


Figure 7: Configuration selection in port properties

15.3 Resource

The Resource Configuration points to the part of the component configuration that is applicable to more than one component. Resources such as XSD schemas come under Resource Configurations as well.

For example, the Database Connection Configuration can be applied to all database components in the Database category in the Service Palette.

To define a resource configuration, right-click on **Resource** in the Configuration Repository view and the select **Add Configuration** option. A dialog is displayed as shown in Figure 8.

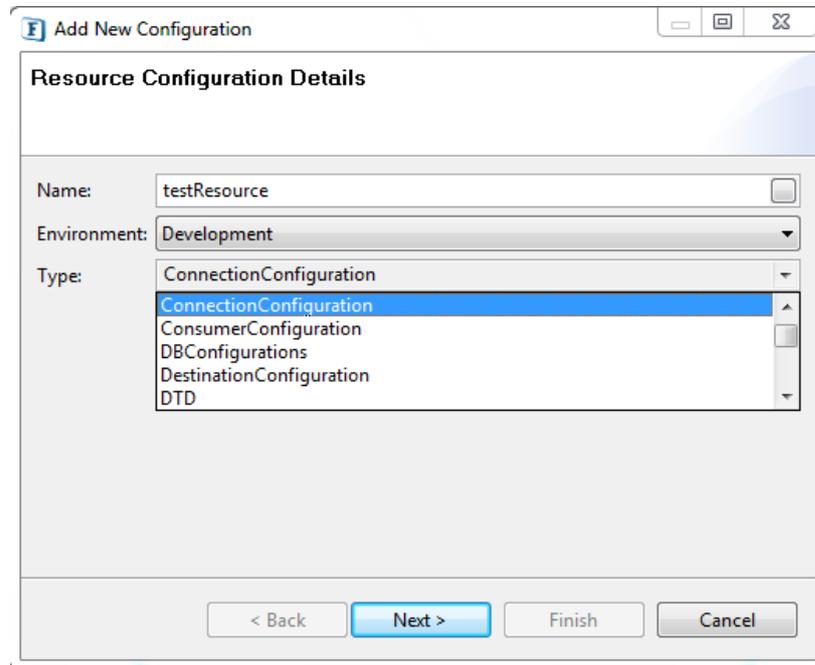


Figure 8: Resource Configuration

Provide basic details of the configuration and select the type of configuration required. Properties are automatically updated on the next page depending upon the type of configuration selected.

Click **Next** and provide configuration details that need to be filled in and click **Finish** to save the configuration, as shown in Figure 9.

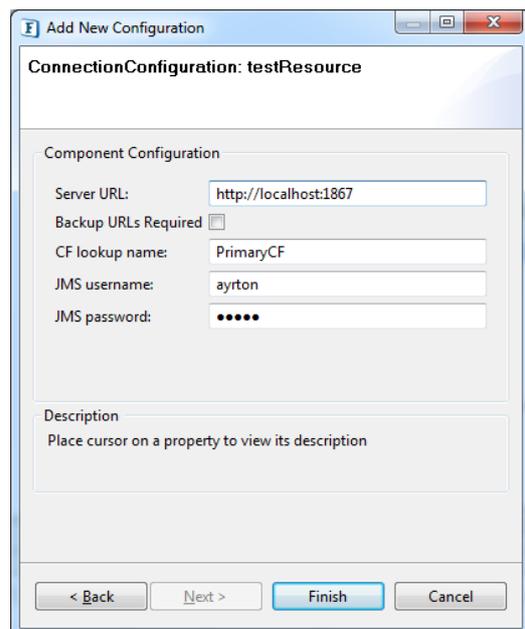
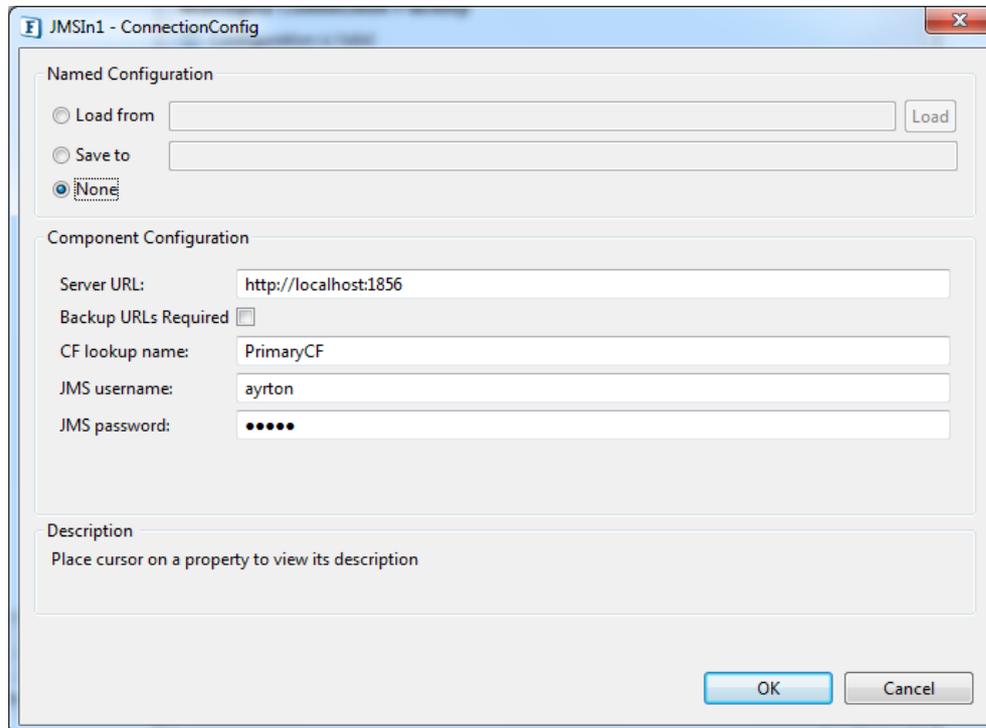


Figure 9: Connection Configuration

The configuration type Connection Configuration is saved to the repository and can be used by components that have Connection Configuration as part of their configuration settings.

Open Custom Property Sheets of a component that has Connection Configuration details could be, for example, JMS: 4.0. Click the ellipsis button against the Connection Configuration property. This launches the dialog box shown in Figure 10.



The dialog box titled "JMSIn1 - ConnectionConfig" contains the following elements:

- Named Configuration:** Three radio buttons: "Load from" (unselected), "Save to" (unselected), and "None" (selected). There is a "Load" button next to the "Load from" field.
- Component Configuration:** A group of fields:
 - Server URL:
 - Backup URLs Required:
 - CF lookup name:
 - JMS username:
 - JMS password:
- Description:** A text area containing the text "Place cursor on a property to view its description".
- Buttons:** "OK" and "Cancel" buttons at the bottom right.

Figure 10: Connection Configuration

By default, no named configuration is used. To use a Named Configuration, select the **Load from** option and type the configuration name and click the **Load** button to load configuration details, as shown in Figure 11.

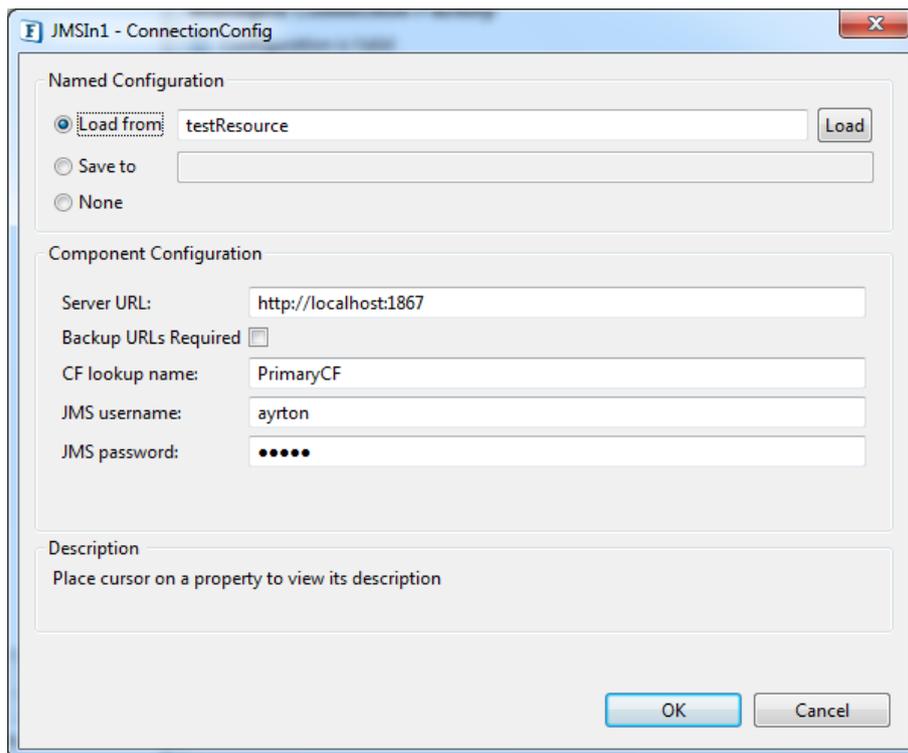


Figure 11: Loading Configuration

Similarly, a resource configuration can also be created using the **Save to** option in the Named Configuration section.

15.4 Route

A route configuration can be created using the **Add Configuration** option in the Configuration Repository view. Route Configuration contains route specific properties such as Compress Messages, Encrypt Messages and Durable Subscription.

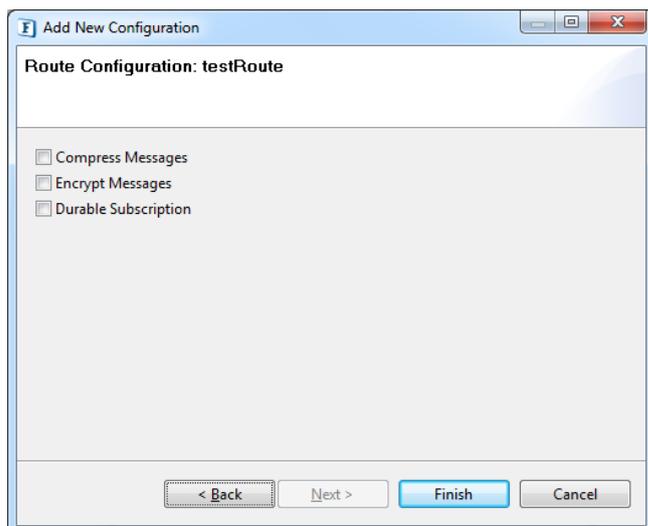


Figure 12: Route Configuration

To use a route named configuration on a route, select the route and in the route properties move to the **Messaging** section. A property called Configuration lists all route named configurations.

Select the configuration required. The corresponding configuration properties are automatically updated, though grayed out to prevent editing, as shown in Figure 13.

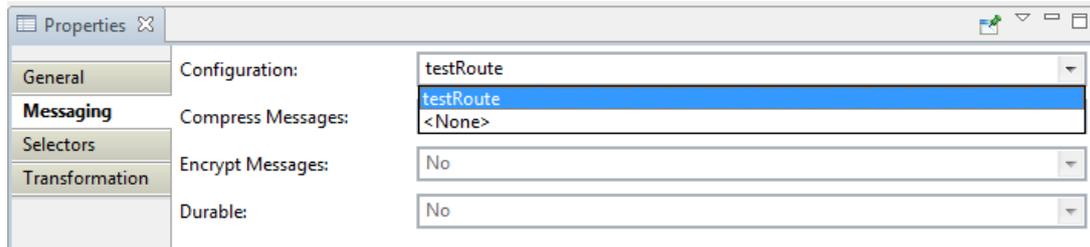


Figure 13: Selecting Route Configuration

15.5 Runtime Arguments

To add Runtime Arguments Named Configuration, right click on **Runtime Arguments** in the Configuration Repository view and select the **Add Configuration** option. A dialog box is displayed as shown in Figure 14.

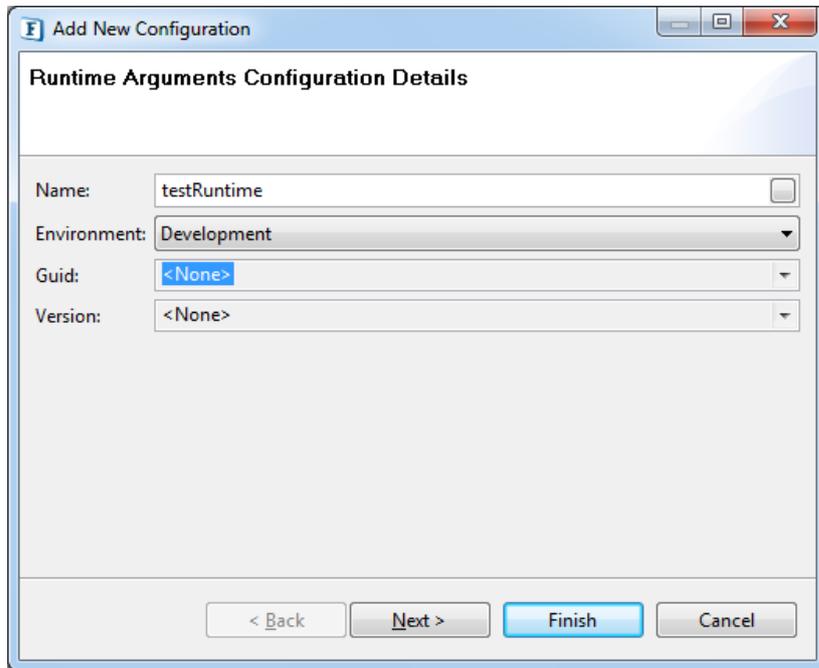


Figure 14: Runtime Arguments Configuration

In addition to basic details such as the configuration name and environment, this page contains the *Guid* and *Version* fields.

Few components like Sender, Receiver and so on contain component specific runtime arguments. To provide these component specific runtime arguments the User must select the components *Guid* and *Version*. If no *Guid* and *Version* are selected then the common runtime arguments will be shown in the next page.

Click **Next** to provide values for runtime arguments and click **Finish** to save the configuration, as shown in Figure 15.

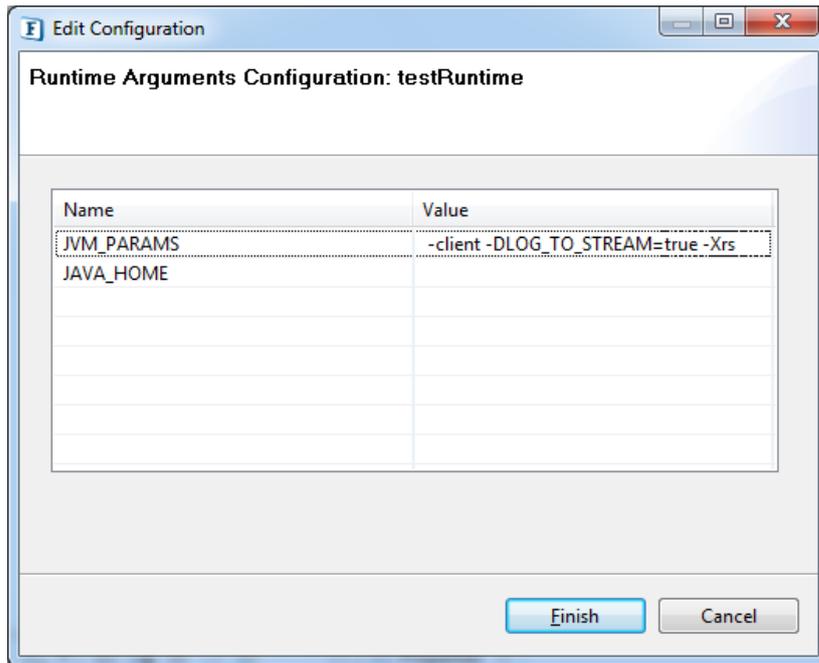


Figure 15: Defining Runtime Arguments Configuration

To use Runtime Argument Named Configurations, select a component and in the properties view move to Runtime Arguments section.

The property **Configuration** lists all available Runtime Arguments configurations. Select the configuration. Properties corresponding to the configuration are loaded automatically. The properties entered are grayed out to prevent further editing, as shown in Figure 16.

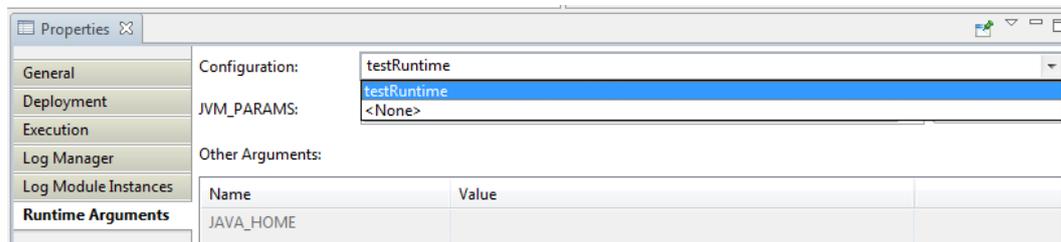


Figure 16: Using Runtime Arguments Configuration

15.6 Selector

Message selectors on the route being used can be defined in Edit Configurations. To add a configuration, right click on Selectors in the Configuration Repository view and select the **Add Configuration** option. The required selectors can be defined and saved as a Named Configuration using this dialog box displayed in Figure 17.

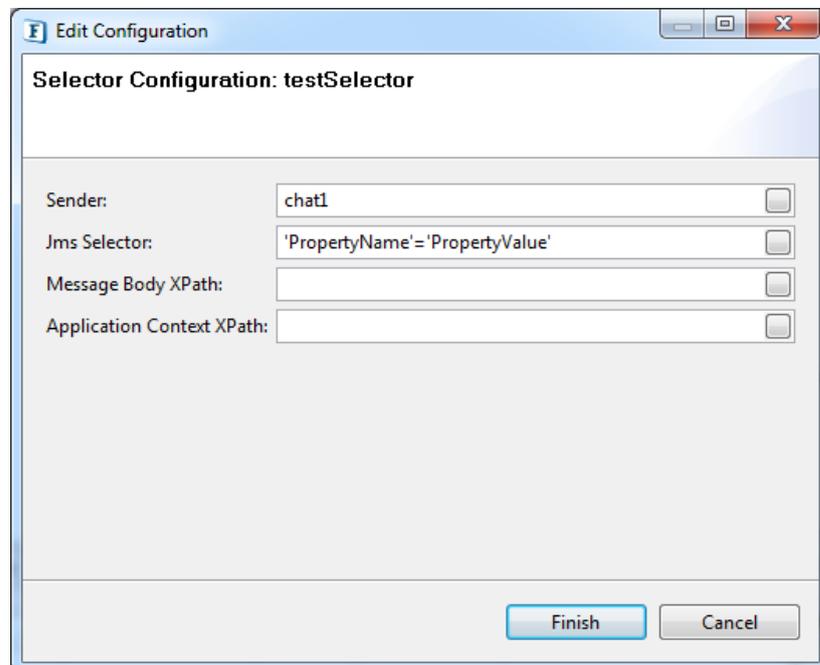


Figure 17: Selector Configuration

To use the selector on the route being used, navigate to the **Selectors** tab in the Route properties section and select the Named Configuration from the drop down menu in the property **Configuration**. The fields of the properties entered are grayed, as shown in Figure 18, so that they can not be edited. The properties entered are selected at runtime.

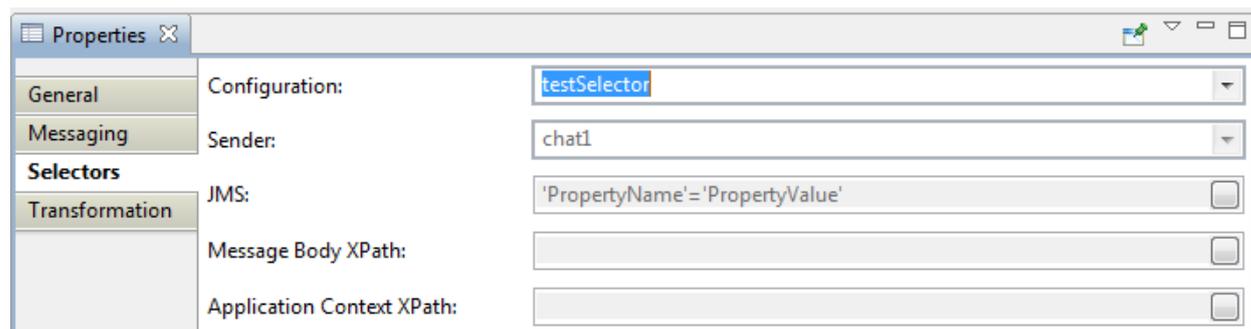


Figure 18: Using Selector Configuration on route

15.7 Service

To add a new service configuration, right click on Service in the Configuration Repository view and select the **Add Configuration** option. Select the Guid and the Version of the service. Few components like FTPGet have multiple configuration types. Select the appropriate type and click **Next**, as shown in Figure 19.

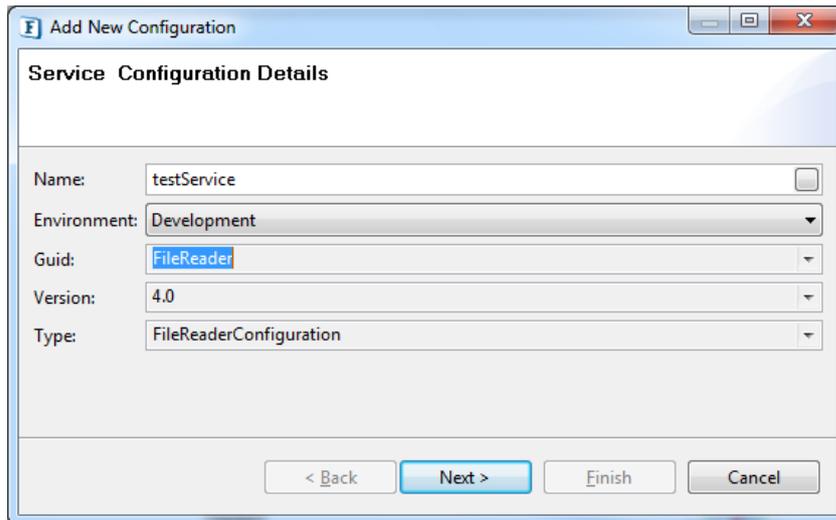


Figure 19: Service Configuration

To demonstrate the addition of a New Configuration, the FileReader is used as an example and the FileReader configuration details are displayed. Click on **Next** and the values of the configuration. Click **Finish** to save the configuration, as shown in Figure 20.

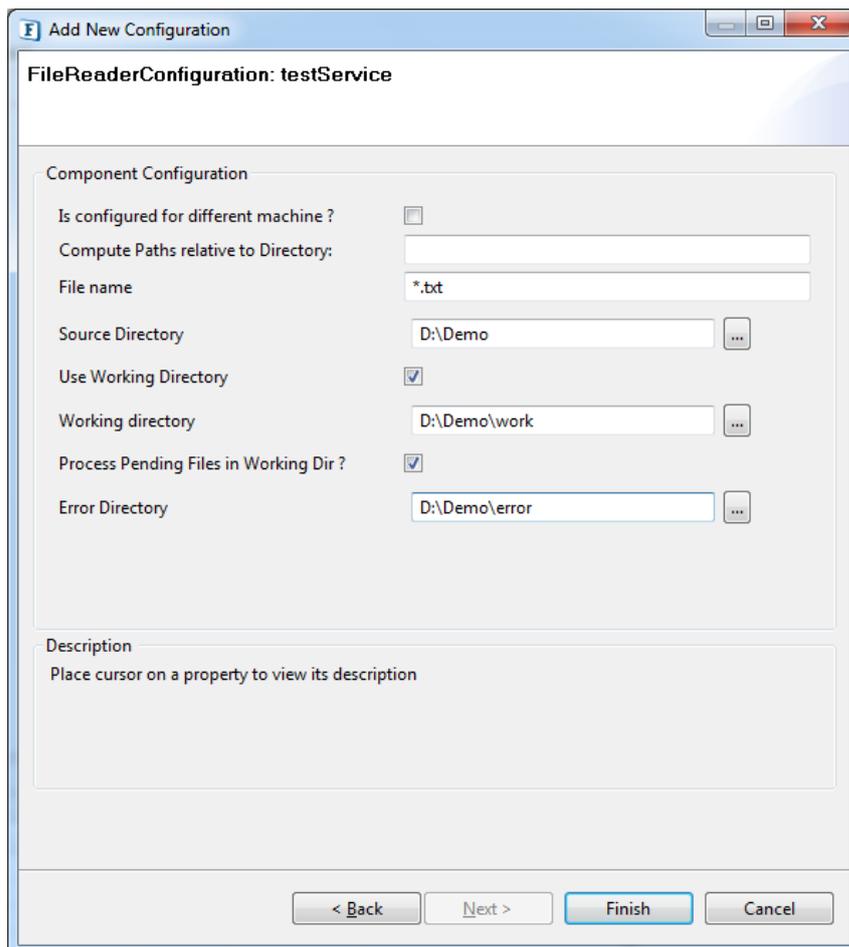


Figure 20: FileReader Configuration

To use the newly added configuration, open the Custom Property Sheet (CPS) of the component and click on the ellipsis button against the property **FileReader Configuration**, as shown in Figure 21.

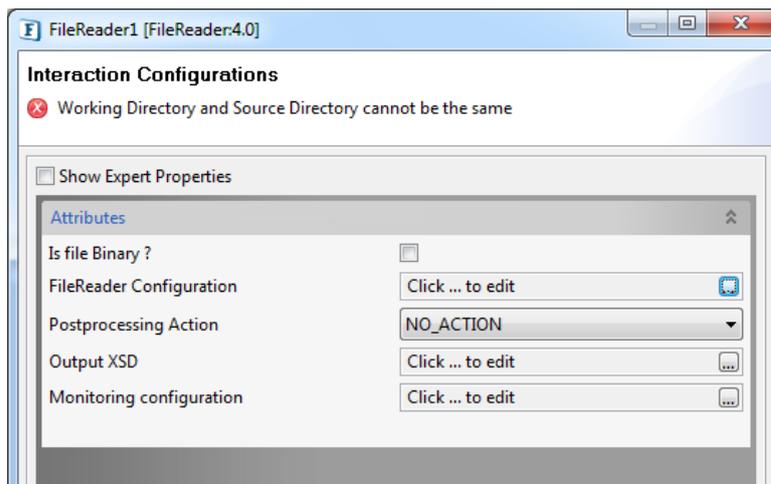


Figure 21: FileReader Custom Property Sheet

The **FileReader Config** dialog box is displayed. Select the **Load from** option and add the configuration name. Click on the **Load** button to load the configuration details. Provide additional configuration details for the FileReader and click on **Finish** to save the CPS settings. The named configuration that is referred by the component will be used by the component both at the time of configuration and at the runtime.

If the configuration property is a manageable property then it can be viewed from the Environment Properties section within the Event Process properties.

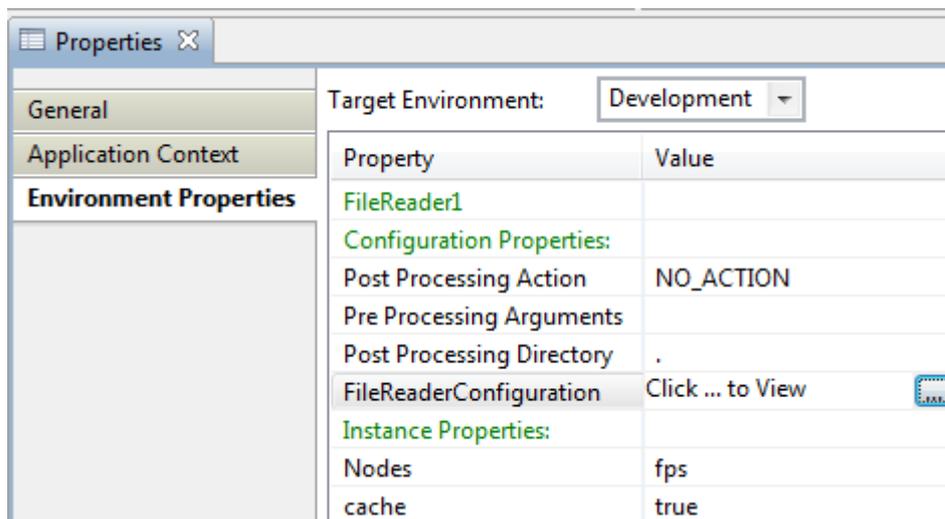


Figure 22: Event Process Environment Properties

As shown in Figure 19, clicking the ellipsis button against the property **FileReaderConfiguration** launches the FileReader configuration dialog box (referred to above).

15.8 Transformation

The Transformation configuration can also be defined as a Named Configuration and be used routes. To define a Transformation Configuration, right click on **Transformation** in the Configuration Repository view and select **Add Configuration** option, as shown in Figure 23.

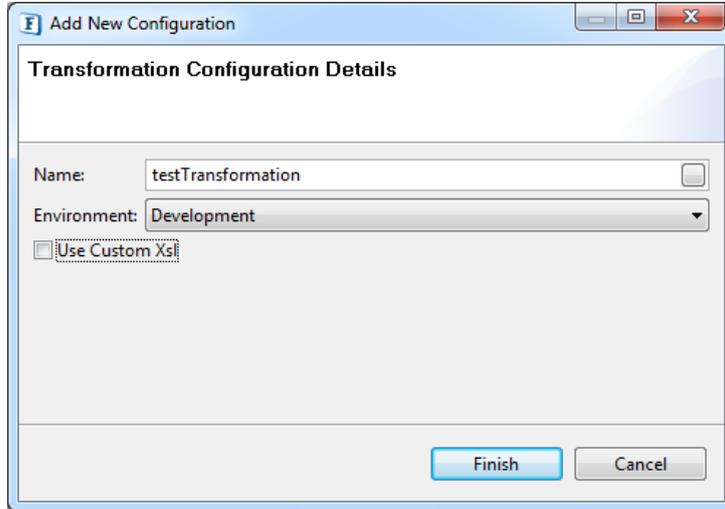


Figure 23: Transformation Configuration

If the **Use Custom XSL** option is selected, on clicking **Finish** a new dialog box (Custom XSL dialog) is displayed. Provide custom XSL and other configuration details and click on **Ok** to save the configuration as a Named Configuration, as shown in Figure 24.

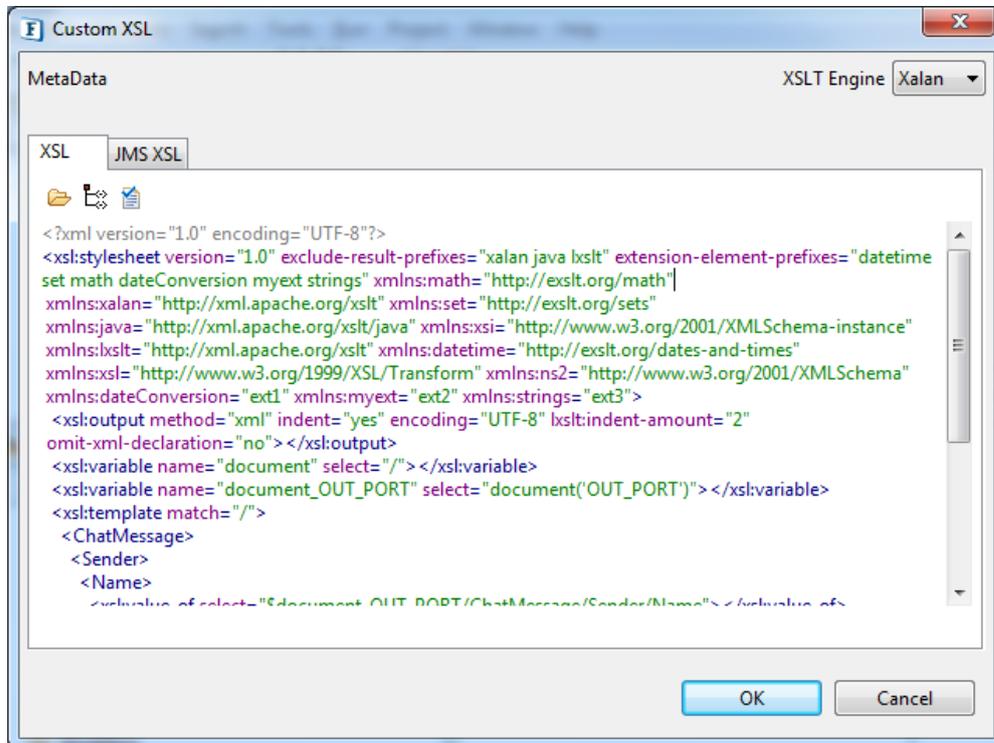


Figure 24: Custom XSL dialog

If the **Use Custom XSL** option is not selected, the Mapper project will be opened. Enter both **Input** and **Output** schemas and save the transformation. This transformation will then be saved as a Named Configuration.

To use the Transformation Named Configuration, right click on a route and select the **Configure Transformation → Use Named Configuration** option, as shown in Figure 25.

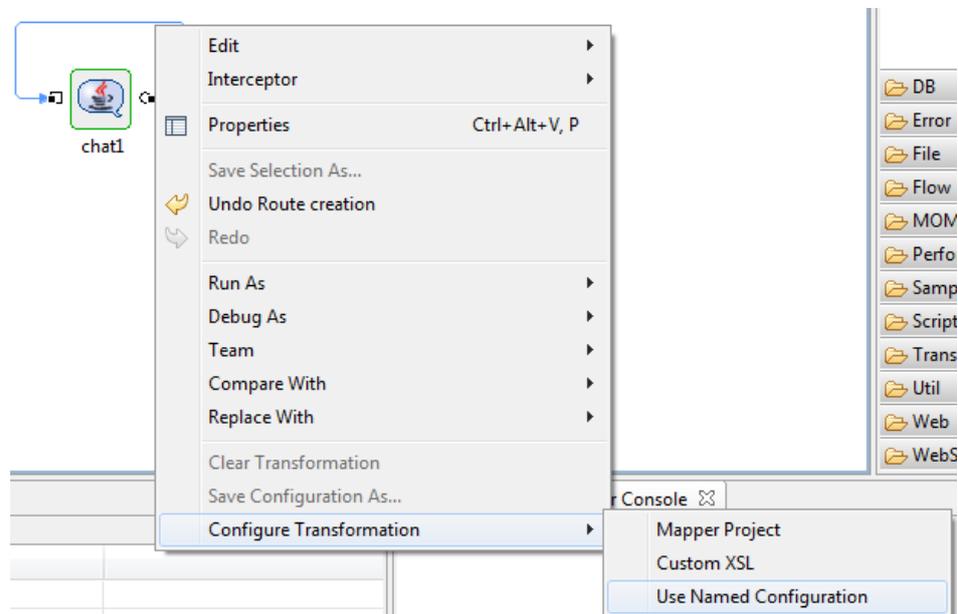


Figure 25: Using named configuration on route

A dialog box is displayed where the configuration to be used can be selected, as shown in Figure 26.

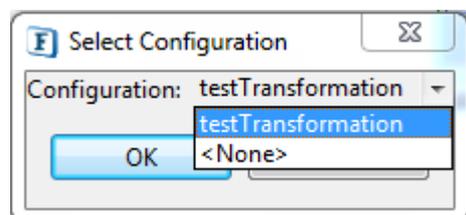


Figure 26: Selecting transformation configuration

Select the configuration and click on **OK**. The configuration selected will be applied on the route. The route is shown in bold indicating the Transformation. The route name is shown in bold indicating the Named Configuration used, as shown in Figure 27.

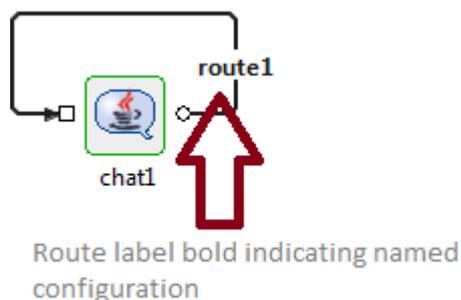
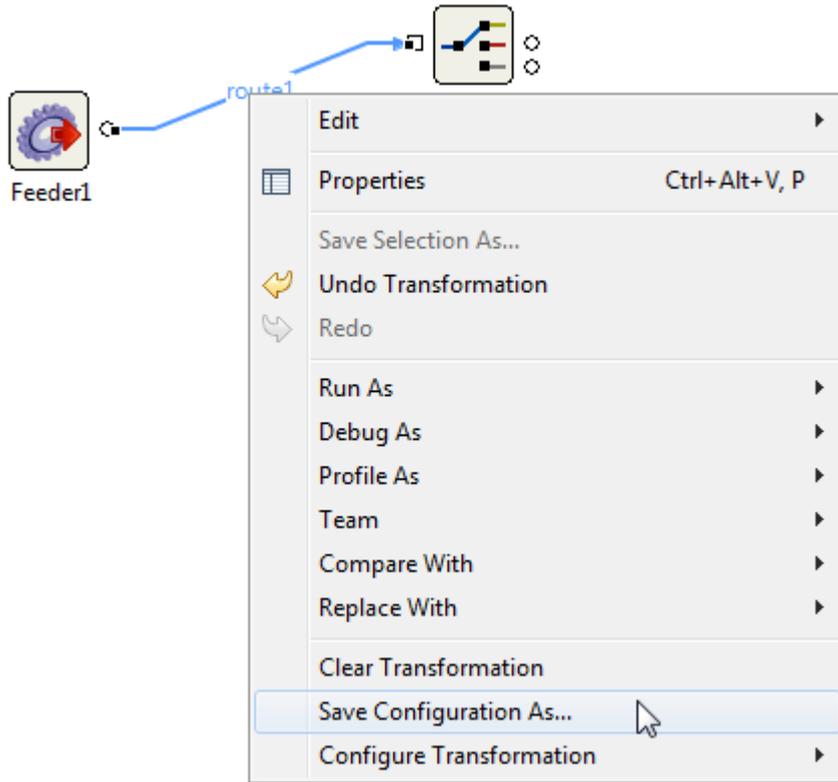


Figure 27: Route using a named configuration

Transformations that are already defined on a route can also be saved in the Configuration Repository. To save a route's transformation as a **Named Configuration**, right click on the route and click on **Save Configuraiton As..**

**Figure 28: Saving a Route Transformation as a Named Configuration**

The Save Transformation Configuration dialog is opened. Provide a name for the configuration and save. A new Transformation Configuration will be added to the Configuration repository and the route name will be shown in **bold** to indicate that a named configuration is used

15.9 Workflow

The Workflow Configurations to be used on a component port can be defined as Named Configurations. To define a Workflow Configuration, right click on **WorkFlow** in the Configuration Repository view and select the **Add Configuration** option. Enter the configuration name, select the environment and click the **Next** button. Enter values for Work Flow Configuration and click **Finish** to save as the Named Configuration, as shown in Figure 29.

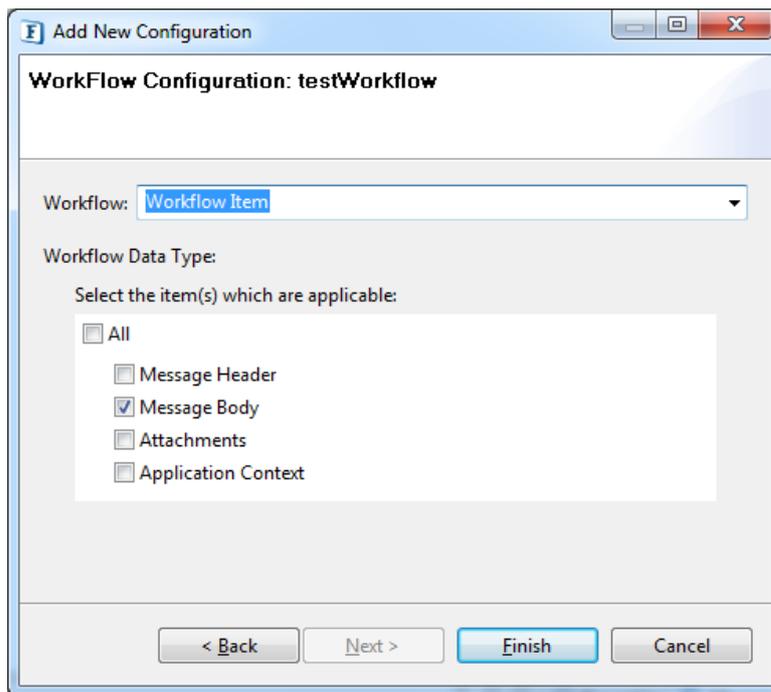


Figure 29: Workflow Configuration

To use the Workflow Named Configuration, select a component port and in the Properties view. Select the configuration as shown in Figure 30. The configuration will be applied on the selected port.

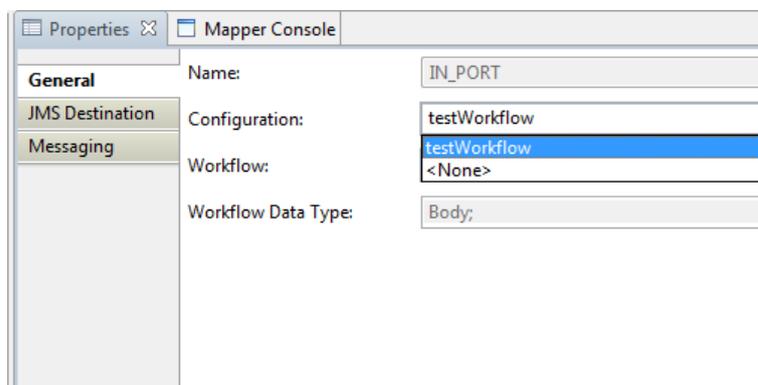


Figure 30: Using workflow configuration on port

15.10 Context Menu Options

On selecting a configuration, the following options are available in the context menu in the Configuration Repository View, as shown in figure 31.

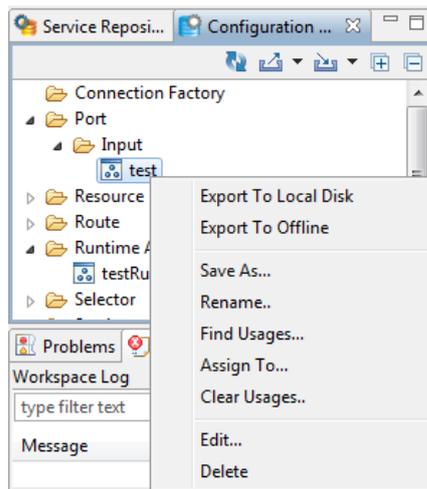


Figure 31: Context menu options

1. Export to Local Disk

Select the Export to Local Disk option to export the configuration to the local disk. The configuration will be stored in a zip file.
2. Export to Offline

Select this option to export the Named Configuration Offline. The Export to Server option is available in the Offline mode.
3. Save As

Save As is used to save the configuration under a different name.
4. Rename

The Rename option is to rename the configuration.
5. Find Usages

Find Usages is used to view the usage of a particular Named Configuration. The usage of Named Configurations are listed in a tree format. A sample of configuration usage is shown Figure 32.

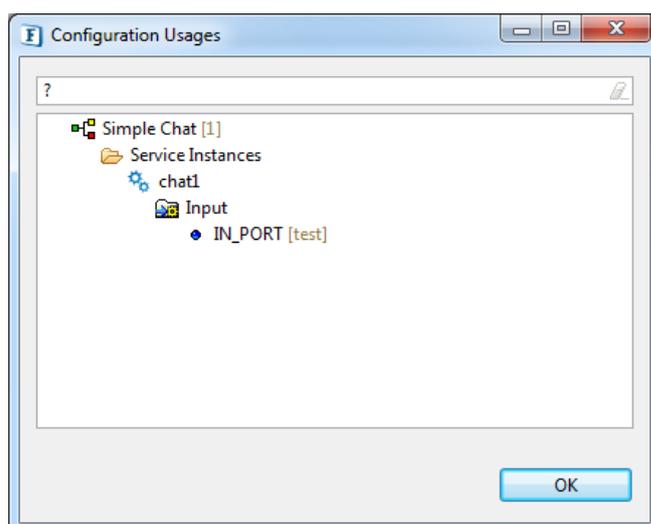


Figure 32: Configuration usages dialog

6. Assign To

A Named Configuration can also be assigned from the Configuration Repository view. To assign a configuration, right click on the configuration and select the Assign To... option, as shown in Figure 33.

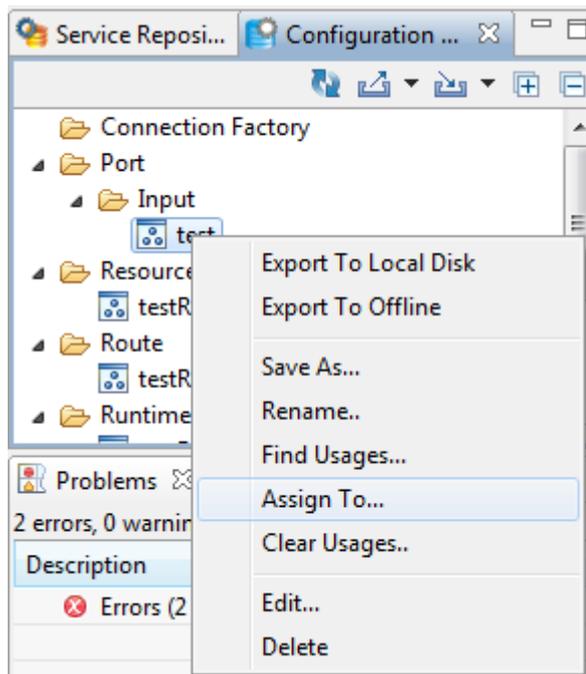


Figure 33: Assigning configuration from repository view

A dialog box is displayed listing all the valid entries to which the selected configuration can be assigned, as shown in Figure 34.

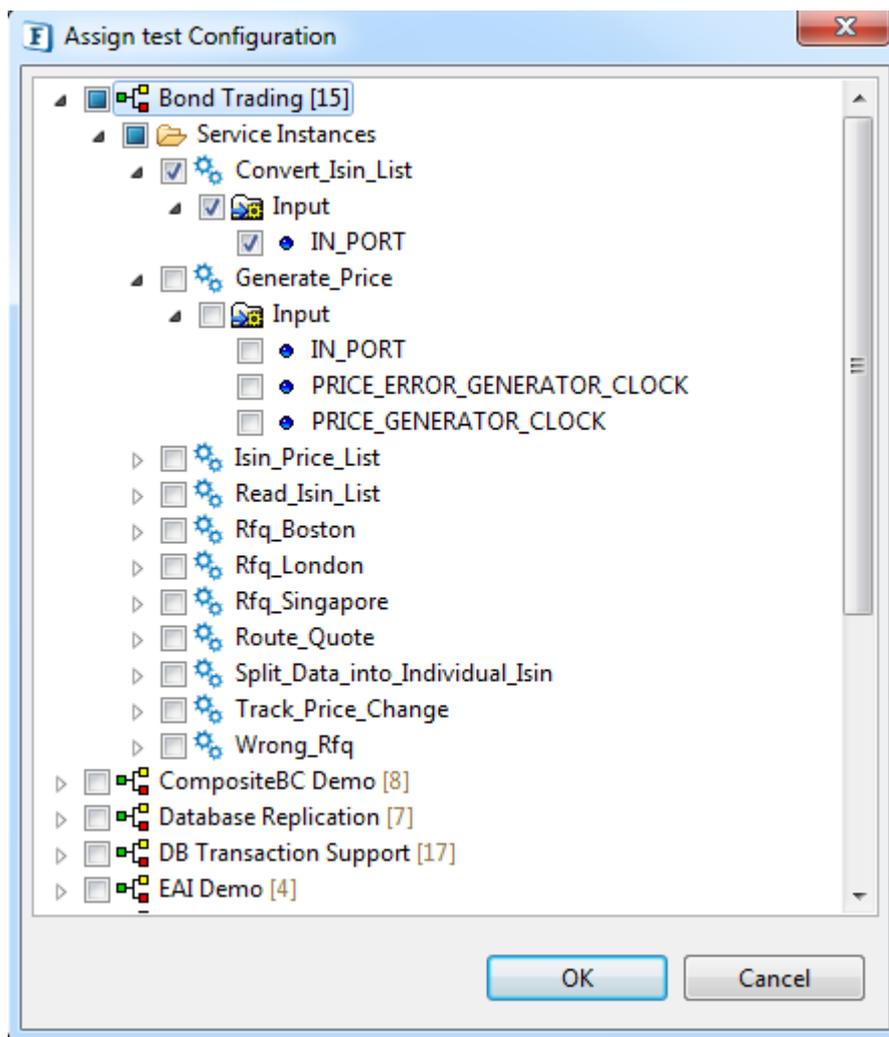


Figure 34: Assign configuration dialog

Select the entry to which the configuration is to be assigned and click on **OK**.

Note:

- a. The **Assign To...** option will not list an entry if the configuration entered is already assigned.
- b. The Assign To... option is not present for the Service and the Resource configurations since these cannot be directly assigned from this view.

7. Clear Usages

The Clear Usages option clears the configurations used. On selecting this option all the usages are displayed in a tree. Select the configuration usages to be cleared and click on OK as shown in Figure 35.

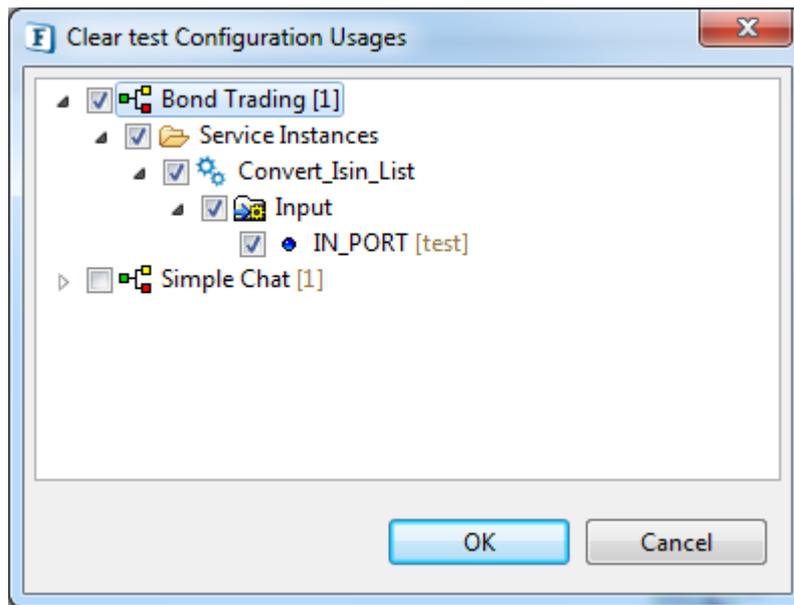


Figure 35: Clear configuration usages

Note: The Clear Usages option is not present for the Service and the Resource configurations. The Clear Usages option clears the usage of configurations and resets usage to the default settings.

8. Edit

The Edit option edits existing configurations.

Note: If a configuration is edited at runtime it can require that the Component or the Event Process be restarted.

9. Delete

The Delete option deletes the configuration.

15.11 Points to Note

1. If an Event Process uses a Named Configuration, when it is export all the Named Configurations used by the Event Process will be listed. Users can select the configurations that are to be included in the Event Process export, as shown in Figure 36.

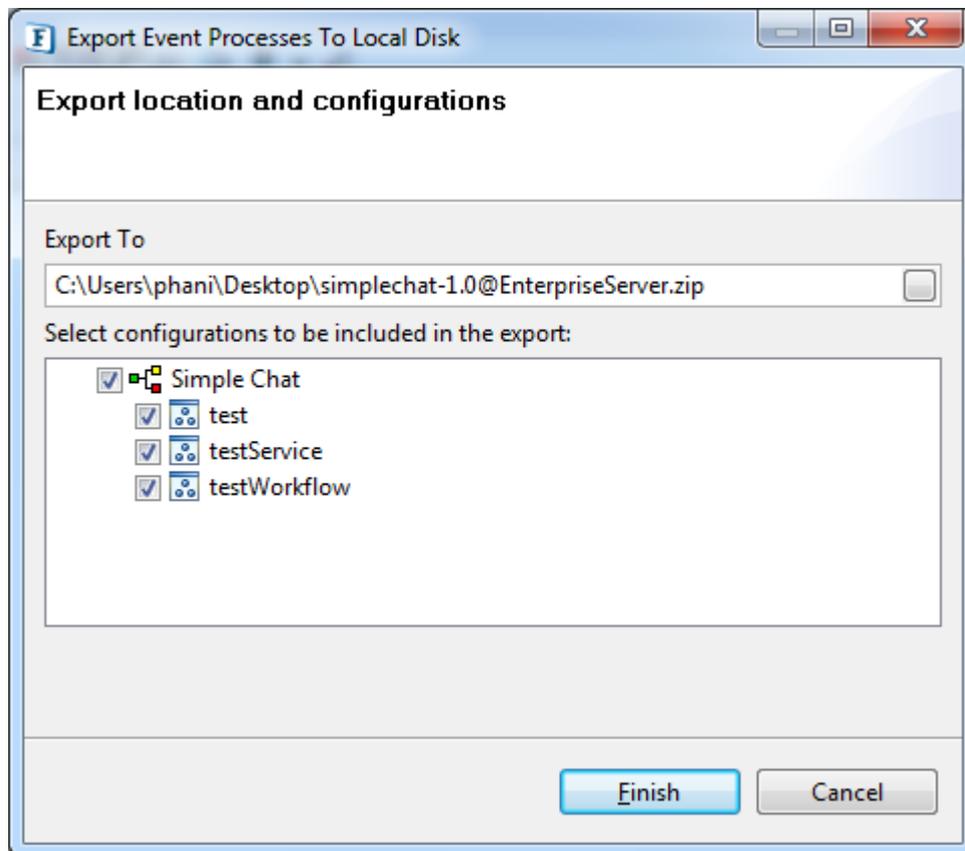


Figure 36: Export Event Process dialog

If an Event Process uses Named Configurations and if these are deleted from the repository, then during the export of the Event Process a visual symbol indicates that the Names Configurations are missing from the repository. Figure 37 show a configuration called testRuntime that is missing from the repository.

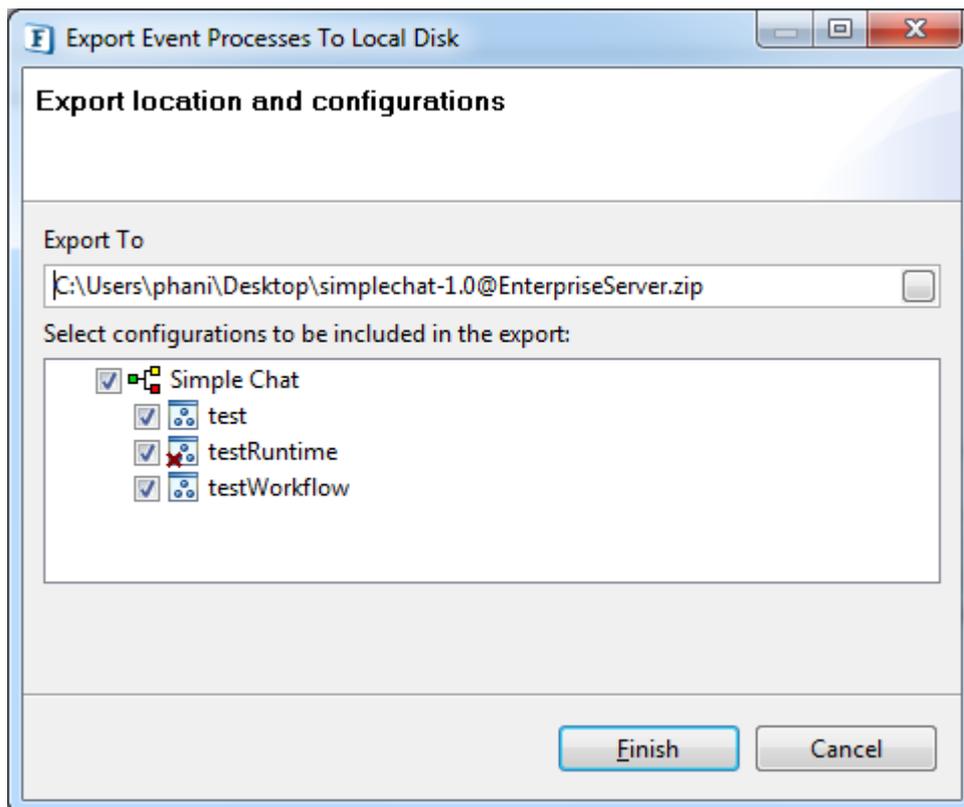


Figure 37: Export Event Process listing missing configurations

When an Event Process is import, all configurations present are listed. If the configurations already exist then they will be shown in red color and the user is provided with options to either overwrite or not import the configuration displayed.

2. To view all the configurations used in an Event Process, right click on Event Process and select the **Show Configuration Usages** option as shown in Figure 38.

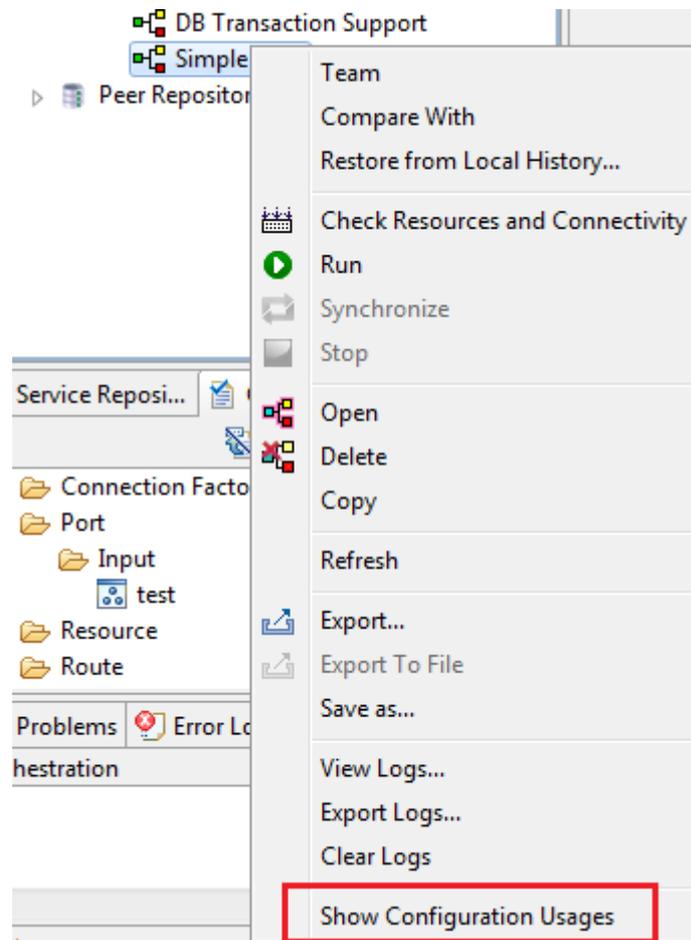


Figure 38: Show configuration usages option on Event Process

All the configurations used are displayed, as shown in Figure 39.

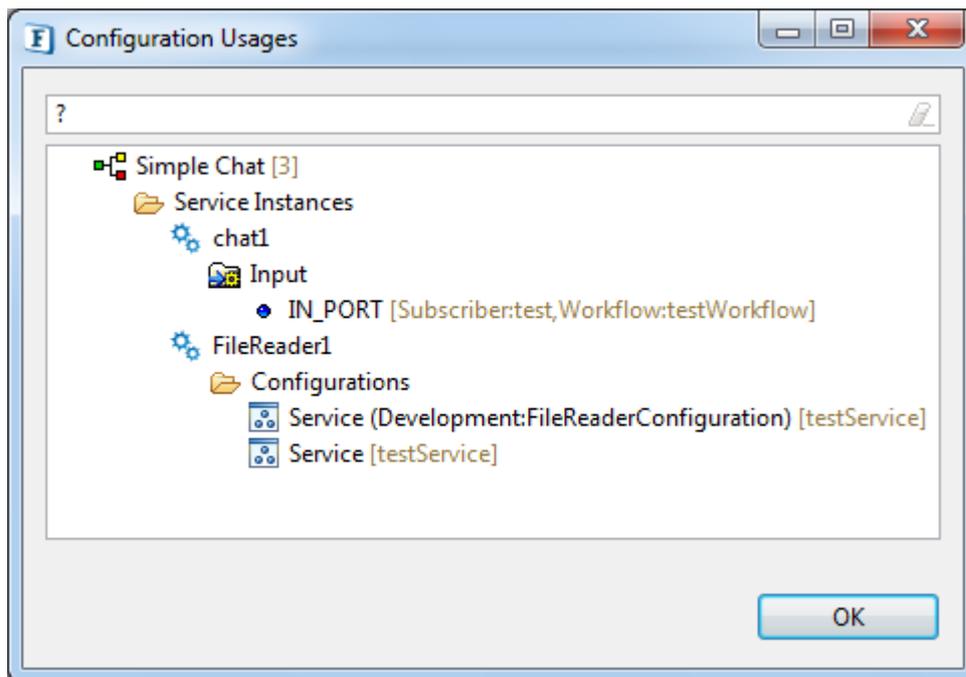


Figure 39: Event Process configuration usages

3. The Undo support is present for the **Assign To** and **Clear Configuration** actions. This option is present in the Configuration Repository view toolbar and displayed in the toolbar only when a configuration is assigned from this view. Please refer to Figure 40. **Note:** This option should be used with care.

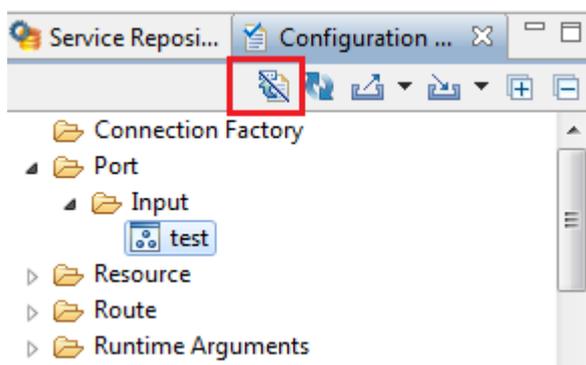


Figure 40: Undo Assign/Clear configuration

4. A configuration defined in a particular environment can be applied to elements in an Event Process only if the Event Process is within the same environment.

Chapter 16: Connection Management

The Connection Management tool is used to manage both Admin and JMX connections to a Server. To open the Connection Management dialogue box click Window → Open Perspective → Other and select Connection Management.

The dialogue box displayed consists of two views – the Server Connections view and the Properties view, as shown in Figure 1.

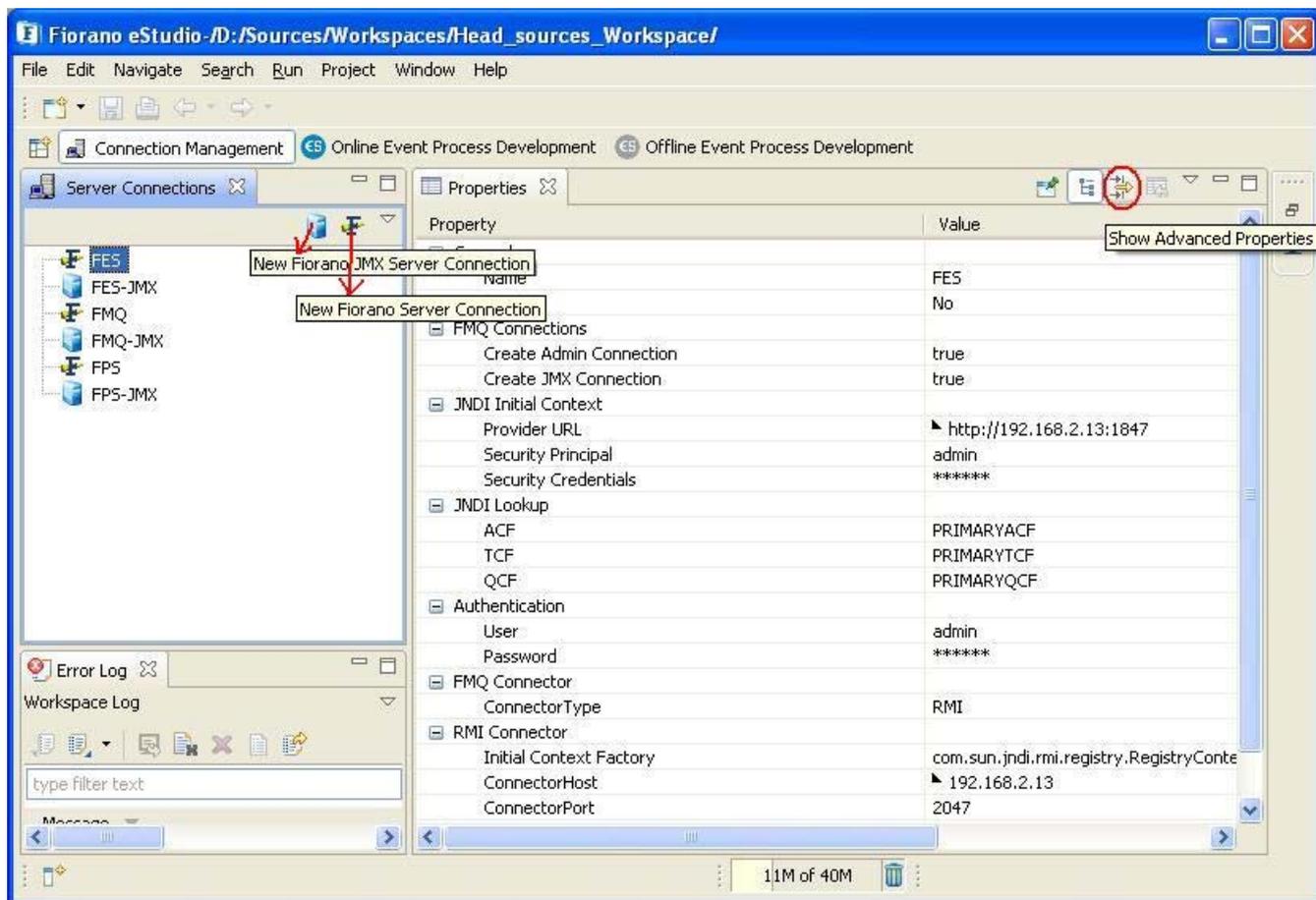


Figure 1: Connection management perspective

- **Server Connections:** Displays the connection nodes through which connections (Admin/JMX) are made to the Server.

Each Server Node displays the Shutdown and Restart options by right-clicking over the menu.

- **Actions on the View Toolbar:**
 - **New Fiorano JMX Connection:** This option is used to create a new JMX connection Login node.
 - **New Fiorano Server Connection:** This option is used to create a new Admin Connection Login Node.

- **Properties:** All the properties for each administered object from the Server connections view is displayed in this view.
- **Actions on the View Toolbar:**
 - Show Advanced Properties: This options is used to show/hide advanced properties if available for any selected node.

The '▲' icon is displayed beside the value of the property to indicate that there has been a change from in the value of the property from its default value.

16.1 Admin Connection Management

Administrative Connection to a Fiorano Server (FMQ/FES /FPS) enables performing tasks such as tracking clients connected to the Server, managing destinations and connection factories, snooping messages on a queue or topic, viewing logs and so on.

1. To login into a server, select the server node and in the properties view to specify the required connection properties, right-click on the node and select Login.
2. After Login if both the Admin and the JMX connections are successful, then six child nodes are displayed under the Server node.
 - a. If only the Admin connection is successful five nodes, excluding the logger node, will be displayed under the Server node.
 - b. If the JMX Connection is successful, only the Loggers node is displayed as a child of the Server node.

Given below is the description and usage of each node under the Server node (including FES/FPS/FMQ).

1. **Clients:** All the clients connected to the Server are displayed under this node.

Option Available:

- a. **Refresh:** This option refreshes client connections. Client connections made or lost after login are reflected in the tree diagram only when the Clients node is refreshed.
- b. **Purge:** This option is used to purge or clear messages on a Topic subscribed to by the client for messages, as shown in Figure 2. This action is useful for the durable subscriber when it is in an inactive state.

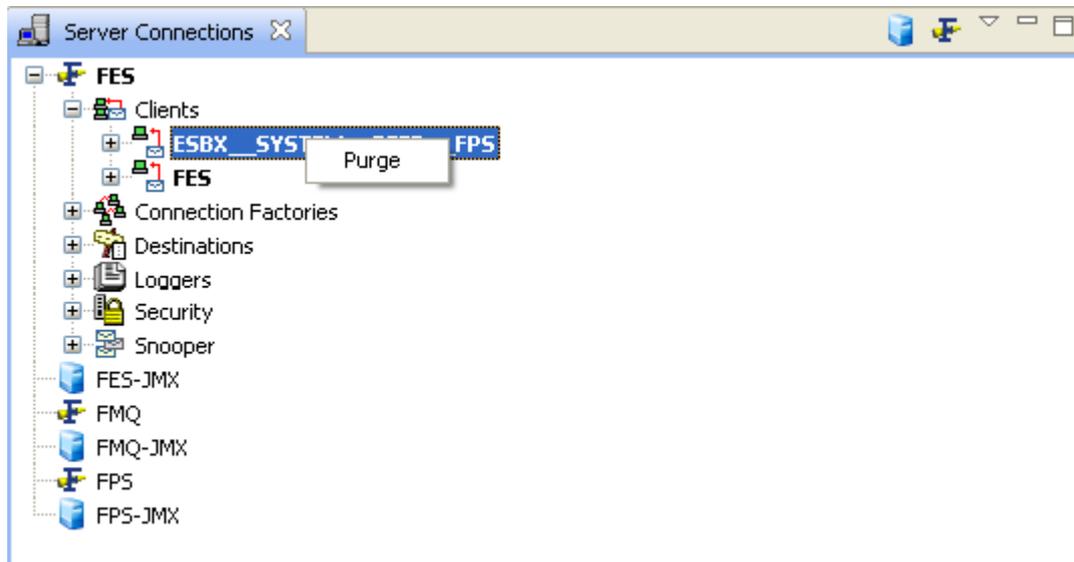


Figure 2: Purge Messages

2. **Connection Factories:** The default connection factories provided by the Server are initially displayed under this node. [Semantic check please]

New Connection Factories can be added by selecting the Add Connection factory option by right-clicking this node, as shown in Figure 3.



Figure 3: Create connection factory dialog

Right-click on the Connection Factory menu to delete the Connection Factory.

3. **Destinations:** The destinations created in the Server are displayed under this node under Queues or Topics categories.

New Queues/Topics can be added by selecting Add → Queue /Topic and right-clicking the Destinations node or by selecting Add Queue/Add Topic on the Queues and Topics nodes, as shown in Figure 4.

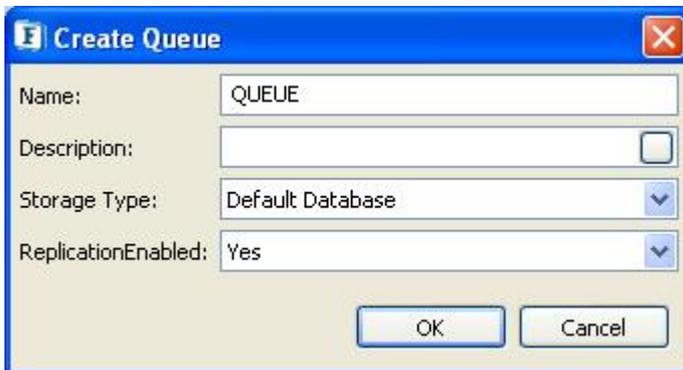


Figure 4: Create Queue dialog

Actions Associated with Queues and Topics

EditACL: This option is used to add Users to the Access Control List (ACL) where Permissions may be edited.

For adding a new ACL Entry, click Add in the EditACL dialog box, as shown in Figure 5.

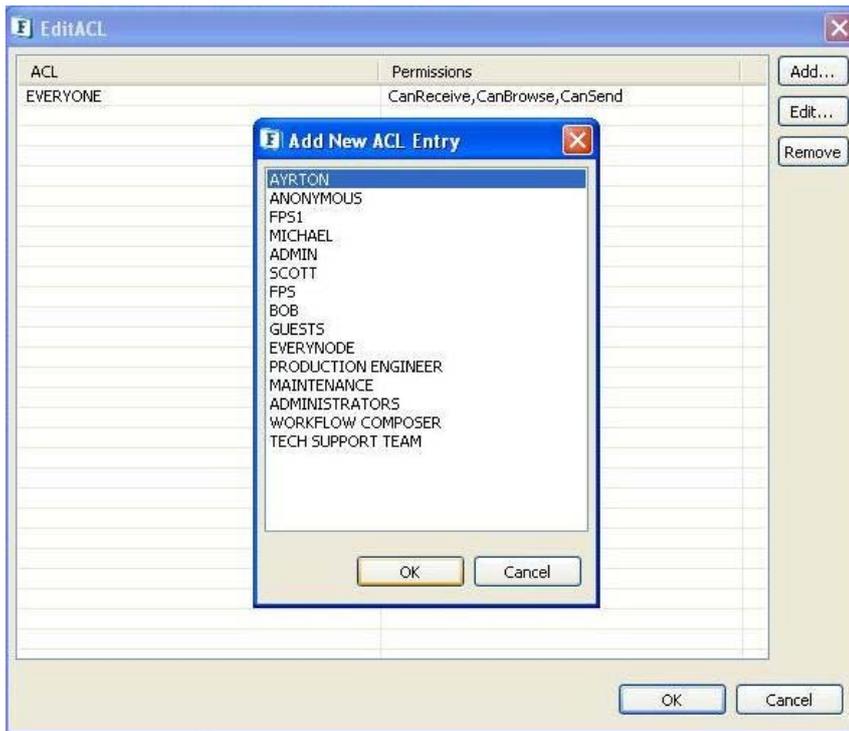


Figure 5: Edit ACL dialog

To edit a new ACL Entry, select the ACL to edit and click on Edit. The Edit Permissions dialog box displayed, as shown in Figure 6.

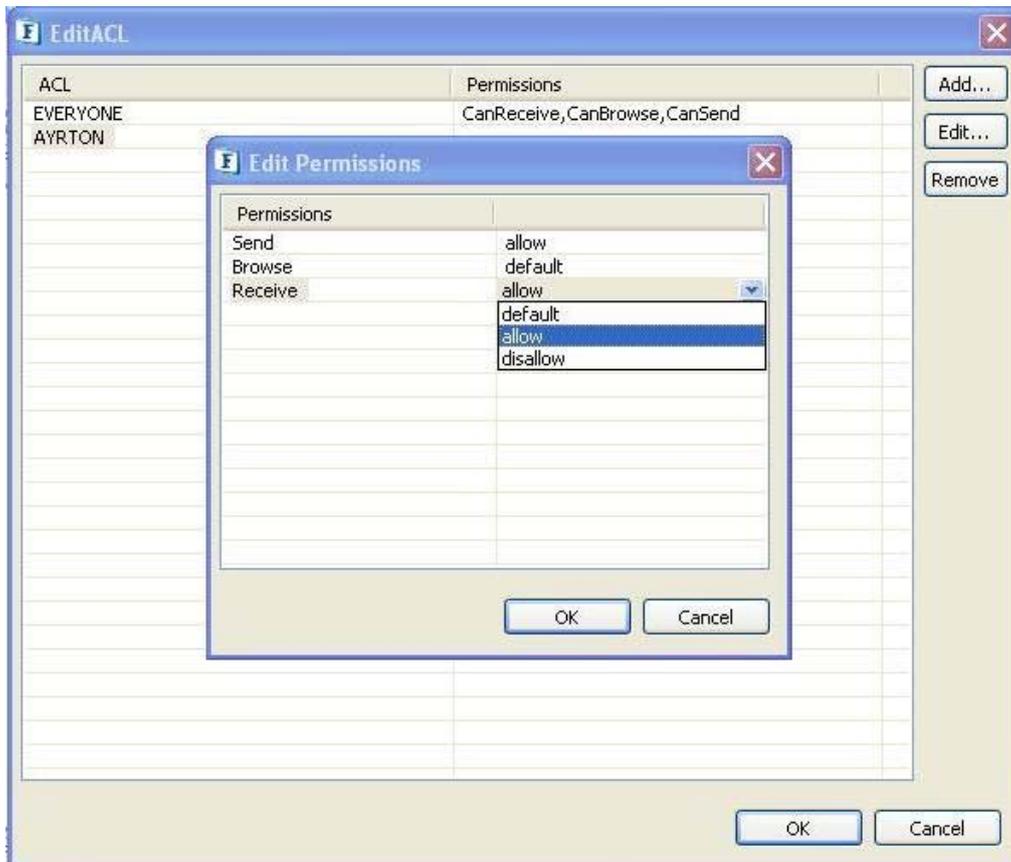


Figure 6: Editing permissions

Subscribe/Receive: This option is used to subscribe or receive messages on a destination. A dialog box is displayed, as shown in Figure 7, containing subscribed or received messages.

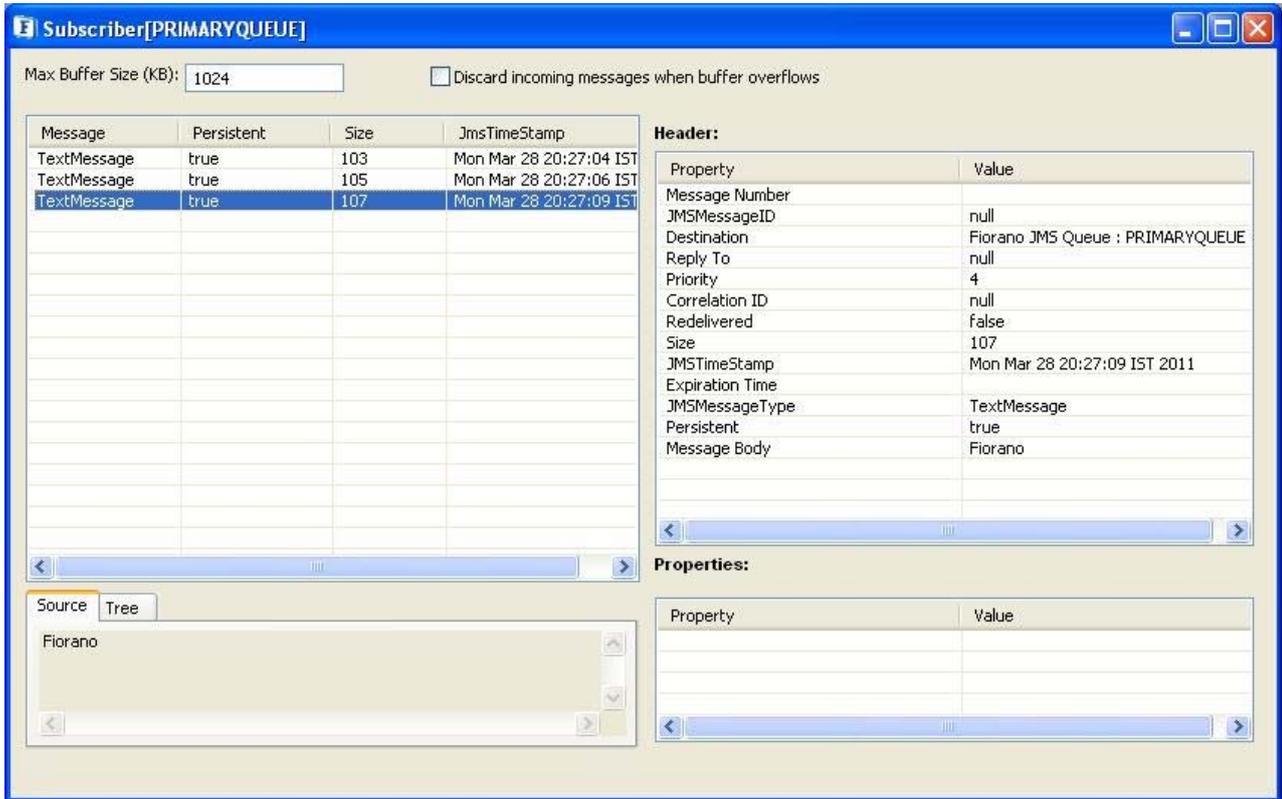


Figure 7: Subscribe messages dialog

Browse Messages: This option is used to view messages on a Queue.

Produce Text Message: This option is used to send/produce messages on a Queue or a Topic.

Delete: This option is used to delete a Queue or a Topic.

4. **Loggers**

Users that are logged in are allowed to view and manage Server logs. For example, the node Fiorano, which is a child node, contains all the Fmq and Esb related events to which an Appender or a Console Appender may be added. The same functions may be added to the Monitoring and the WebManagement nodes.

- a. To add an appender right-click on the logger and select Add ->File or Console Appender. A dialog box is displayed where New File Appender/Console Appender properties may be added. Enter the Values and click on OK to create the Appender.
- b. To View/Clear Logs of an Appender right-click and select View Logs /Clear Logs.
- c. To delete an Appender, right-click on the appender and select Delete.

5. **Security:**

Groups and Users created for access control are added under this node.

- a. To add a new Group, right-click on Groups and select Add Group.
- b. To edit the members in a Group, right-click on a Group and select Members. A dialog box is displayed, as shown in Figure 8, where new Users or Groups may be added.



Figure 8: Add members dialog

- c. To add a new User, right-click on Users and select the **Add Users** option.
- d. To change the password of a User, right-click on User and select **Change Password**. A dialog box is displayed, as Shown in Figure 9, where the password can be changed.

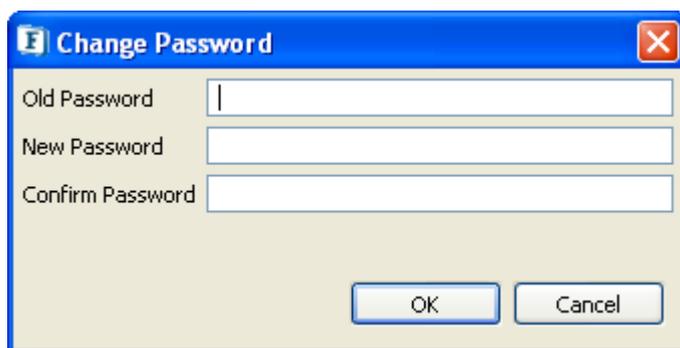


Figure 9: Change password dialog

6. Snooper

The Snooper function enables the replication of messages under a Queue/Topic to another Queue/Topic.

The Queue/Topic on which snooping is desired can be added by right-clicking on Snooper/Queues/Topics nodes and selecting Add/Remove Destinations option. A dialog box is displayed, as shown in Figure 10, where the destination list is displayed. Destinations may be added or removed from this list.

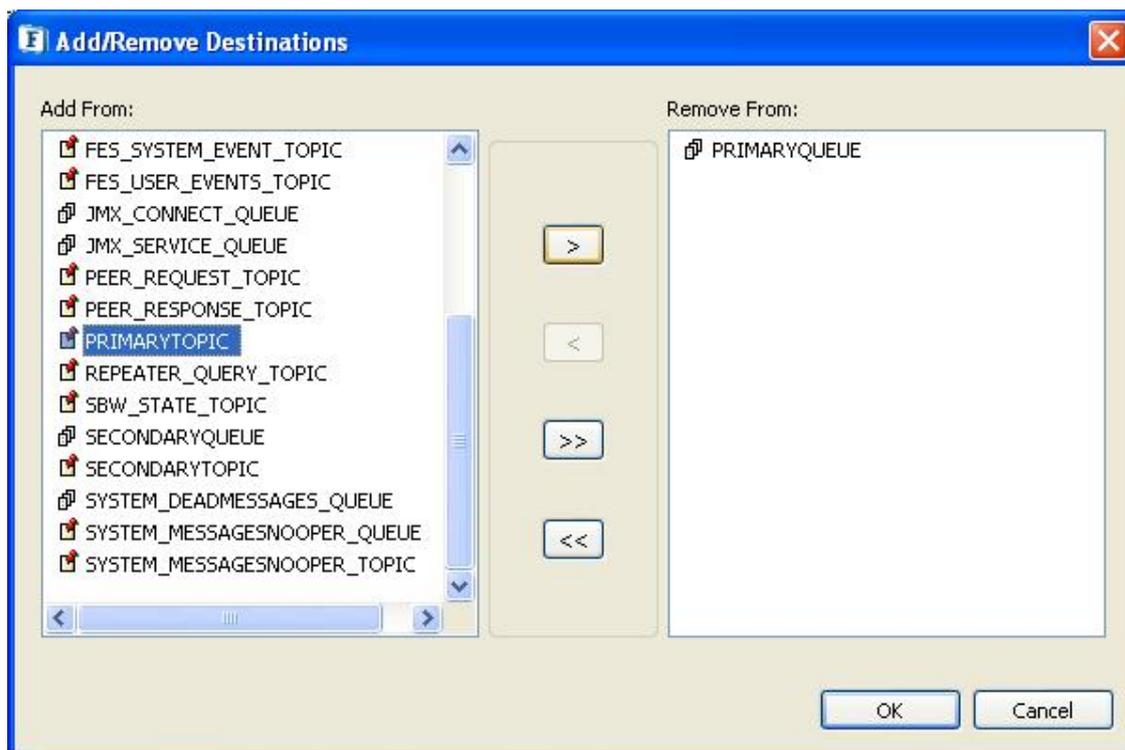


Figure 10: Add/Remove destinations

16.2 JMX Connection Management

The JMX Connection Management lets the User connect to a JMX compliant Server such as FMQ-JMX/FES-JMX/FPS-JMX and perform tasks like browsing Server mbeans, monitoring data, changing Server properties runtime and so on.

Changes made using JMX are applied on the Server that is running. Some Server configurations (such as server ports, memory settings and so on) are applied only after restarting the Server. A message dialog box instructing the Server be restarted is displayed when required.

To log into the Server, select the JMX Server node. In the properties view specify the required connection properties and right-click on the node and select Login.

On successful login, the JMX tree containing domains (🔗) and mbeans (🔗 and 📁) is displayed, as shown in Figure 11.

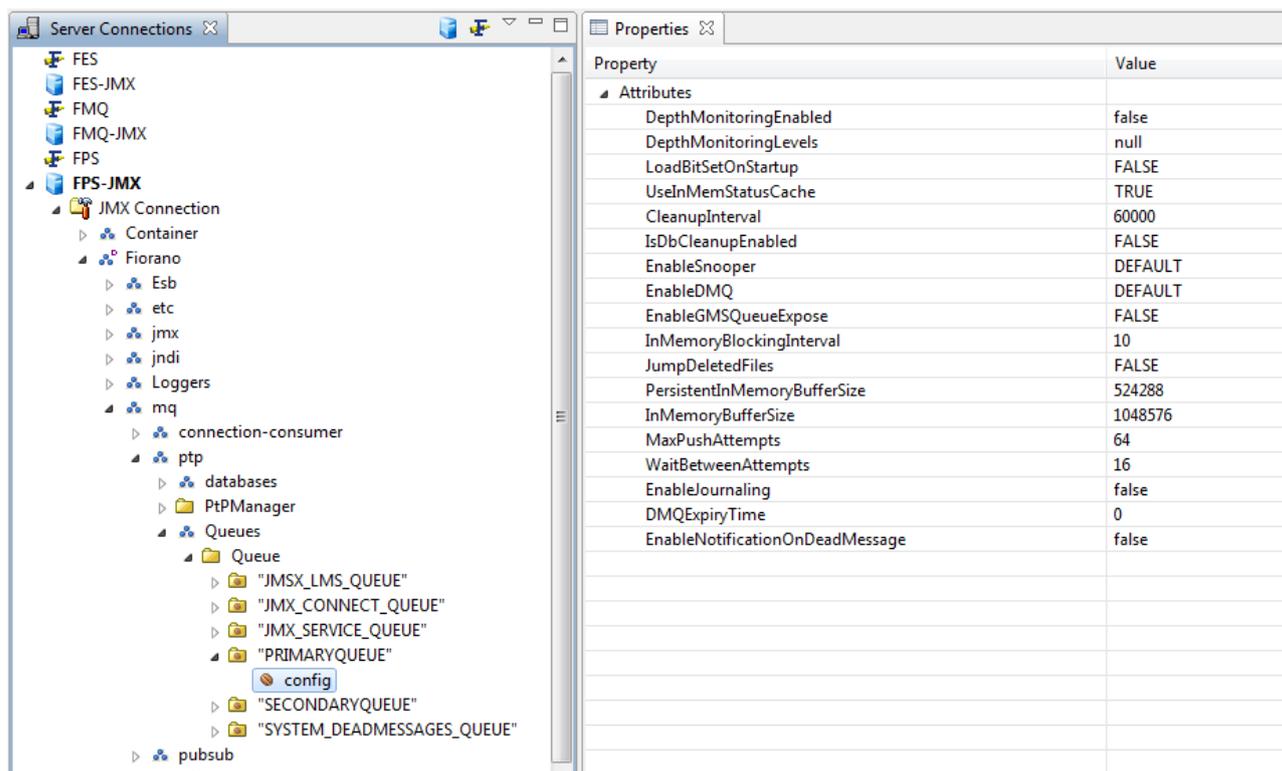


Figure 11: JMX Login node

Using the JMX connection a User can invoke JMX operations or change the Server properties at runtime.

1. Invoking JMX operations

To invoke an operation, right-click on an MBean and select View Operations. The View Operations option is present only if the MBean has one or more operations, as shown in Figure 12.

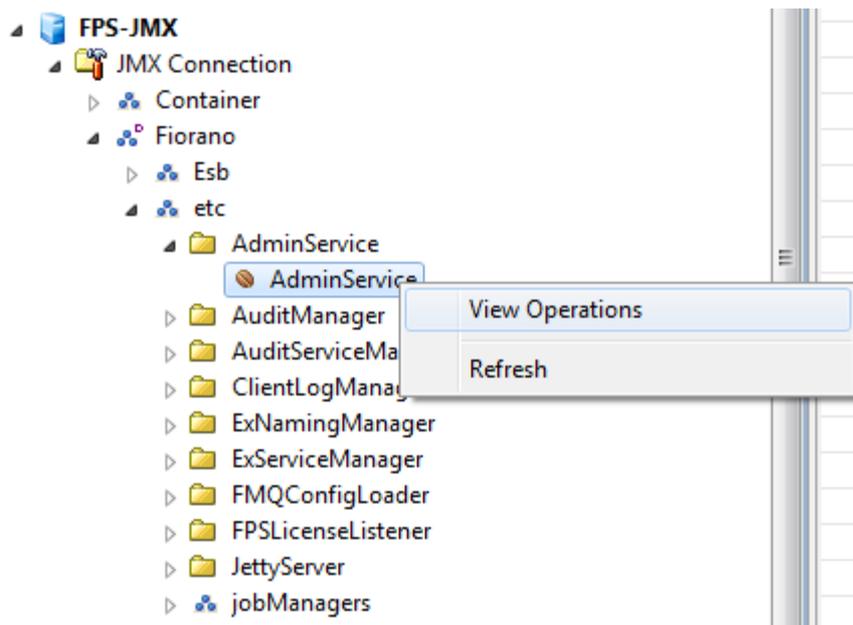


Figure 12: View JMX operations

A dialog box is displayed listing operations on the left and operation information on the right, as shown in Figure 13.

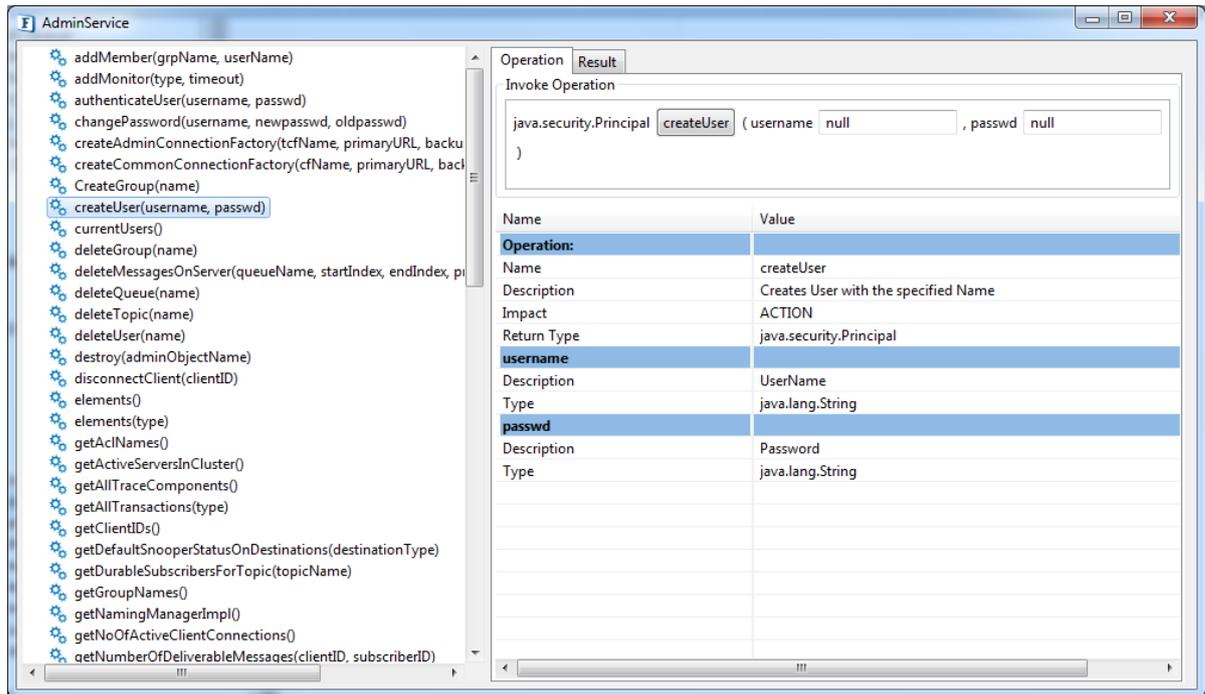


Figure 13: Operation information dialog

Each operation is shown as a function along with the return type, operation name and function parameters. The signature for this function [Semantic check please – what signature?] is (<Return type> <Operation Name> <Parameter><Parameter Value>).

Operations, parameters descriptions and types are shown in the dialog box. Add values for the various parameters and click the operation name button to invoke the operation, as shown in Figure 14.

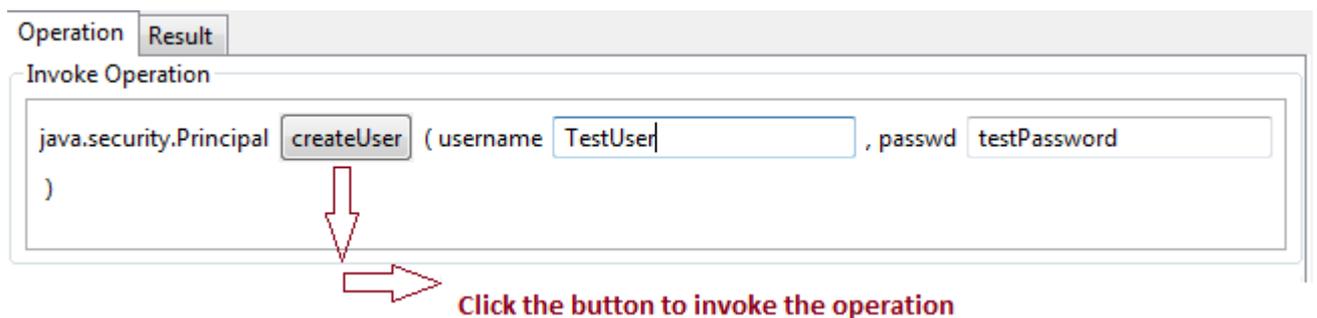


Figure 14: Operation signature

The result is displayed in the Result tab, as shown in Figure 15.

Operation		Result
Result of operation createUser (java.security.Principal)		
Item	Value	
ID	18	
name	TESTUSER	
type	0	

Figure 15: Result of operation invocation

Points to note:

If the result of invoking an operation is of a complex object consisting of java bean information, click the ellipsis button which will launches a java bean editor where all object information is displayed. An example of this is shown in Figure 16, below.

Operation		Result
Result of operation listCommonConnectionFactories (java.util.HashMap)		
Item	Value	
PRIMARYLCF	[name = PRIMARYLCF , description = LPC UnifiedConnectionFactory set u...	
PRIMARYCF	[name = PRIMARYCF , description = UnifiedConnectionFactory set up for <input type="checkbox"/>	

PRIMARYCF	
Item	Value
CBREnabled	null
CSPUpdateFreque...	null
HTTPProxyURL	null
LMSEnabled	true
SOCKSProxyURL	null
TCPBatchSize	null
XASocketTimeout	null
adminConnection...	100
autoDispatch	null
autoUpdationAllo...	true
backupConnectURLs	null
backupURLStrings	null
batchTimeoutInter...	null
batchingEnabled	true
clientProxyURL	null
compressionMana...	null

Figure 16: Result showing bean information

If an operation parameter is a complex known object, the java bean editor will be displayed where a User can add values required to invoke the operation. An example of this is shown in Figure 17, below.

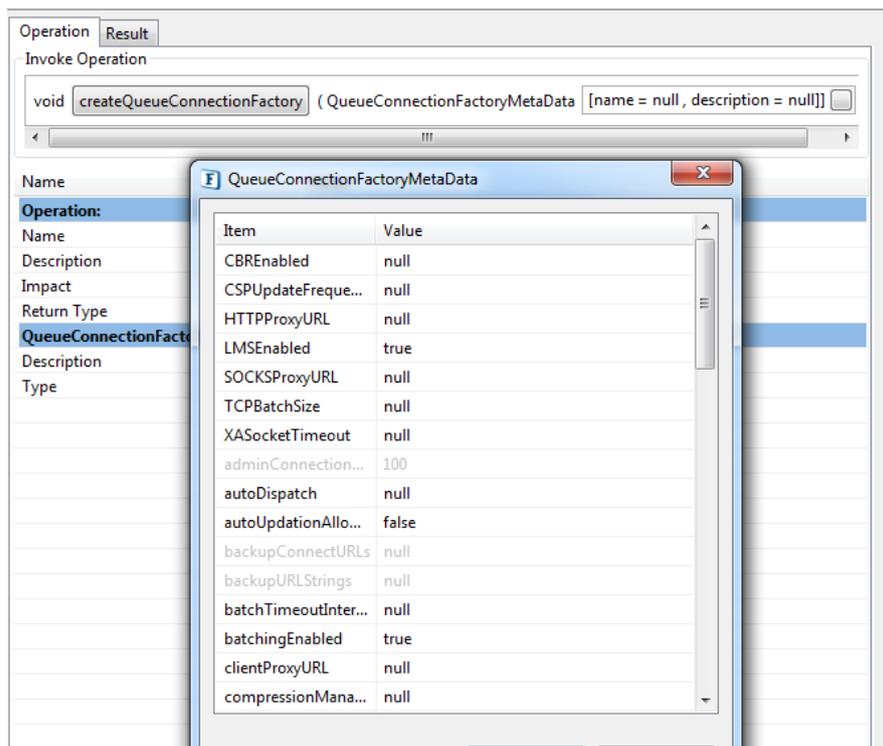


Figure 17: Input parameter with bean editor

If an operation parameter is a complex unknown object then the parameter and the operation invoke button are disabled and the User is not allowed to invoke the operation. An example of this is shown in Figure 18, below.

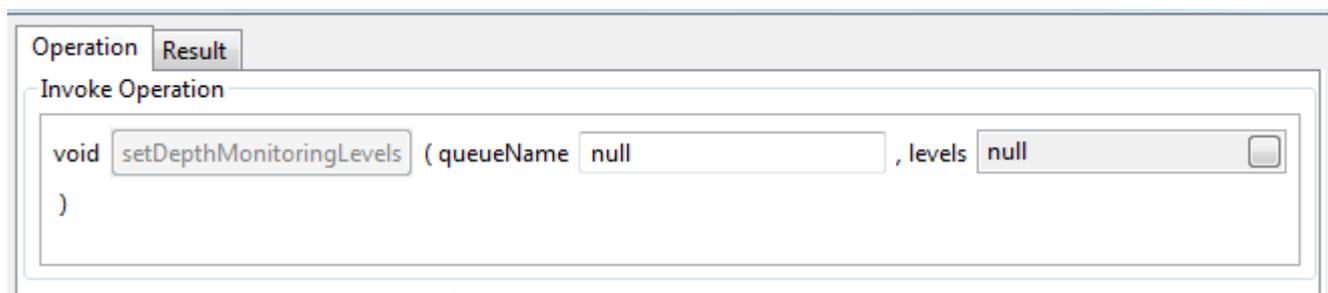


Figure 18: Operation parameter without bean editing support

By default, expert and hidden operations are not shown. To view these, change the preferences in the JMX Connections section under Fiorano preferences (Window → Preferences → Fiorano).

2. Changing Server Properties

Server properties can be changed at runtime. Select an MBean. In the properties view change the value of the property that requires modification, as shown in Figure 19.

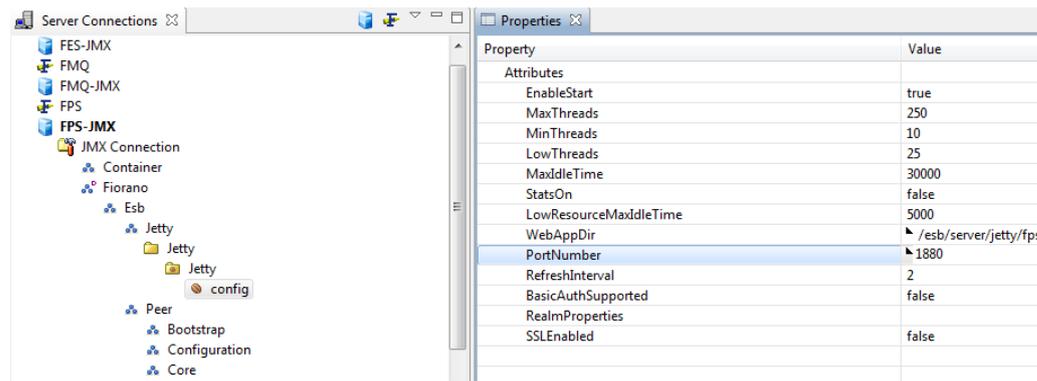


Figure 19: MBean attributes/properties

Once the property changes are made, right-click on the JMX Server and select Save Configurations to save the changes.

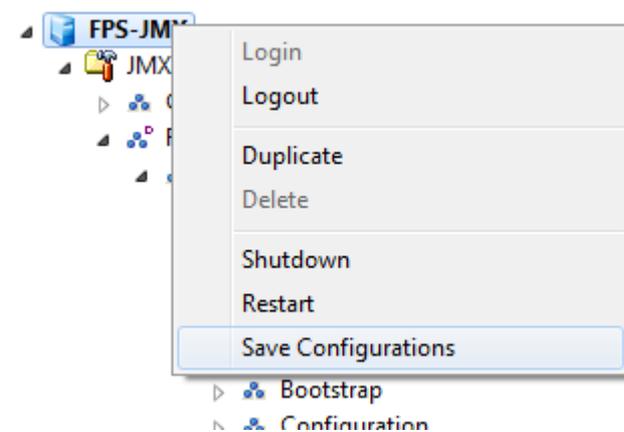


Figure 20: Saving configuration changes

Some properties are applied to Server that is running. However, some Server configurations (such as Server ports, memory settings and so on) are applied only after restarting the Server. When the Server requires restarting a message indicates this when the changes are made, as shown in Figure 21.

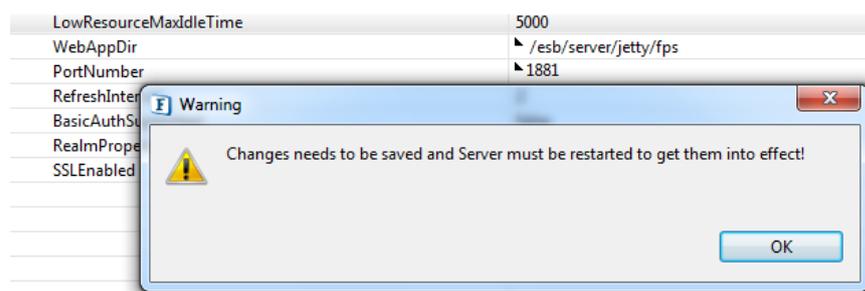


Figure 21: Dialog displaying server restart warning

Chapter 17: Profile Management

Profile Management Perspective in eStudio can be used for Offline configuration of Fiorano profiles. The Profile Management Perspective has the following components:

- Profile Manager view
- Profile Editor
- Profile Validations Messages view
- Error Log

17.1 Profile Management Perspective

To open the Profile Management Perspective, navigate to Window → Open Perspective → Other and select Profile Manager as shown in Figure 1.

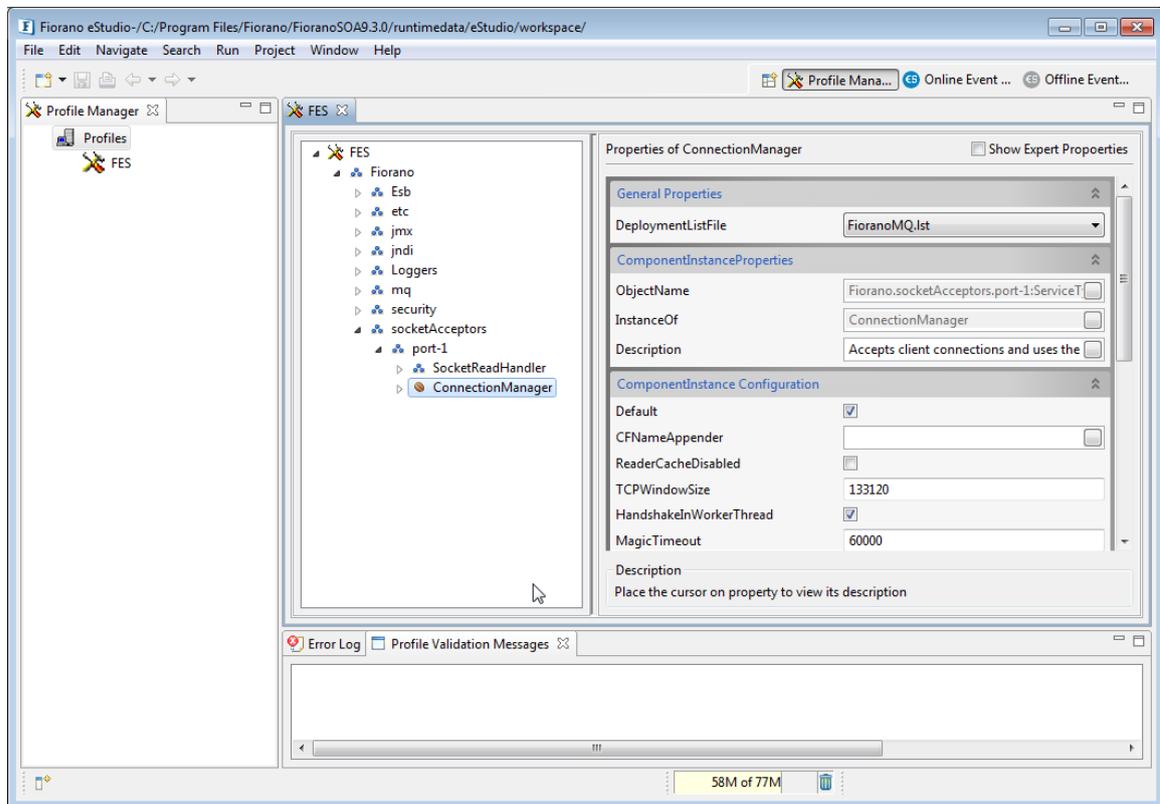


Figure 1: Profile Management Perspective

17.1.1 Profile Manager View

This view acts as an explorer for the profiles that are loaded onto the eStudio workspace. To load a profile onto the workspace, right-click on the Profiles node and go to the Load Profile sub-menu. By default all the MQ and ESB profiles are present in:

- \$FIORANO_HOME/fmq/profiles

- \$FIORANO_HOME/esb/server/profiles

The profiles present are displayed in subsequent menus. Choose a profile to load it onto the workspace, as shown in Figure 2.

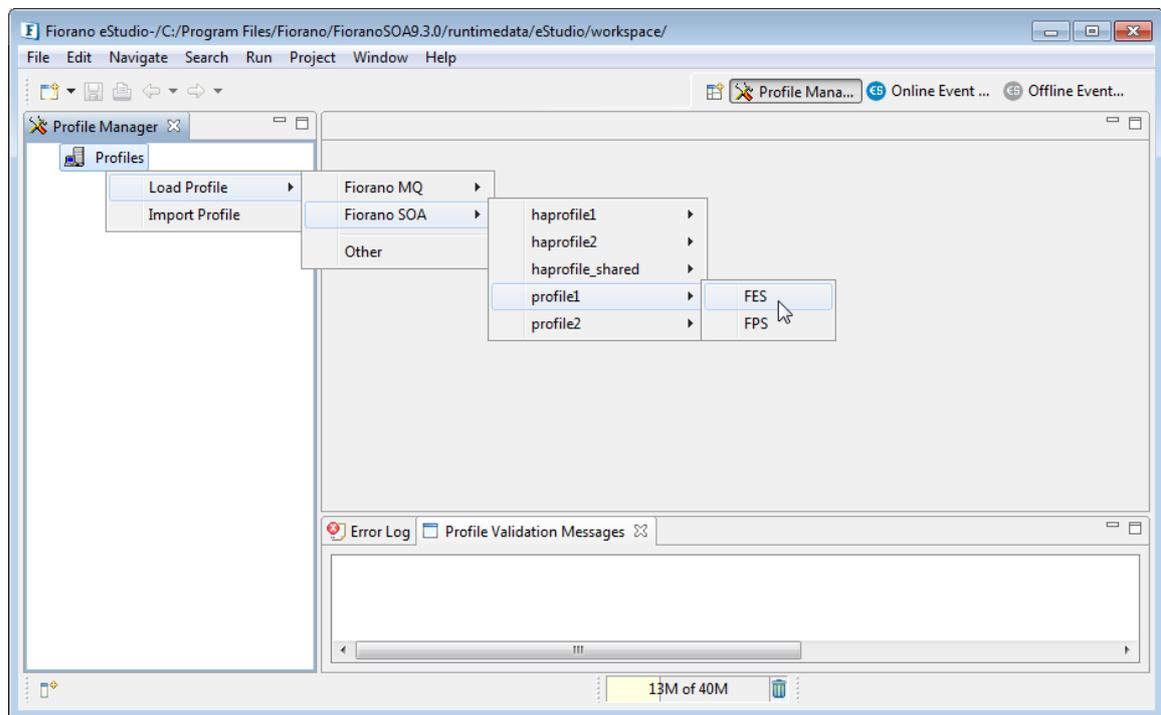


Figure 2: Loading a profile

To load profiles from locations other than the default profiles directories:

1. Click on the Other option present in the Load Profile menu.
2. Select the profile directory to load the profile onto. Profiles can also be imported from .zip files using the Import Profile option available in the right-click menu.
3. When a profile is loaded, a new child profile is added to the Profiles node with the profile name. The new profile is opened in a profile editor.
4. A profile once loaded can be opened by double-clicking the profile node.

Note: If the profile being loaded is already present in the workspace, the User will be presented with an option to reload the profile. If the User clicks Yes, the components of the profile will be reloaded.

17.1.2 Profile Editor

The Profile Editor is divided into two parts – the Component tree on the left and the Properties UI on the right.

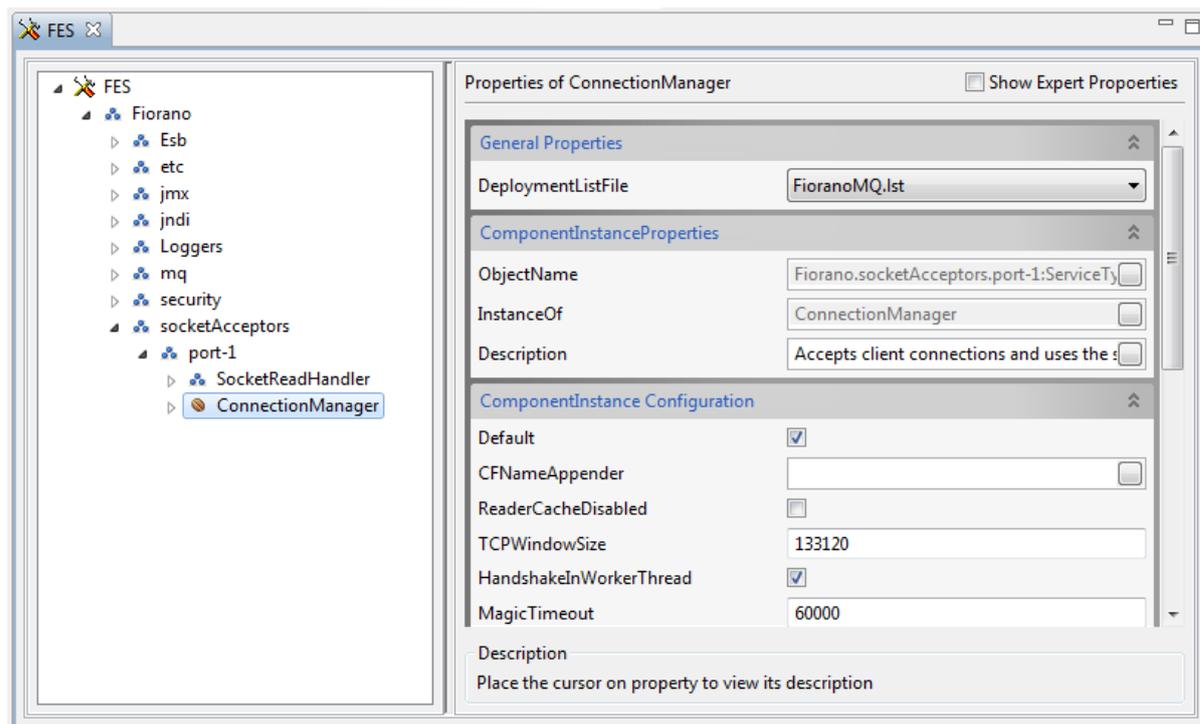


Figure 3: Profile Editor

17.1.2.1 Component Tree

When a profile is opened, it is presented in the form of a tree containing domain nodes () and component instance nodes (). The component instances that are part of the profile are specified in the Deployment list files (*.lst) present in the {PROFILE_DIR}\deploy directory where PROFILE_DIR is the root directory of the profile.

The component tree has a node for each Component Instance in the profile. If a component instance has any dependencies then a node called DependsOn will be added as a child node. This node lists all the dependencies of the component instance.

The component tree in eStudio Profile Editor also has a search feature that enables the user an easy access to any component instance. To find a component instance in the tree, press Ctrl+F or right-click on the profile's root node and select **Search**. In the text box opened at the bottom of the tree, enter the name or a part of the name of a component instance and click Enter to find the node.

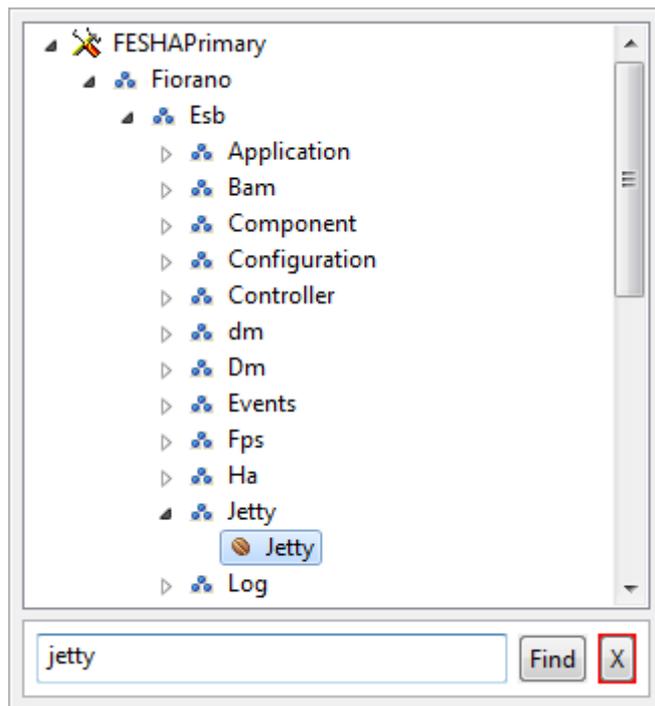


Figure: Search in Profile Editor's Component Tree

17.1.2.2 Properties UI

The Properties UI of a profile editor consists of property Name-Value pairs and a description area. It displays the properties of the node selected in the Component tree. Hovering the mouse over a property will display its description in the description area.

17.2 Using a Profile Editor to modify a Fiorano Server Profile

The profile editor can be used to modify a profile in the following ways:

17.2.1 Modify Component Instance Properties

When a component instance node is selected in the Component tree, the attributes of the MBean represented by the selected Component Instance Node are populated as Name-Value pairs in the Properties UI. The properties of a component instance are categorized into different groups.

17.2.1.1 General Properties

The General properties section of the Component Instance properties contains the property DeploymentListFile. Its value is the name of the .lst file from which the Component Instance is loaded. If the value is changed, upon saving the profile the service will be added to the .lst file specified by the User.

17.2.1.2 Component Instance Properties

This section contains the properties `ObjectName`, `InstanceOf` and `Description` which denote the Object Name of the MBean, Service type of the component instance and its description respectively.

The properties `ObjectName` and `InstanceOf` are non-editable as they are greyed out.

17.2.1.3 Component Instance Configuration

The properties of the MBean corresponding to the selected Component Instance Node are specified under this section, as shown in Figure 4.

- If the property is hidden, it is not shown in the properties UI.
- If the property is read-only, it is shown in a non-editable state.
- If the property is an expert property, it is hidden in the properties UI by default. A User may view and edit the values of expert properties by checking the Show Expert Properties check box at the top of the properties UI.

For all editable properties of a component instance, depending on the type of property, a property editor is provided to help the User change the property value. Please refer to Table 1, below, for details on Property Types and Property Editors:

Property Type	Property Editor
String	Simple text editor.
Boolean	Check box.
Number (Integer/Float/Double)	A single line text editor which accepts only numbers as input.
Value extracted from a set of Legal Values	A Combo box which allows the user the select one value from set of legal values.

Table 1: Property Editors for different Property types

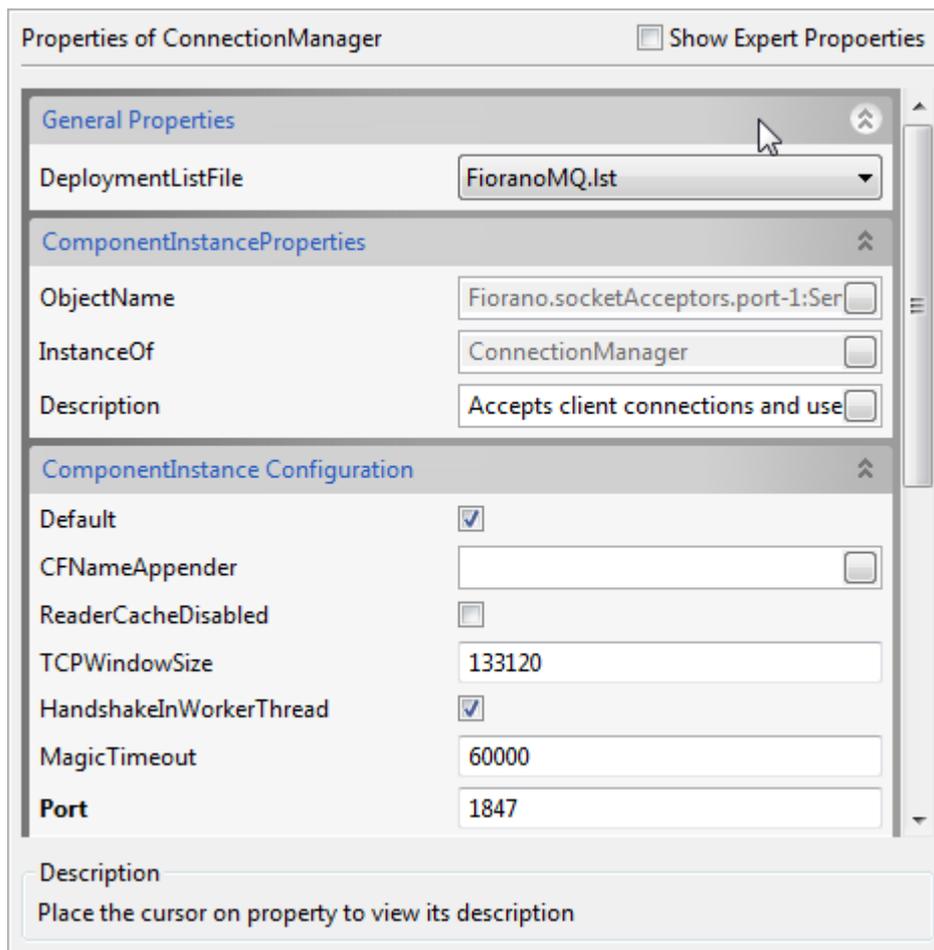


Figure 4: Properties UI

17.2.1.4 Component Instance Attributes

If the Component Instance contains attributes, these attributes are listed under the section Component Instance Attributes in the Properties UI, as shown in Figure 5. Editing these attributes is similar to editing the Component Instance Configuration properties.

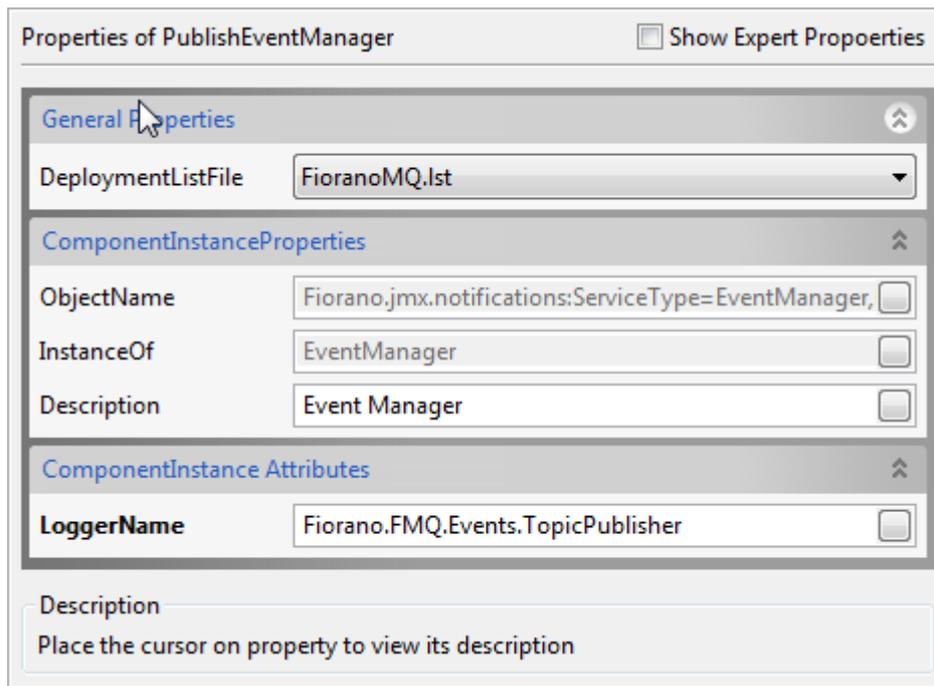


Figure 5: Component Instance Attributes

Note: When the value of a property is changed to a value other than the default value specified by the component MBean, the property name label will be displayed in Bold.

17.2.2 Add / Remove Component Instances

Domains/Component Instances can be added to profiles. To add a component instance, right-click on a domain and select the Add Components option. To add a new Domain right-click on a Domain or on profile node and select the Add Domain option.

1. Clicking the Add Components option opens the Add Components dialog box. The Add Components dialog box contains a table with two columns. The first column has a tree with all the components listed from which the required component can be selected. In the second column, Instance Count, the number of instances of the component that need to be added to the profile need to be specified. In the example below, as shown in Figure 6, only one instance of
2. Fiorano -> FioranoFW -> Services -> ConnectionManager is added to the domain Fiorano.socketAcceptors.port-2.
3. Select all the instances to be added and clicking OK adds the component instances to the component tree.

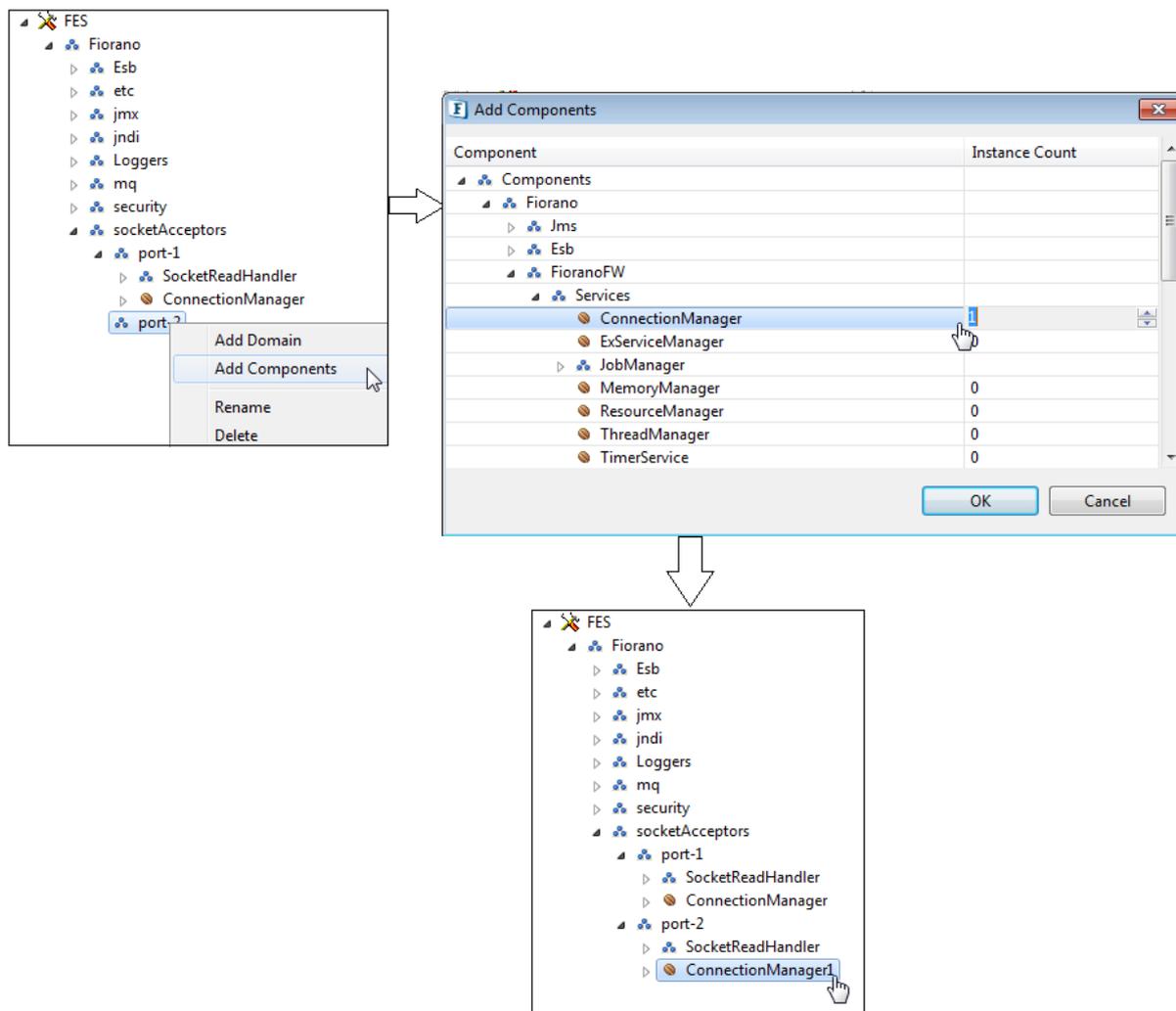


Figure 6: Adding a New Component Instance

The name of the component instances that have been added will be automatically generate automatically using <Component><instance_count>. A component instance can be renamed by choosing the Rename option using the right-click menu. The values of the properties are set to the default value. Component instance properties can be configured by providing relevant/valid values in the Properties UI.

Note:

- When a new component instance is added, unresolved dependencies of the component will be listed in the Profile Validation Messages view. Please refer to Section 17.2.3 for further information on resolving dependencies.
- If the component instance added has exclusive dependencies, a new instance of each exclusive dependency will be added to the profile within the same domain as the component instance. Please refer to Section 17.2.3 for further information on exclusive dependencies.
- A component instance can be deleted from a profile by selecting the Delete option using the right-click menu. Deleting a component instance removes any exclusive dependencies the component possesses from within the profile.

17.2.3 Resolving Component Dependencies

The dependencies for a component are listed as a child value of the DependsOn node of a component instance. These dependencies can be Required Dependencies, Optional Dependencies or Exclusive Dependencies.

Required Dependencies are those dependencies that are required to configure a component instance.

If a component in an open profile has unresolved required dependencies, error messages will be shown in the Profile Validation Messages view, as shown in Figure 7.

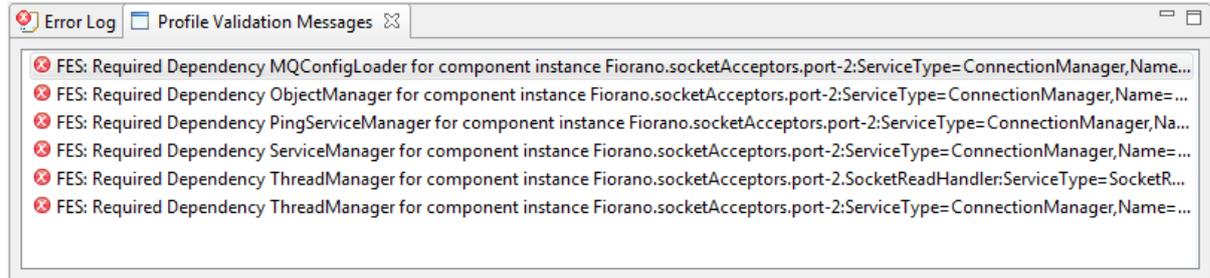


Figure 7: Profile Validation Messages

If a component has unresolved optional dependencies, warning will be logged in the Validation message.

Exclusive Dependencies are dependencies that should not be allowed to be used with more than one component instance within a profile.

Details about dependency instances are displayed in its properties, as shown in Figure 7.

17.2.3.1 Properties of a Component dependency

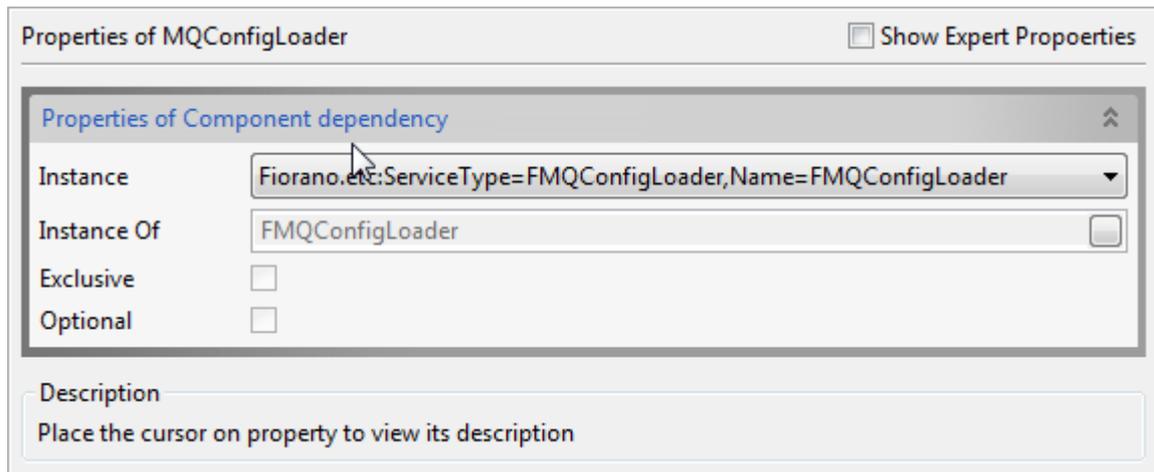


Figure 8: Properties of Component Dependency

Instance: This is the instance of a component dependency. A list of all available instances within the profile is provided in a drop-down list. A User is able to select the required instance.

Where there are exclusive dependencies, the fields are grayed out so they cannot be edited. This is because particular dependency is fixed for a given component instance.

- Instance Of: Specifies the service type of the dependency instance.
- Exclusive: Specifies if the dependency is exclusive to the component.
- Optional: Specifies if the dependency is optional to the component.

17.2.3.2 Resolving Component Dependencies

When a profile is opened, unresolved dependencies are listed in the Profile Validation Messages view either as errors or as warnings, as shown in Figure 9. To resolve a dependency, double click on its corresponding error message. This selects the dependency node in the tree, displaying its properties in the Properties UI. The User may now select an instance from the available instances displayed.

For example, in the scenario shown in Figure 9, the dependency ObjectManager is unresolved for the instance ConnectionManager1. To resolve this:

1. Click on the ObjectManger node.
2. In Properties, click on the combination next to the property Instance. A drop-down list with all the available instances is displayed.
3. Click an instance to select it as the dependency instance.

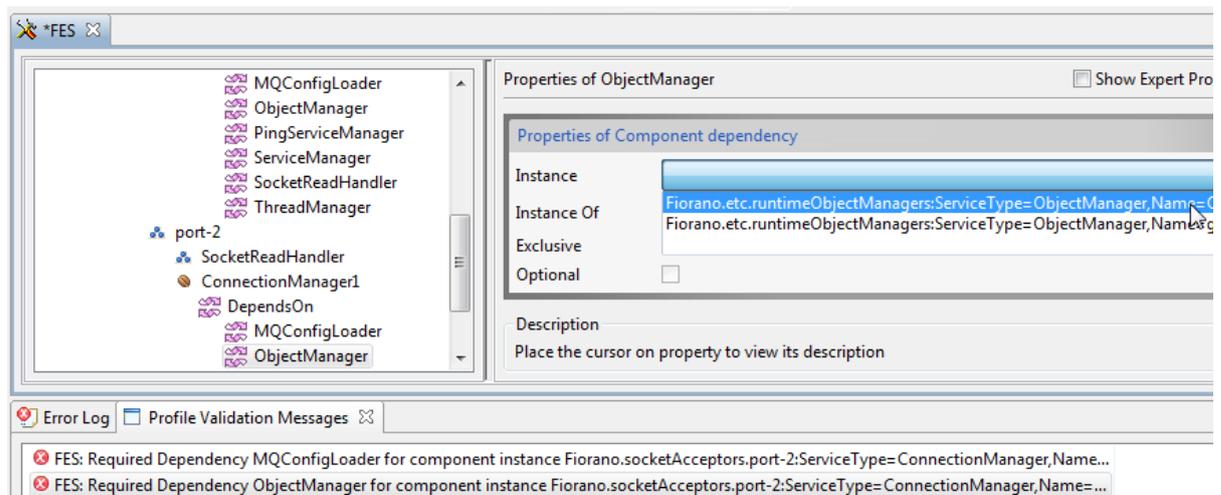


Figure 9: Resolving dependencies

If a new instance is to be created:

1. Right-click on the dependency node and select the Add New Instance option. A dialog box is displayed with all available domains.
2. By clicking OK, a new instance of the dependency is created within the selected domain. This dependency can now be resolved by selecting the new dependency instance and adding it to the Instance property of a component, as shown in Figure 10.
3. Once an instance is selected, the error message is removed from the Validation Messages view indicating that the dependency has been resolved.

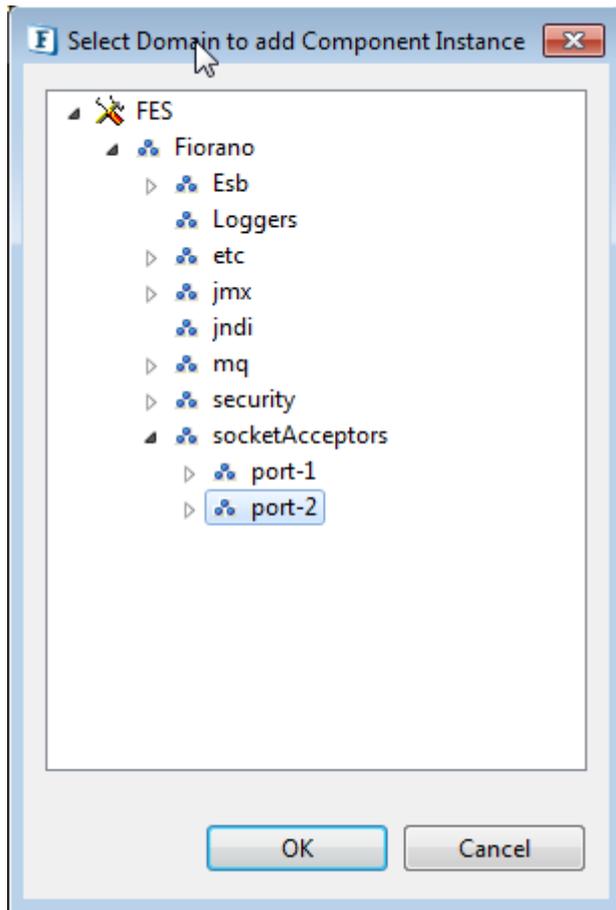


Figure 10: Selecting a domain for a new Dependency Instance

17.2.4 Add a new Variable Dependency Instance

If a component has variable dependencies, the User can add new dependencies to its instances. To add a new variable dependency:

1. Right-click on the DependsOn node of the component. A menu is displayed with all available variable dependencies.

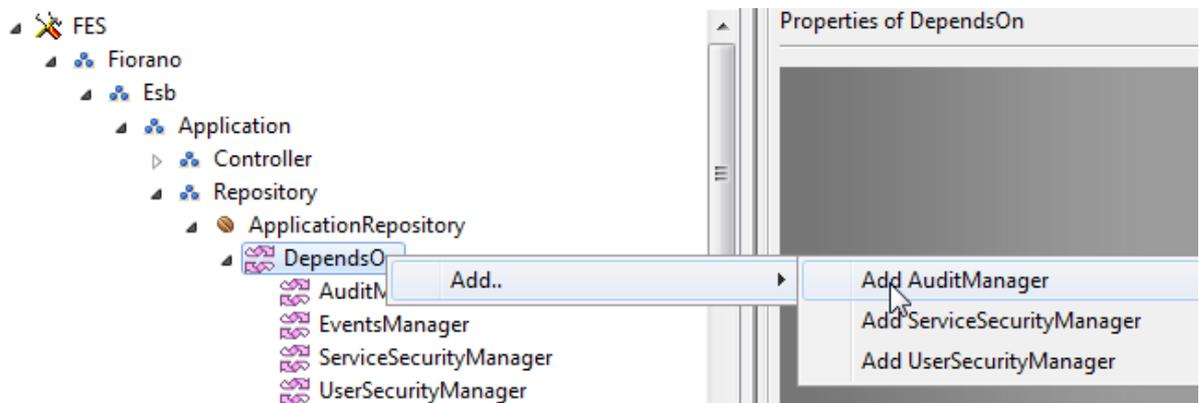


Figure 11: Choosing a Dependency type to add

2. Click on a dependency that needs to be added, as shown in Figure 11. A dialog box is displayed with a list of all components from which the component type of the new dependency can be chosen.
3. Click OK to add the new dependency, as shown in Figure 12.

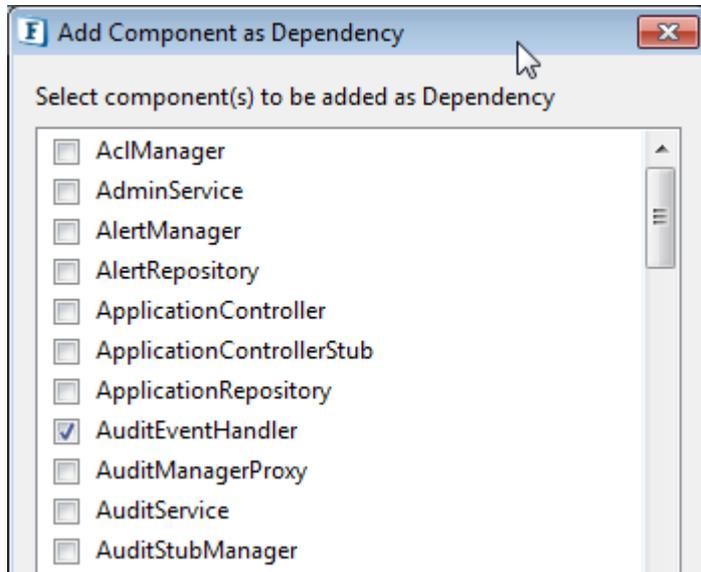


Figure 12: Choosing the component type of the Dependency

In the example below, a new dependency AuditEventHandler is added to the component instance Application Repository, as shown in Figure 13.

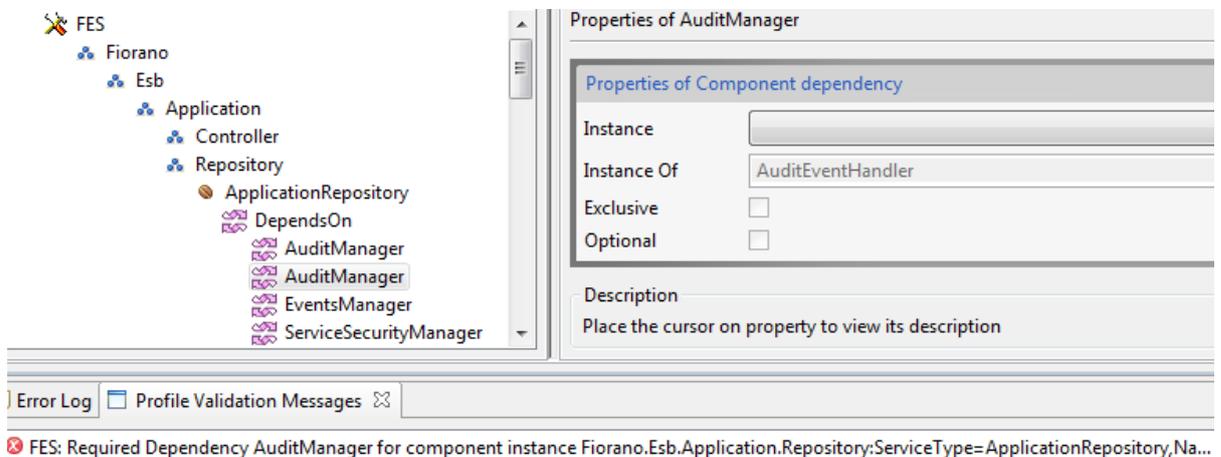


Figure 13: Newly added Variable Dependency Instance

17.2.5 Change Implementation of a Component Instance

Some components provide multiple implementations. The instances of such components have a property Implementation under the General Properties section. This lists all the available implementations. To change the implementation,

- Click on the combo next to the Implementation property and select the required implementation. This will update the ObjectName, properties and dependencies of the component instance with the new implementation.

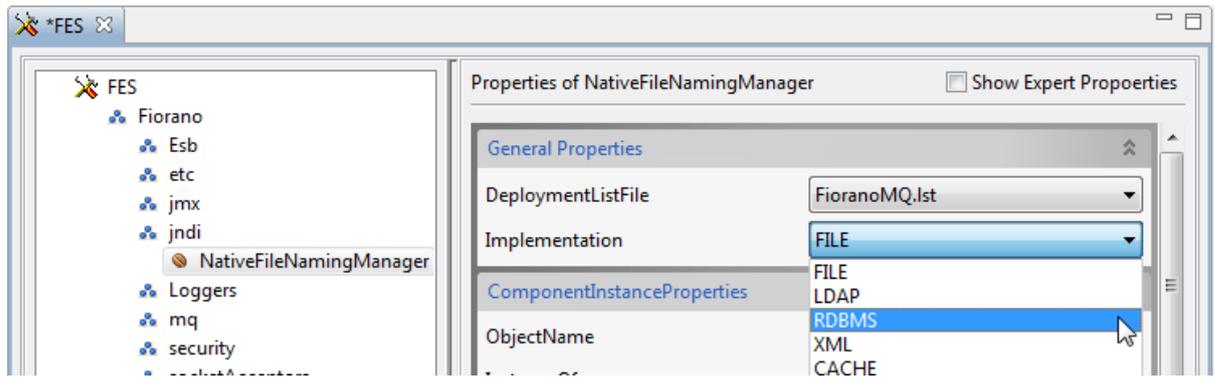


Figure 14: Modifying the implementation of a Component Instance

17.2.6 Adding attributes to a Component Instance

To add new attributes to a component instance:

1. Right-click on the instance node and click Add Attributes. A dialog box is displayed where the name and value of the attribute can be provided.
2. Specify attribute details and click OK to add the attribute to the component instance, as shown in Figure 15.

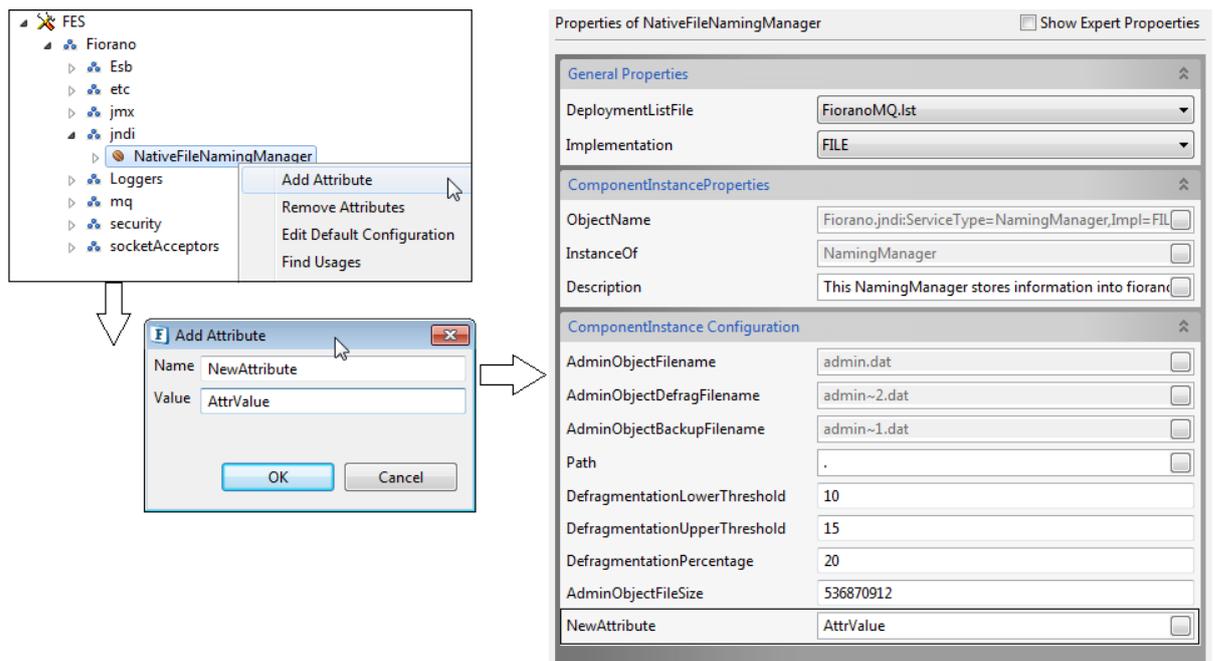


Figure 15: Adding an Attribute to a Component Instance

17.2.7 Remove custom attributes from a Component Instance

To remove User-defined attributes from a component:

1. Click on Remove Attributes from the right-click menu of the component instance node. A list of all the attributes added by the User is displayed.
2. Select the attributes to be removed and click OK to remove the attributes, as shown in Figure 16.

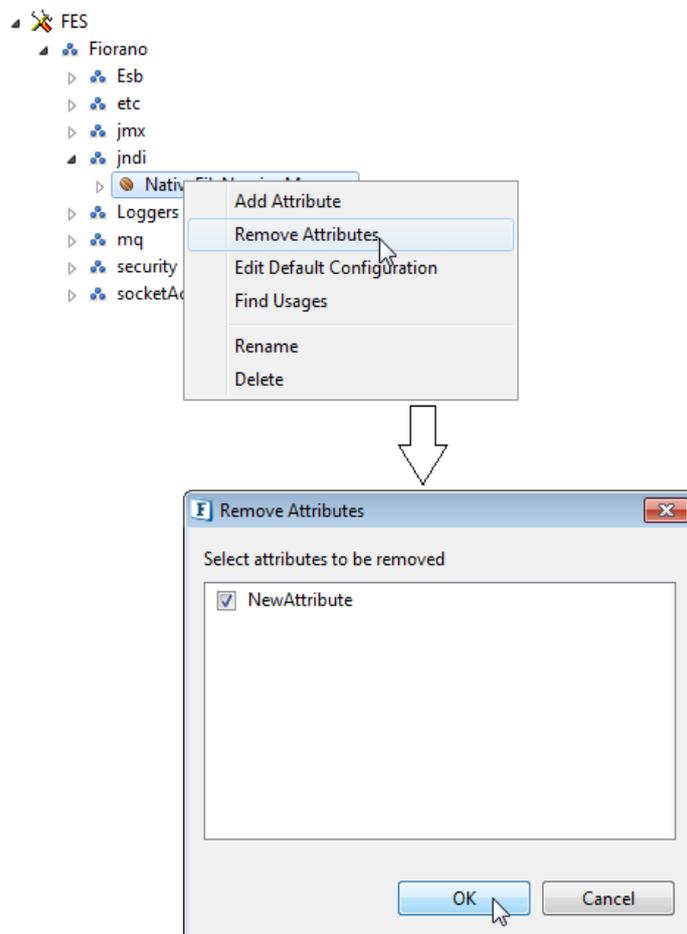


Figure 16: Removing user-defined attributes

17.2.8 Edit Default Configuration for a Component Instance

To edit the default configuration of a component instance:

1. Click Edit Default Configuration. A dialog box is displayed with all properties along with their default values.
2. Modify the default values of the properties using their respective editors and click OK to save the changes, as shown in Figure 17. The default values of the component will be modified and the new values will be used as default values at the time of adding new instances.

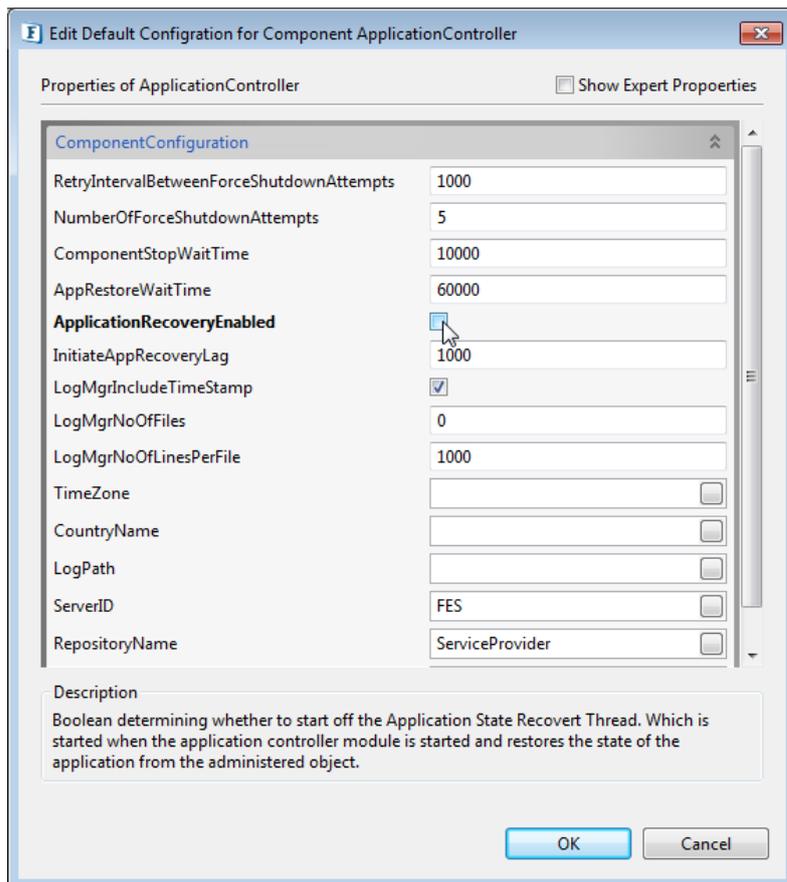


Figure 17: Edit Default Configuration Dialog

17.2.9 Modify a domain

Domains can be added, removed or renamed by using the right-click option.

1. Click Add Domain on the parent domain and give the name of the new domain to be added.
2. Click OK to add the new domain.

To remove a domain:

- Right-click on the domain and click Delete. The domain and all the component instances which are the children of the domain will be removed from the profile. If a component instance that is being removed has an exclusive dependency, the dependency instance will also be removed from the profile.

To rename a domain:

1. Right-click on the domain and click Rename. A dialog box is displayed where the user can enter the new domain name.
2. Click OK to change the name of the domain.

Note: When a domain is renamed, the ObjectNames of all the component instances which are the children of the domain will be modified in accordance with the new value.

17.2.10 Edit Profile Metadata

Click on the profile node in the component tree. The properties of the profile such as Profile Name, Description, Created On, Author are shown in the Properties UI. These values can be modified by specifying the new values in the text boxes provided, as shown in Figure 18.

Profile properties also include the DeployedLists property which is a comma separated list of all the .lst files (from which the components are loaded).

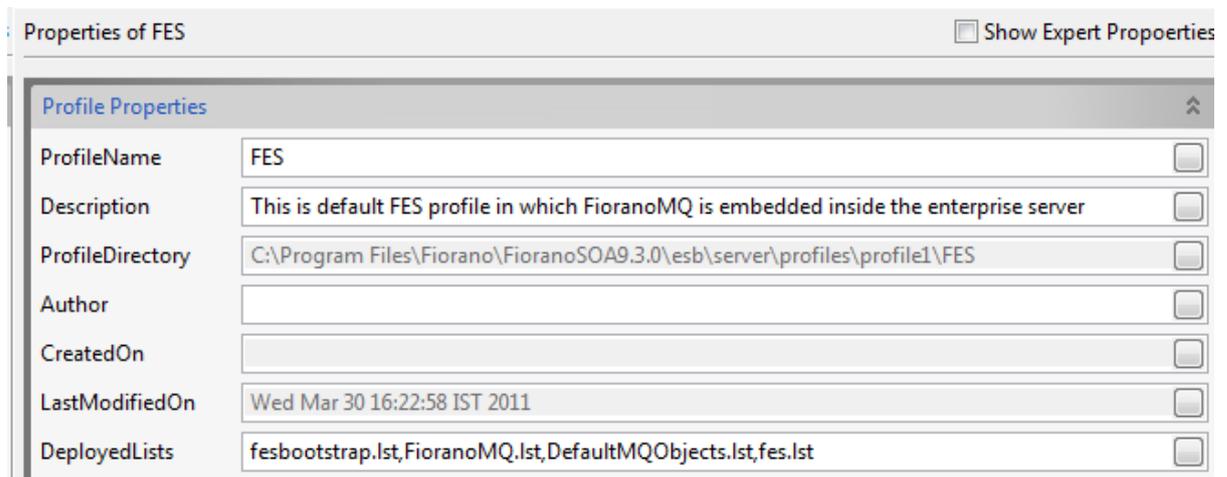


Figure 18: Profile Metadata

17.2.11 Locate a Dependency Instance in the Component Tree

To locate the actual instance of a component dependency:

- Right-click on the dependency node and select the Locate Instance option. This locates and selects the instance in the tree. The values of its properties are automatically updated in the properties UI.

17.2.12 Find Usages of a Component Instance

To find all the function of a particular component instance, right-click on the node and select the Find Usages option. The component tree will be filtered to show only those component instances which use the selected component and its dependencies, as shown in Figure 19.

The tree is returned to its previous state (displaying all the component instances) when the close button (marked by an X) is clicked.

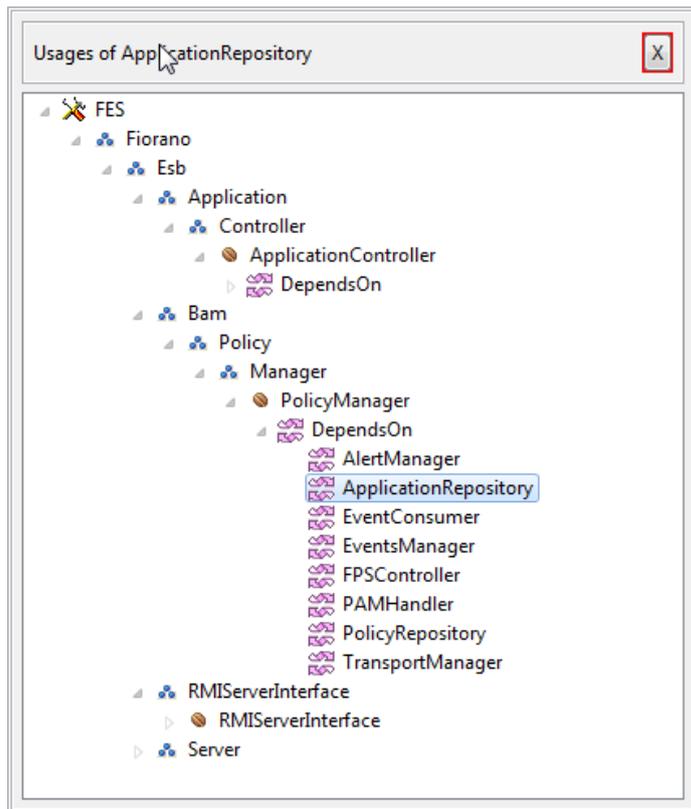


Figure 19: Filtered Component Tree showing the Usages of a Component Instance

17.3 Saving the Changes from Profile Editor

After the necessary actions to make changes are performed, click Save in the tool bar to save the profile. The profile can also be saved by choosing Save by right-clicking the menu of the profile node in the Profile Manager view.

If the profile saved has errors, the profile will not be saved and the User will be notified about the errors, as shown in Figure 20. These errors can be resolved by clicking on the error message in the Profile Validation Messages view.

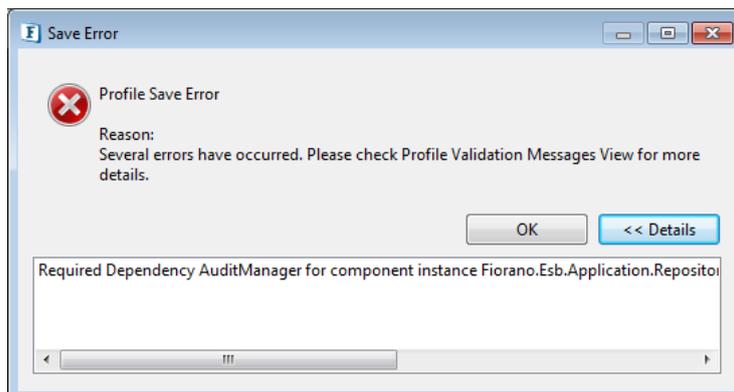


Figure 20: Error dialog shown when a profile with errors is saved

On successfully saving the profile, a message will be logged into the eStudio log view indicating that the profile has been saved.

17.4 Saving a Profile to a Different Location

To save a profile to a location other than the current location, use Save as.. from the tool bar. A dialog box is displayed with the target directory to be chosen. Once the target directory is chosen, the profile will be saved to the location specified.

This can also be done by using the File -> Save As option from the menu bar or Save as.. option by right-clicking the menu of the profile node.

17.5 Managing Profiles from the Profile Manager View

17.5.1 Import / Export profiles

Profiles can be exported as zip files. Profiles can be exported and imported by using the Export/Import options by the right-clicking the menu of a profile node.

To export a profile,

1. Right-click on the profile node in Profile Manager view and select the Export option.
2. Choose the destination directory to which the zip file is to be exported.

To import a profile,

1. Select the Import option on the Profiles node and choose the zip file and the destination directory to which the profile needs to be imported. A confirmation dialog box is displayed enquiring if the User wishes to load the imported profile.
2. On selecting the Yes option the imported profile is loaded.
3. On selecting the No option, no further action is performed.

17.5.2 Validate Profile

A profile can be validated by clicking Validate by using the right-click menu of the profile node. Errors and warnings, if any, will be shown in the Profile Validation Messages view.

17.5.3 Reload Profile

The components of a profile can be reloaded from its original directory location by using the Reload option by the right-clicking the menu of the profile node. When the profile is reloaded, the components of the profile are refreshed and the component tree will be re-created.

17.5.4 Close a Profile

An opened profile can be closed by clicking the Close option by the right-clicking the menu of the profile node. Closing a profile will delete the profile references from the workspace. The actual profile will remain unaffected. If the User wants to edit the profile again, the profile must be loaded into the workspace again.

17.5.5 Delete a Profile

A profile can be deleted from its location by clicking the Delete option by right-clicking the menu of the profile node. Deleting a profile deletes it from the Fiorano installation directory.

17.5.6 Points to note

- Modifications to a profile cannot be performed from the Profile Editor when the profile is running. All the configuration options will be disabled. The User will not be able to save the profile.
- Unless the profile is open, the actions Save, Save as, Export, Validate and Reload will be disabled and the right-clicking option the menu of the profile node in Profile Manager View will not be functional.

Chapter 18: Installing eStudio Dropins In Eclipse

The programs, below, must be installed before using eStudio as a drop-in:

- **Eclipse Version:** Eclipse 3.5 Stable Release 2 (recommended) or above.
- Ensure that the dropins EMF (version 2.5.0), GEF (3.5.2) and WTP (Web Tools Platform Version R-3.1.2) are installed in order to use eStudio as a drop-in. All the above dependencies can be installed via their respective update sites or via drop-in.

18.1 Installation

1. Copy the folders **features** and **plugins** from **\$FIORANO_HOME/eStudio** directory and paste to the **\$ECLIPSE_HOME/dropins/Fiorano/eclipse** directory. Create folders **Fiorano**, and **Eclipse** if these don't already exist.
2. Create a folder **licenses** in **\$ECLIPSE_HOME/dropins/Fiorano** directory and place the Fiorano license file(s) in it. The directory structure is shown in **Figure 1**.

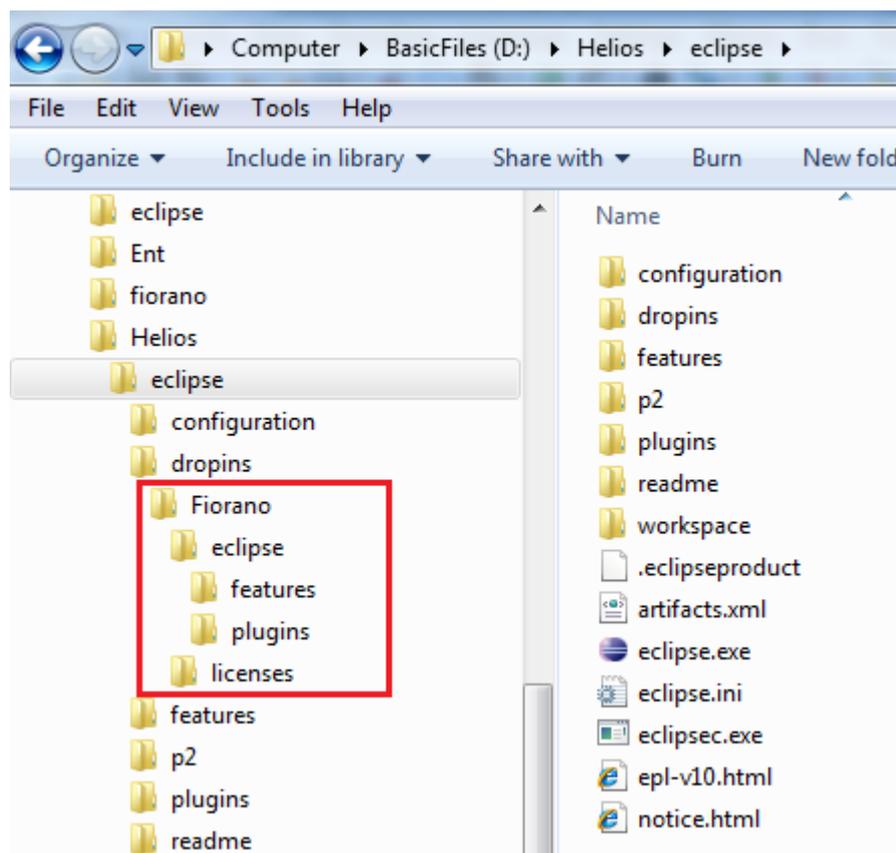


Figure 1: Fiorano dropins installed in eclipse

3. Open **ECLIPSE_HOME/eclipse.ini** file and add the properties given below to the end of the file:

- a. -DFIORANO_RMI_CALL_TIMEOUT=200000
- b. -DFIORANO_HOME=<Fiorano installation path>

Note: Additional properties may be required SSL [Expand SSL] is used for communication between the Servers. The properties that may need to be added are listed, below, in **Figure 2**.

```
-Djava.endorsed.dirs=D:/Fiorano/FioranoSOA9.3.0/esb/lib/endorsed
-DFIORANO_RMI_CALL_TIMEOUT=200000
-DFIORANO_HOME=D:/Fiorano/FioranoSOA9.3.0
-DFMQ_DIR=D:/Fiorano/FioranoSOA9.3.0/fmq
-Djavax.net.ssl.trustStore=D:/Fiorano/FioranoSOA9.3.0/fmq/profiles/FioranoMQ/certs/jssecacerts
-Djava.naming.factory.initial=org.apache.naming.java.javaURLContextFactory
-Djava.naming.factory.url.pkgs=org.apache.naming
```

Figure 2: Fiorano related properties within the eclipse.ini file.

4. Start eclipse with the **-clean** option from the command line. eStudio will be installed successfully. To verify installation, navigate to **Window → Open Perspective → Other**. Fiorano specific perspectives such as **Online Event Process Development, Offline Event Process Development, Profile Management, Connection Management** and **Mapper** should be visible.
5. Go to **Window → Preferences → General → Workspace → Local History** and change the value of the property **Maximum File Size (MB)** to 10. [Why? Explain.]
6. To use Fiorano specific Key Shortcuts, go to **Window->Preferences->General->Keys** and change the scheme to Fiorano.

18.2 Uninstallation

1. Delete the properties mentioned in step 2 of the Installation section from the eclipse.ini file.
2. Remove the Fiorano directory from the drop-in directory and restart eclipse with the **-clean** option. eStudio will be successfully uninstalled.

Chapter 19: Flat File 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 facilitate 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.

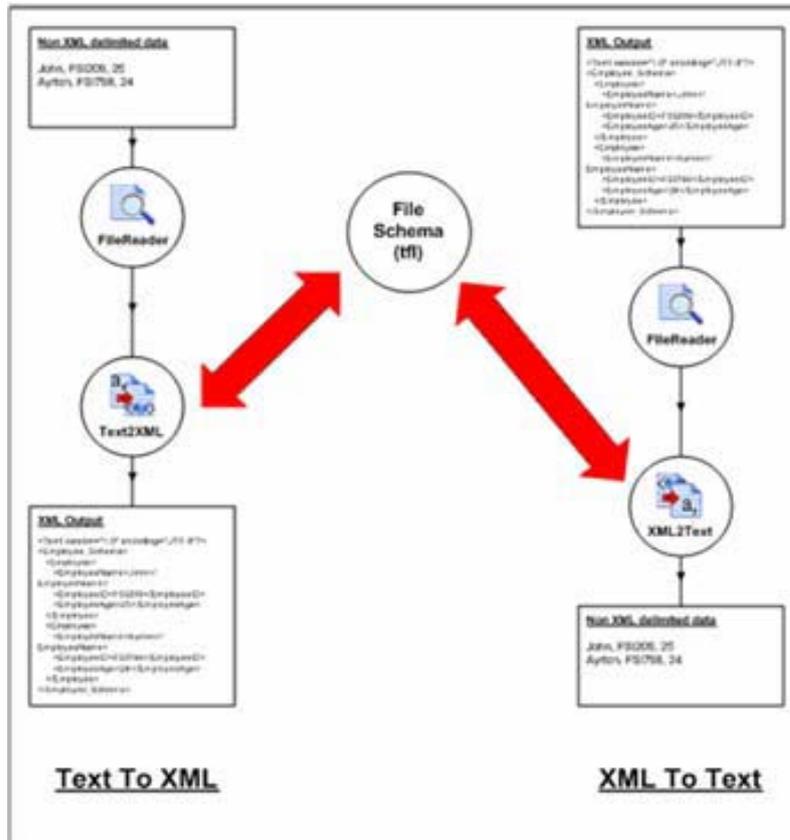


Figure 19.1: Using File Reader and Transformation components

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.

19.1 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.

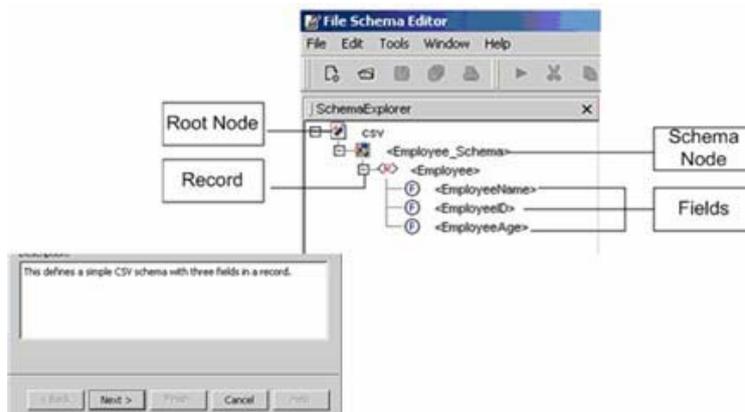


Figure 19.2: Structure of a tfl document

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.

19.2 Creating Flat File Projects

Flat file projects can be created using Flat File Schemas view in Fiorano Tools Perspective. In the eStudio menu bar navigate to Windows -> Open Perspective -> Other, select Fiorano Tools and click Ok. This takes to Fiorano Tools perspective where a view called Flat File Schemas can be seen.

In this section a sample Employee Schema will be created to illustrate the usage of Flat File Schema Editor. Let's assume a flat file containing all the Employee Records (Employee Name, Age, Address) with each employee record in a new line where individual fields are comma separated as shown below.

```
Mike, 30, University Ave
Harry, 25, Kensington Oval
Ayrton, 40, Thousand Oaks
```

Figure 19.3: Sample csv data

1. To define schema for this data, first create a new Flat File Project. To do this, right-click on **Flat File Schemas** Node and select **New File Schema Project** option.

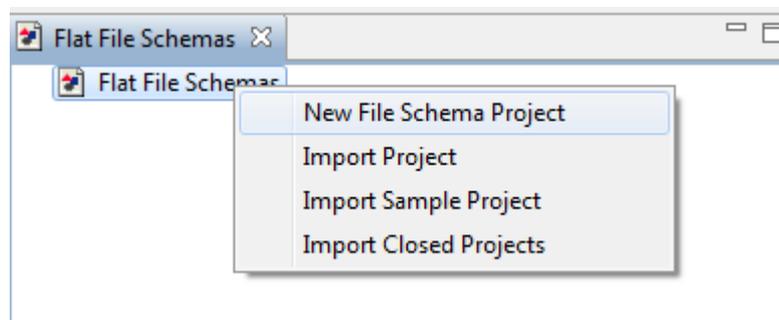


Figure 19.4: New File Schema project

2. This launches a wizard where project details can be configured. Provide name for Project and click Finish button. A new project will be created with project name as root node and "Empty Schema" as default schema node.

Note: Load from flat file option can also be used to create flat file schemas by loading the data from the flat file. This will be discussed in section 19.4.

3. Rename Empty Schema as Employee Schema by selecting the node and editing the Name property in properties view. Properties like **Record Delimiter** and **Default Field Delimiter** can be configured. Since the Employee records are separated by a new line provide the value for Record Delimiter as `\r\n`.

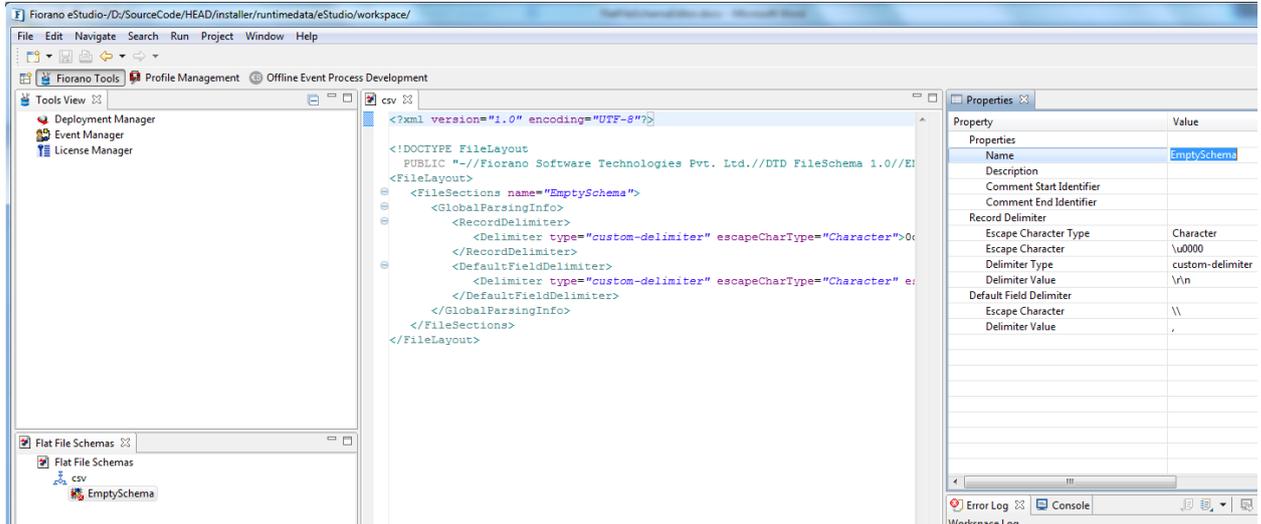


Figure 19.5: Flat file editor

4. Right-click on Employee Schema option and select **Add->Record** option. Give the record name as **Employee** and click OK.
Record will be added to the tree on the left hand side and the editor content is updated accordingly.
5. Since an employee record is comma separated, select the parsing type as Delimited and provide comma (,) as the delimiter value.

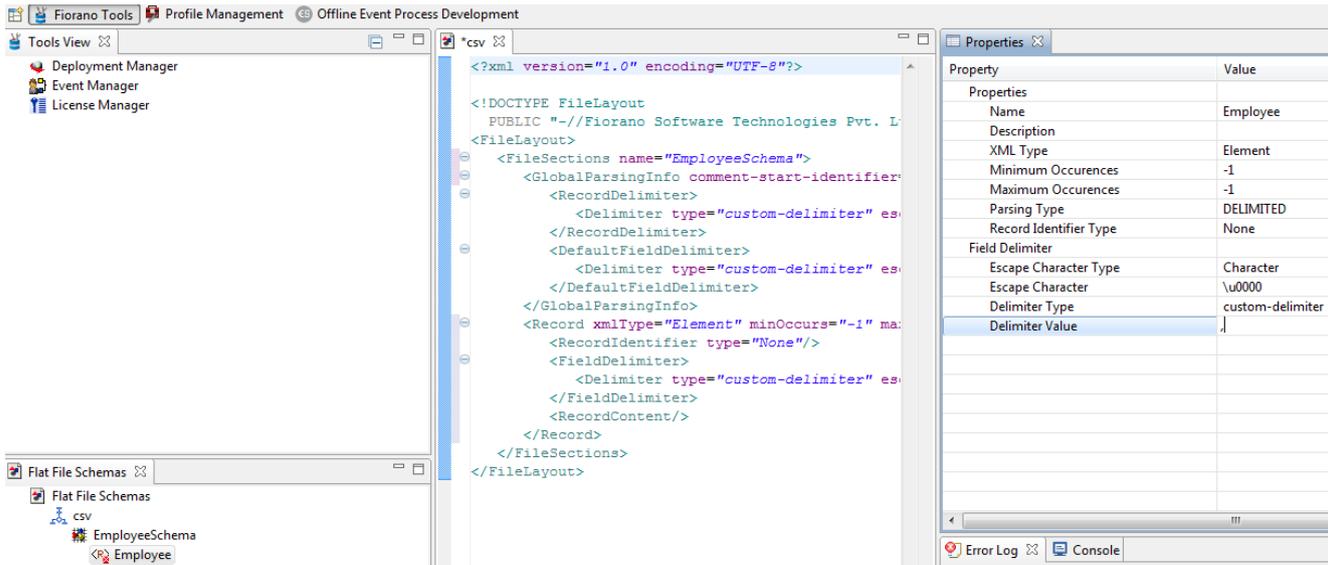


Figure 19.6: Configuring Flat File Schema elements

6. Right-click on Employee Record and select Add -> Fields option.

Fields can be added one by one or comma separated values can be provided to add multiple fields to a record. Provide Field Name as **Name, Age, Address** and click Ok.

Three fields Name, Age, Address are added and the schema is updated accordingly.

19.3 Testing Flat File Schema

1. Flat file schema generated can be tested in Test page. Click on the Test tab in the schema editor to open the Test page.

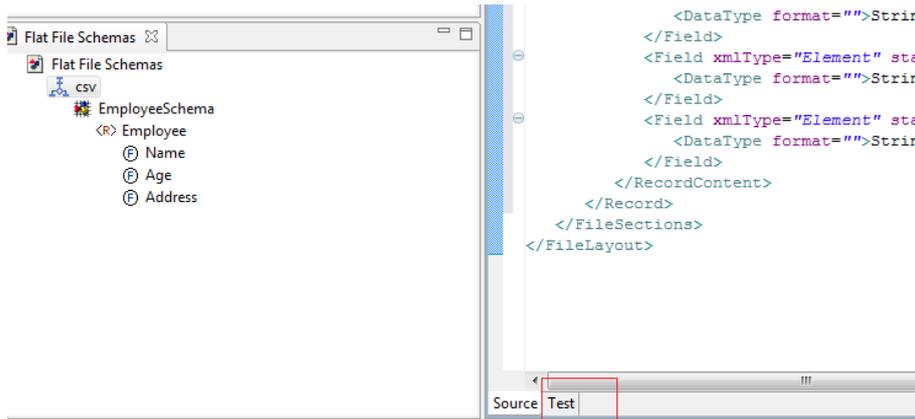


Figure 19.7: Flat file schema test

2. Sample data can be generated by clicking the **Generate Sample Flat Format** button or the sample can be pasted in the Flat Format section.

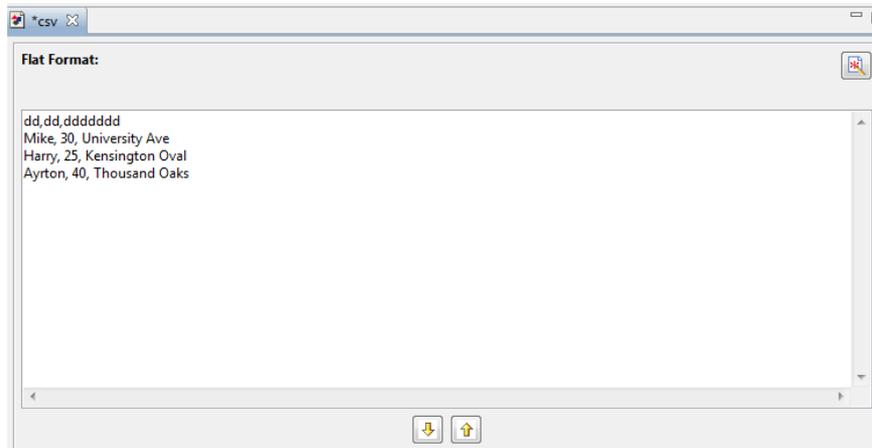


Figure 19.8: Flat format sample data

3. Click the Convert Flat Format to XML button. The flat format sample will be converted to XML and is displayed in the XML Format section.

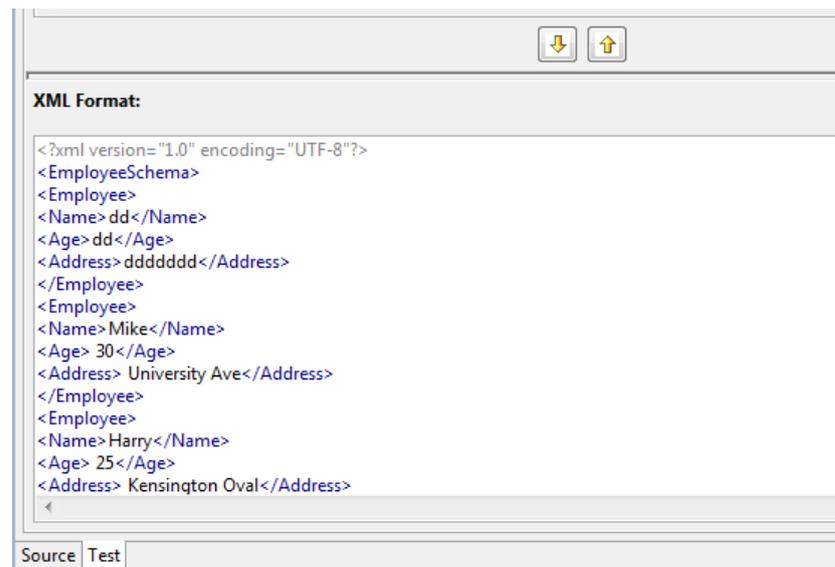


Figure 19.9: XML output

19.4 Generating Flat File Schema using sample data

Flat file schema can be generated by configuring flat file elements as mentioned in section 19.2. The same can be done by loading the tfl data which is detailed in this section.

The same example discussed in section 19.2 is used.

1. To define schema for this data, first create a new Flat File Project. To do this, right-click on **Flat File Schemas** Node and select **New File Schema Project** option.

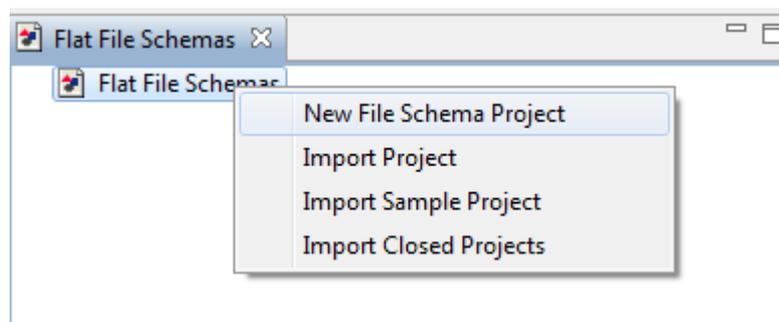


Figure 19.10: New File Schema project

This launches a wizard where project details can be configured.

2. Provide the project name and select **Load From Flat File** option. Select the flat file and click **Next**.
3. Provide the Schema Node name, record delimiters values. Since the Employee records are separated by a new line provide the value for Record Delimiter as **\r\n**. Click **Load Data** button to load the data from flat file. The data is loaded and records are displayed based on the delimiter value.

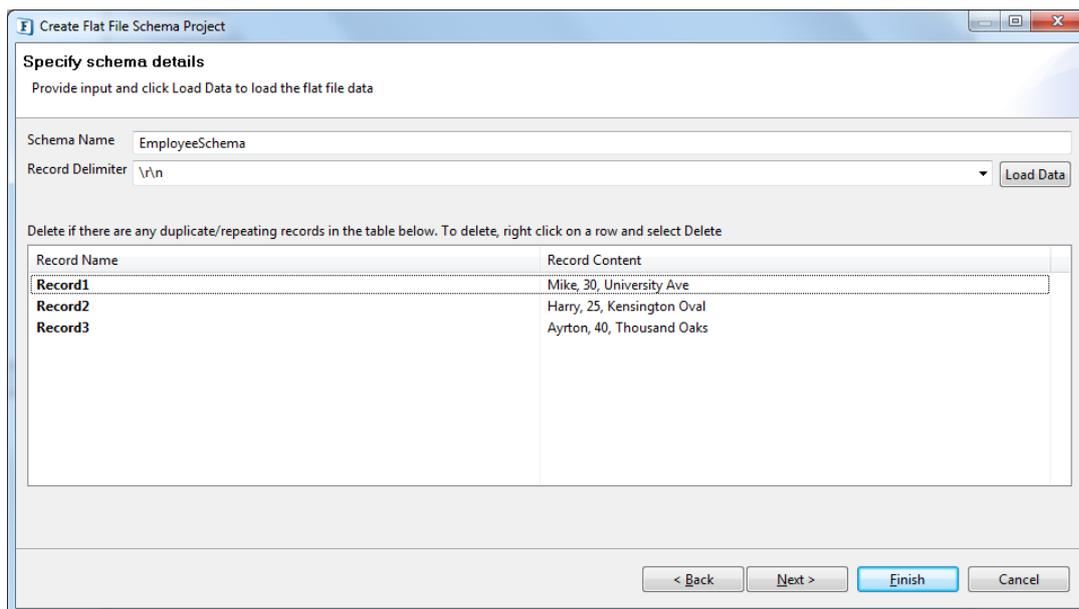


Figure 19.11: Schema configuration

4. Since all the data corresponds to individual employee records, duplicate rows can be removed. To remove a row, right-click and select **Delete**. Duplicate rows Record2 and Record3 can be removed.
5. Rename **Record1** as **Employee** and click Next button.

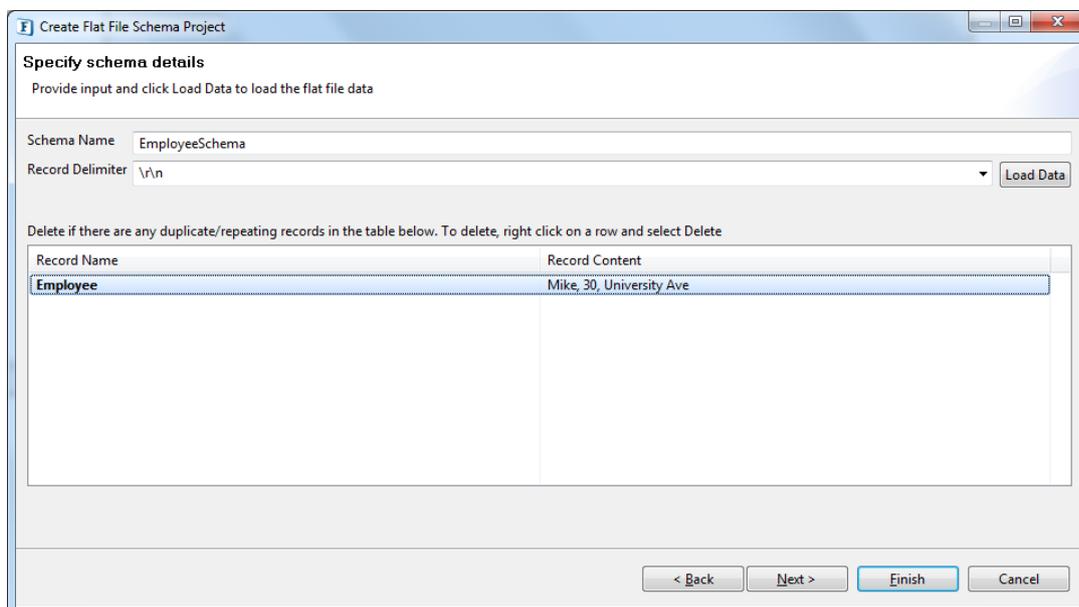


Figure 19.12: Configuring Records

6. The child elements can be configured in the **Schema Configuration** page. Select Employee node in the left hand tree viewer. The details of the node are displayed.
7. Provide comma (,) as the Field delimiter value.

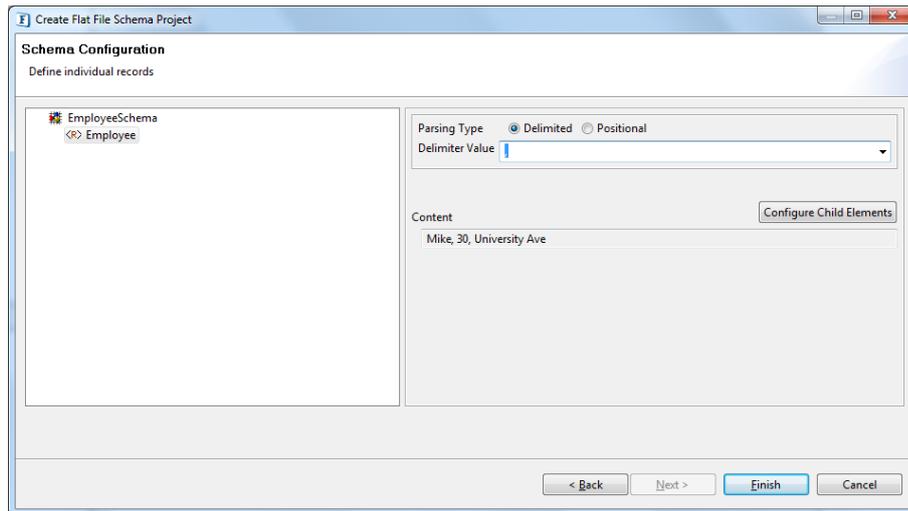


Figure 19.13: Configuring Record child elements

8. Click the **Configure Child Elements** button. The data is parsed using the child delimiter value and is displayed in a table. The element Name, type and data type can be chosen in this page. Provide the details and click Ok.

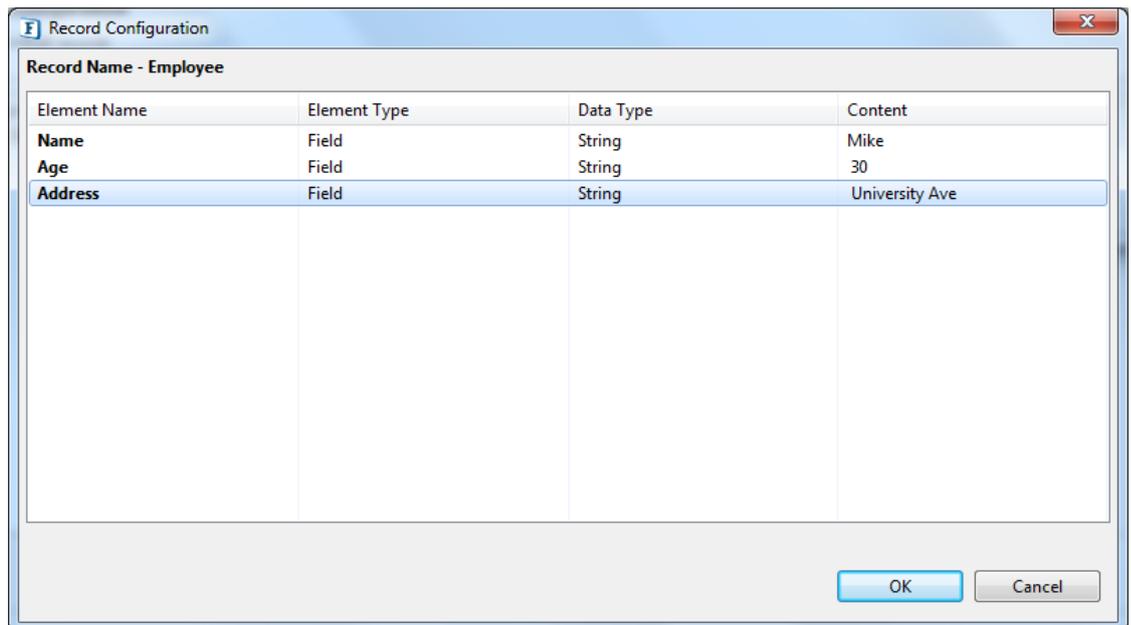


Figure 19.14: Defining Record child elements

9. The individual Fields are generated. Details of each node can be seen by selecting the node on the left hand side tree viewer. Click Finish to finish the configuration.

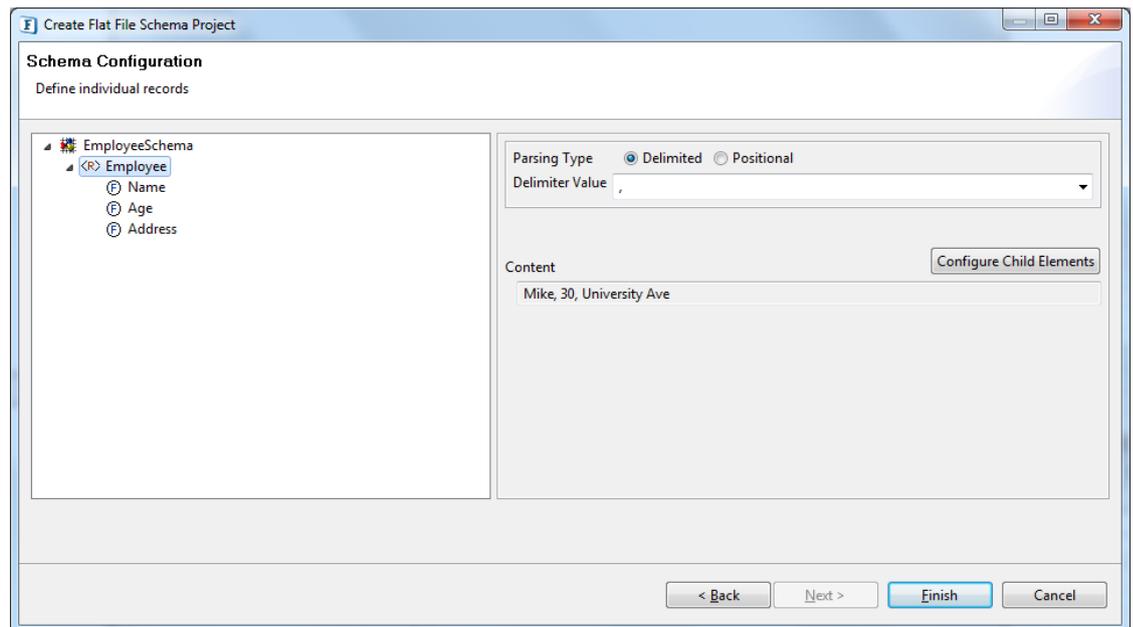


Figure 19.15: Employee Schema configuration

10. The schema is generated and is shown in the editor. This can be tested as described in section 19.3.

19.5 Sample 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:

1. Delimited File Schema samples
 - CSV File Schema
 - Nested CSV File Schema
2. Positional File Schema samples
 - Positional File Schema
 - Nested Positional File Schema
 - Positional in Delimited File schema

To import the prebuilt schema samples right-click on **Flat File Schemas** node and select **Import Sample Project** option.

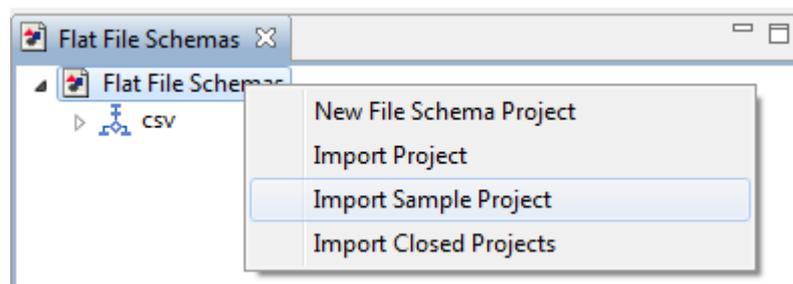


Figure 19.16: Import Sample Project

A dialog is launched listing all the available samples. Select a sample and click ok to load it in the editor.

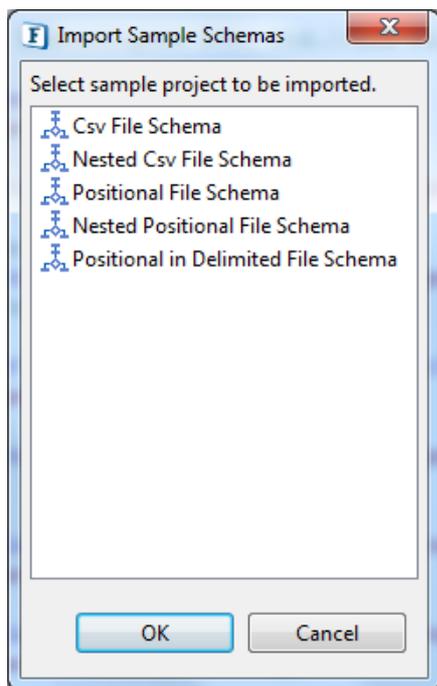


Figure 19.17: Select sample projects

19.6 Points to note

- Records can be positional or delimited. A Delimited record can contain a positional record as a child but a positional record cannot contain a Delimited record as a child.

Delimiters have to be provided for delimited records where as Start and End Positions have to be provided for Positional records.
- The Filed properties change based on the parent record parsing types (Delimited/Positional)
- Whenever there's any error in the generated schema, an error badge is shown on the corresponding element indicating the error. Place the cursor to see the error message.

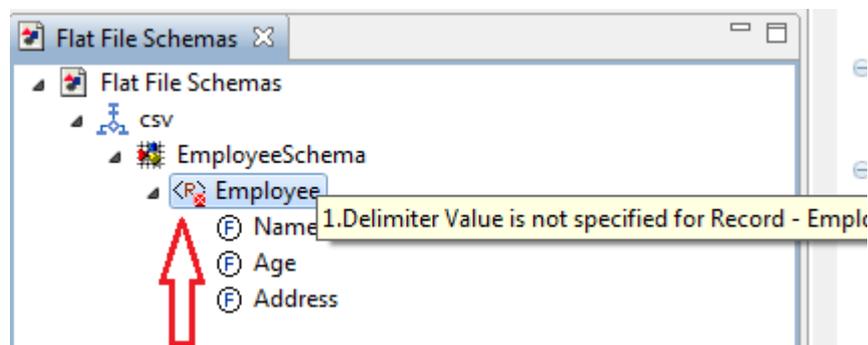


Figure 19.18: Schema Errors

- The order of Fields or Records can be changed. To change the order, select the parent element and select **Change Order** option. A dialog is displayed where the order can be altered.

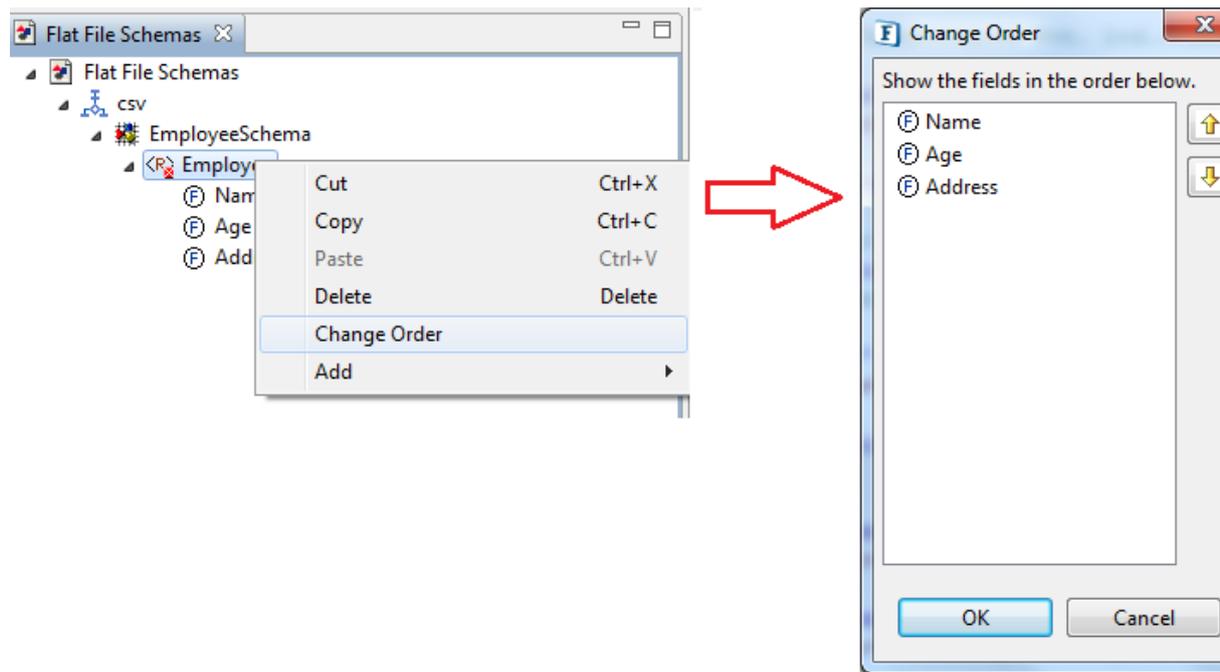


Figure 19.19: Change elements order

5. A flat file schema project can be exported using the **Export** option available on the context menu of the project. Similarly a project can be imported by selecting **Import Project** option available on Flat File Schemas context menu.

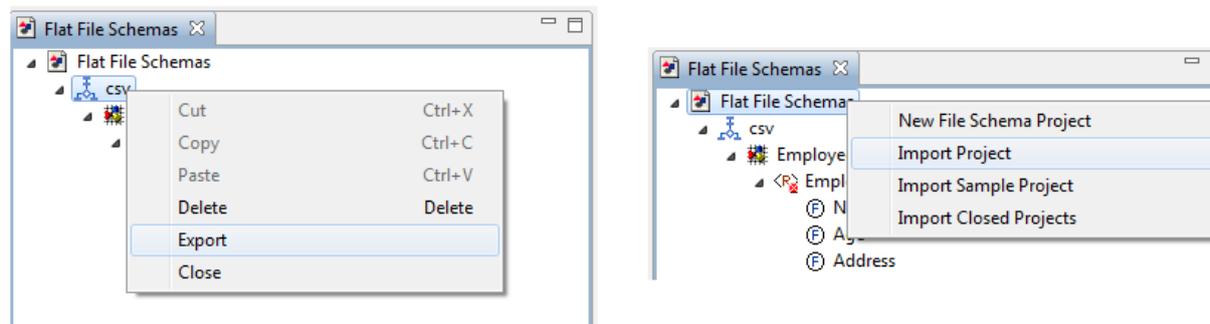


Figure 19.20: Import/Export project

6. We can close a project and load them later by using **Open Closed Projects** option on Flat File Schemas Node.

19.7 Flat File Element Properties

The properties associated with flat file nodes are described in this section.

19.7.1 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.

<p>Escape Character Type</p>	<p>You can choose the escape character type from the following values:</p> <p>Default Escape Character - Indicates that the escape character is the value of the Default Escape Character property which is defined for the schema instance.</p> <p>Character - Allows the user to designate an escape character value. If you select Character, you must also specify an escape value.</p> <p>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.</p> <p>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:</p> <p>Fiorano,Software,USA</p> <p>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:</p> <p>Fiorano\,Software,USA</p> <p>TSE interprets the comma after the backslash as field data rather than a delimiter character.</p>
<p>Escape Character</p>	<p>This is the escape character which is to be used as the field delimiter.</p>
<p>Delimiter Value</p>	<p>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.</p>
<p>Escape Character Type</p>	<p>You can choose the escape character type from the following values:</p> <p>Default Escape Character Indicates that the escape character is the value of the Default Escape Character property which is defined for the schema instance.</p> <p>CharacterAllows the user to designate an escape character value. If you select Character, you must also specify an escape value.</p> <p>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.</p> <p>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:</p> <p>Fiorano,Software,USA</p> <p>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:</p> <p>Fiorano\,Software,USA</p> <p>TSE interprets the comma after the backslash as field data rather than a delimiter character.</p>

Delimiter Type	This is the field delimiter of this file schema. The delimiter can be multiple characters.
----------------	--

19.7.2 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.

Following are some basic rules pertaining to records.

1. Every new record, which you create, is inserted as a descendant of the record that you selected.
2. 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.
3. 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) 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 occur 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	<p>Type of the Identifier to be used for identifying a record.</p> <p>You can choose the Record Identifier from the following values:</p> <p>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.</p> <p>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.</p> <p>None Record data is parsed against the record definition irrespective of the fact that the data satisfies the complete record definition or not.</p>
Name	<p>The name of the record. The name of the node should be a valid XML name. You cannot provide an existing record the same name as an existing record. Sibling record cannot have the same name.</p>
Description	<p>The description of the record.</p>
Escape Character Type	<p>You can choose the escape character type from the following values:</p> <p>Default Escape Character - Indicates that the escape character is the value of the Default Escape Character property which is defined for the schema instance.</p> <p>Character - Allows the user to designate an escape character value. If you select Character, you must also specify an escape value.</p> <p>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.</p> <p>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:</p> <p>Fiorano,Software,USA</p> <p>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:</p> <p>Fiorano\,Software,USA</p> <p>TSE interprets the comma after the backslash as field data rather than a delimiter character.</p>
Delimiter Value	<p>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.</p>
Delimiter Type	<p>This is the field delimiter of this file schema. The delimiter can be multiple characters.</p>
Escape Character	<p>This is the escape character which is to be used as the field delimiter.</p>

19.7.3 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 popup 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.

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 field. This value can either be Element (default), Attribute or None. If None is selected, then the field 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 <code>java.text.DecimalFormat</code> . For Date data type, data format can be defined based on the syntax rules of <code>java.text.SimpleDateFormat</code> .
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 its value is the same as the default value. Can be used to set Headers and Column Names.
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	<p>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.</p> <p>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:</p> <p>Fiorano,Software,USA</p> <p>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:</p> <p>Fiorano, Software, USA</p> <p>The comma between the double quotation marks is interpreted by TSE Parser to be field data rather than a delimiter value.</p> <p>This property field displays only if the structure of the parent record is delimited.</p> <p>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.</p> <p>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.</p>
Padding character	<p>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.</p>
Valid Characters	<p>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 does not belong to this list, then parsing error is thrown.</p>
Invalid Characters	<p>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.</p>
Trim Spaces	<p>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:</p> <p>Both (Leading and Trailing)</p> <p>Leading</p> <p>Trailing</p> <p>None</p>
Name	<p>The name of the field. The name of the node should be a valid XML name.</p>
Description	<p>The description of the field.</p>

Chapter 20: Fiorano Tools Perspective

Fiorano Tools Perspective in eStudio contains ESB Tools like Event Manager, Deployment Manager and License Manager along with Flat File Schema Tool. The default Fiorano Tools Perspective has the following components:

- Tools View
- Flat File Schemas View
- Properties view
- Error and Console Logs.

20.1 Fiorano Tools Perspective

To open Fiorano Tools Perspective, navigate to Window -> Open Perspective -> Other and select Fiorano Tools.

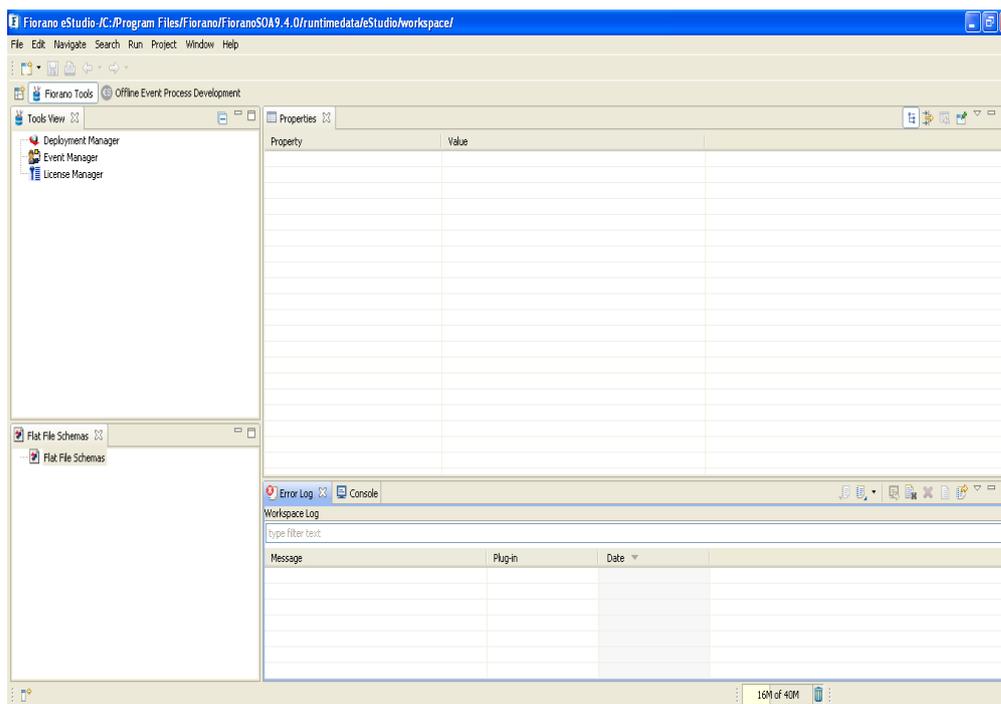


Figure 1: Fiorano Tools Perspective

20.1.1 Fiorano Tools View

The Fiorano Tools view acts as an explorer for the ESB Tools. It holds the following ESB Tools:

- Deployment Manager
- Event Manager
- License Manager.

20.2 Deployment Manager

Fiorano Deployment Manager allows you to control the deployment of business components and event processes on the various nodes in Fiorano network. You can use a combination of Labels (Development, QA, Staging and Production) and other identifiers (GUIDs, version numbers and node names) to create comprehensive and powerful rules to control the deployment of Event Processes.

Select Deployment Manager Node from Tools View, right-click and select Login to connect to Enterprise Server.

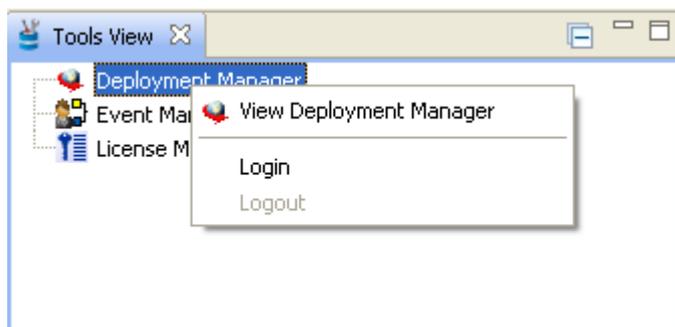


Figure 2: Tools View

After Login, a Deployment Manager View is opened adjacent to the Tools View. All Deployment Manager related actions like designing of new rules, modifying existing rules, etc can be performed from this view. To explicitly open Deployment Manager View, Right click on Deployment Manager Node and select “View Deployment manager”.

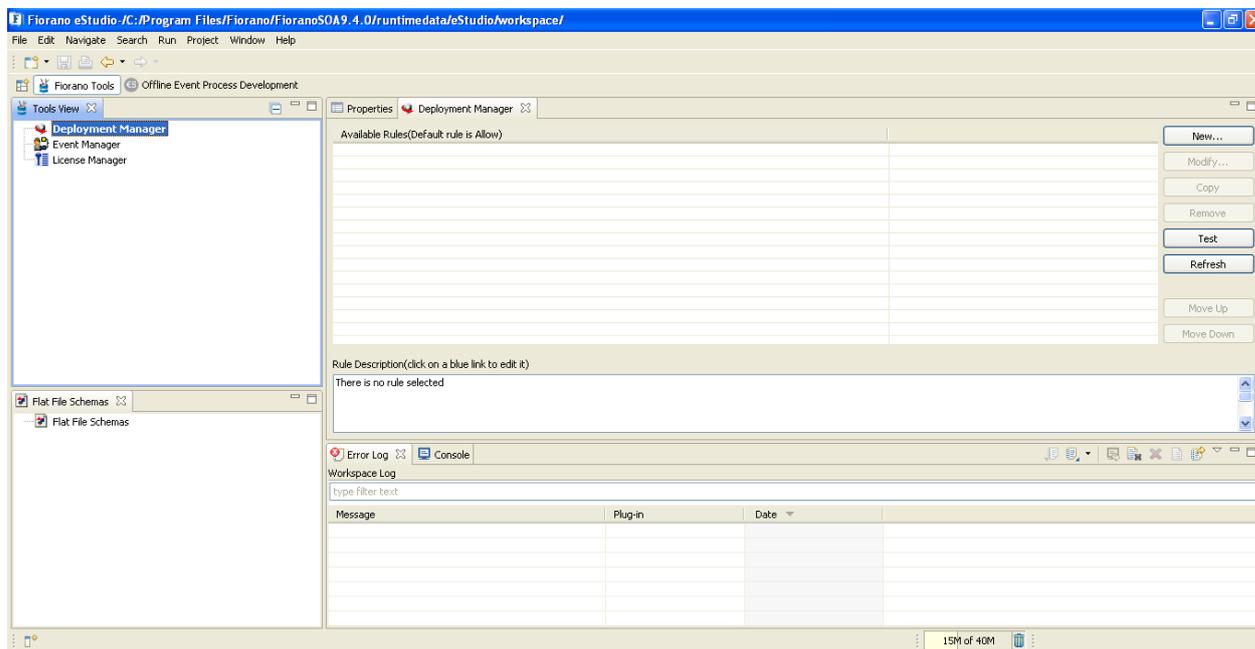


Figure 3: Fiorano Deployment Manager View

20.2.1 Using Fiorano Deployment Manager

Fiorano Deployment Manager is a fairly straightforward tool with self explanatory controls. This section briefly describes the various actions that can be performed using this tool.

20.2.1.1 Creating a Rule

You can set rules for the deployment of a business component in an event process(s) on a particular node(s).

To create a new rule, perform the following steps:

- Click on **New** button, the **New Rule** dialog box is displayed.

The 'New Rule' dialog box is shown with the following fields and controls:

- Name of the Rule:** An empty text input field.
- Allow/Disallow:** Two radio buttons, with 'Allow' selected.
- Execution of Business Component[s]:** A table with three rows of check boxes:

Execution of Business Component[s]	
<input type="checkbox"/> where business component guid contains guid	
<input type="checkbox"/> where business component version matches version	
<input type="checkbox"/> where business component label contains label	
- As part of Event Process[s]:** A table with three rows of check boxes:

As part of Event Process[s]	
<input type="checkbox"/> where event process guid contains guid	
<input type="checkbox"/> where event process version matches version	
<input type="checkbox"/> where event process label matches label	
- On peer server[s]:** A table with two rows of check boxes:

On peer server[s]	
<input type="checkbox"/> where peer server name contains name	
<input type="checkbox"/> where peer server label contains label	
- Rule Description:** A text area containing the word 'Allow'.
- Validation:** A red error message 'Name should not be empty' is displayed at the bottom left.
- Buttons:** 'OK' and 'Cancel' buttons are located at the bottom right.

Figure 4: New Rule Dialog

The New Rule window consists of the following controls:

1. **Allow:** Select this option to create a positive rule. A positive rule is one where a business component, event process, or peer server are allowed to perform a task.
2. **Disallow:** Select this option to create a negative rule. A negative rule is one where a business component, event process, or peer server are restricted from performing a task.
3. **Execution of business service:** This field contains three check boxes for creating business service specific rules:

- Where business component guid contains guid: Select this option to specify the GUID of a business service in a rule.
 - Where business component version matches version: Select this option to specify the Version of a business service in a rule.
 - Where business component label contains label: Select this option to specify the Label of a business service in a rule.
- 4. As part of event process(s): This field contains three check boxes for creating event process specific rules
 - Where event process guid contains guid: Select this option to specify the GUID of an event process in a rule.
 - Where event process version matches version: Select this option to specify the Version of an event process in a rule.
 - Where event process label matches label: Select this option to specify the Label of an event process in a rule.
- 5. On peer server[s]:
 - Where peer server name contains name: Select this option to specify the name of a peer server in a rule.
 - Where peer server label contains label: Select this option to specify the label of a peer server in a rule.
- Select Allow or Disallow button for choosing the deployment of business service.
- Set conditions for the rule in Business component[s] panel selecting the desired check boxes. You can specify multiple conditions for a single rule by selecting more than one check box.
 - To set the rule on the basis Business Component GUID, select Where business component guid contains guid condition and select GUID hyperlink from “Rule Description panel”, and select the required Business Components from the Business Component selection dialog

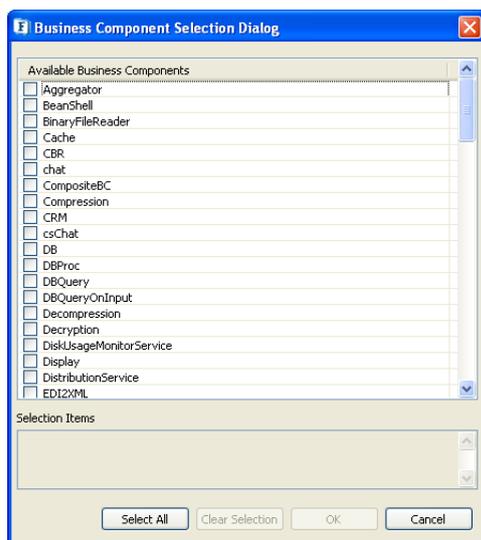


Figure 5: Business Component Selection dialog

- To set the rule on the basis of Business Component Version, select Where business component version matches version condition and select Version hyperlink from “Rule Description panel”, and select the required Business Component Versions from the Business Component Version selection dialog

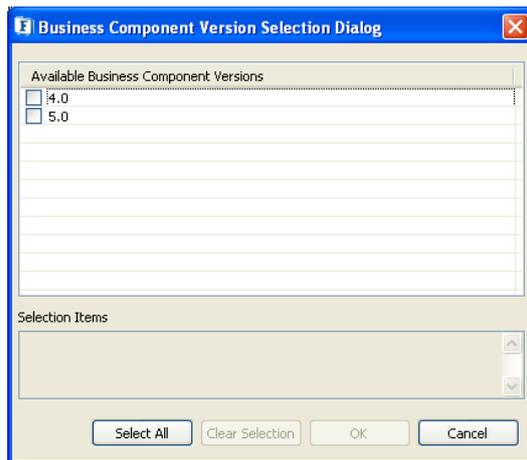


Figure 6: Business Component version selection dialog

- To set the rule on the basis of Business Component Label, select Where business component label matches label condition and select Label hyperlink from “Rule Description panel”, and select the required Business Component Label from the Business Component Label selection dialog

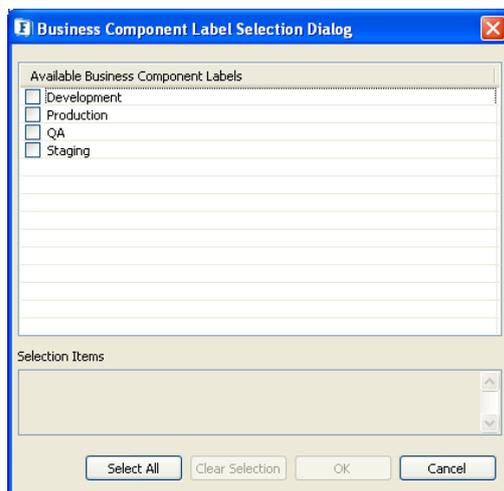


Figure 7: Business Component Version selection dialog

- Set conditions for the rule in Event Process[s] panel selecting the desired check boxes. You can specify multiple conditions for a single rule by selecting more than one check box.
 - To set the rule on the basis of Event Process GUID, select Where event process guid contains guid condition and select GUID hyperlink from “Rule Description panel”, and select the required Event Processes from the Event Process selection dialog

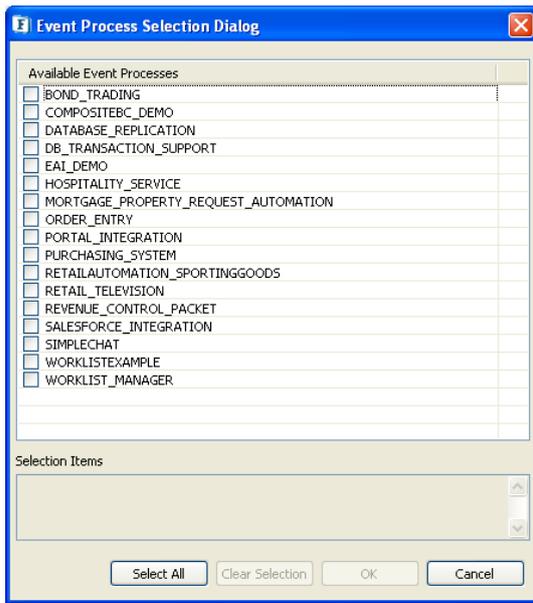


Figure 1: Event Process selection dialog

- To set the rule on the basis of Event Process Version, select Where event process version matches version condition and select Version hyperlink from “Rule Description panel”, and select the required Event Process Versions from the Event Process Version selection dialog

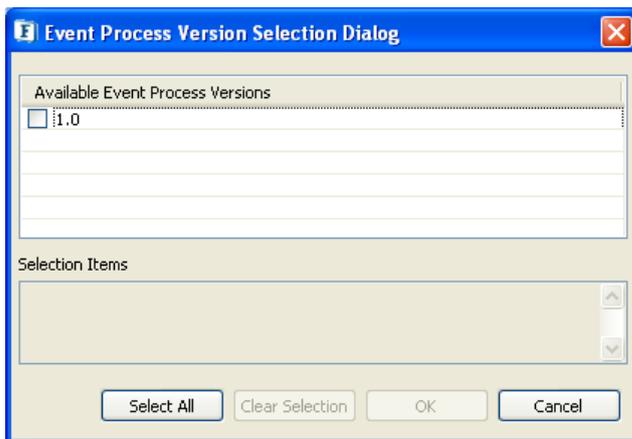


Figure 9: Event Process Version selection dialog

- To set the rule on the basis of Event Process Label, select Where event process label matches label condition and select Label hyperlink from “Rule Description panel”, and select the required Event process labels from the Event Process label selection dialog

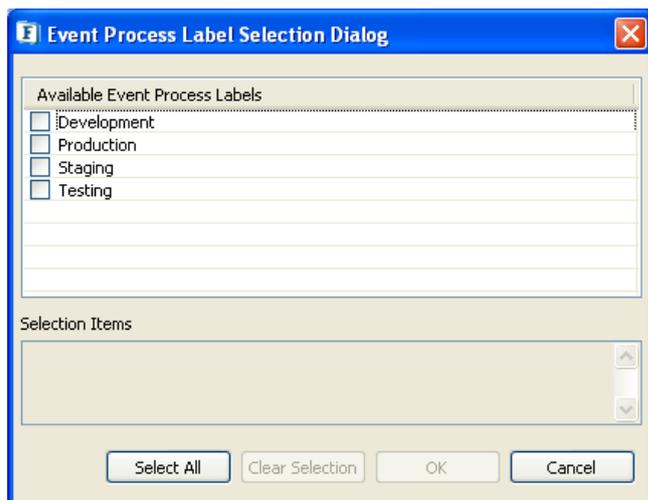


Figure 10: Event Process Label selection dialog

- Set conditions for the rule in Peer Server[s] panel by selecting the desired check boxes. You can specify multiple conditions for a single rule by selecting more than one check box.
 - To set the rule on the basis of Peer Server name, select Where peer server name contains name condition and select name hyperlink from “Rule Description panel”, and select the required peer servers from the Peer Server selection dialog

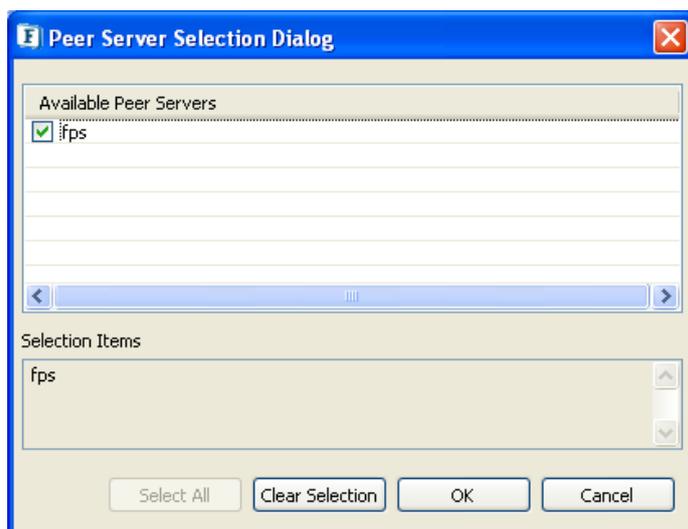


Figure 11: Peer Server selection dialog

- To set the rule on the basis of Peer Server label, select where peer server label contains label condition and select label hyperlink from “Rule Description panel”, and select the required peer servers from the Peer Server label selection dialog.

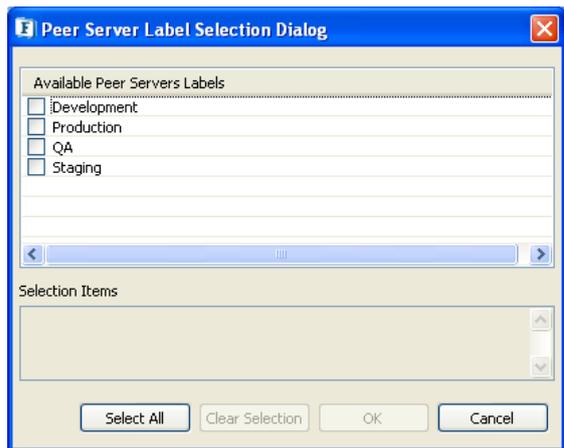


Figure 12: Peer Server label selection dialog

- Enter the name of the rule and click Ok button to create a new rule.

20.2.1.2 Modifying a Rule

To modify a rule, perform the following steps:

1. Select a Rule in the Available Rules section.
2. Click on the Modify button. The Modify Rule dialog is opened.
3. Make the required changes and click on Ok button.

20.2.1.3 Creating a copy of a Rule

Although creating Rules using Deployment Manager is straight forward, creating variants of complex rules can be cumbersome and repetitive. To simplify the process, the tool allows you to create copies of a rule. In this way, you can create copies of complex rules and tweak them to create variants. To create a copy of a rule:

1. Select a Rule in the Available Rules section.
2. Click on the Copy button. A copy of the Rule will be created with all the set conditions.
3. Make the required changes to the Rule.
4. Enter the name of the new Rule and click Ok button.

20.2.1.4 Deleting a Rule:

To delete a rule, perform the following steps:

1. Select a Rule in the Available Rules section.
2. Click on the Remove button.

20.2.1.5 Testing the Rule[s]

You can check the functionality of a Rule by testing it against the relevant parameters of a business component, event process or peer server. To test the Rule[s], perform the following steps:

1. Click on the Test button. The Test dialog box is opened.

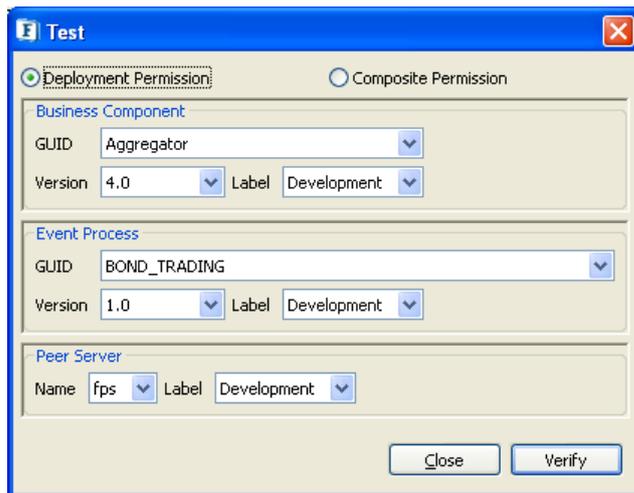


Figure 13: Test dialog

2. Select the Deployment Permission or Composition Permission option.
3. From the Business Component section, select applicable GUID from the GUID drop down.
4. Select Version and Label from their respective drop down lists.
5. From the Event Process section, select applicable Event Process GUID, Version and Label from their respective drop down lists.
6. Next, select peer server name and label from Peer Server section.
7. Click on Verify to test the Rule.

20.2.1.6 Refreshing the display of Rules

To refresh the display of Rules in the Available Rules section, click on Refresh button.

20.2.1.7 Changing the precedence of Rules:

In the Available Rules section, all the Rules starting from the top-most one are arranged in decreasing order of priority in the Rules pane.

To change the precedence of Rules, select the Rule and click Move Up or Move Down button.

20.3 Event Manager

Fiorano Event Manager Tool can be used to view Event Process events, normal business service events, monitor the states of various tracked documents, or to Track the status of an activity or document across a distributed workflow. The functionality of the tool can be broadly categorized into one of the following:

- Managing Services
- Managing Events
- Managing Document Tracking.

1. Select Event Manager Node from Tools View, right-click and select Login to connect to Enterprise Server.

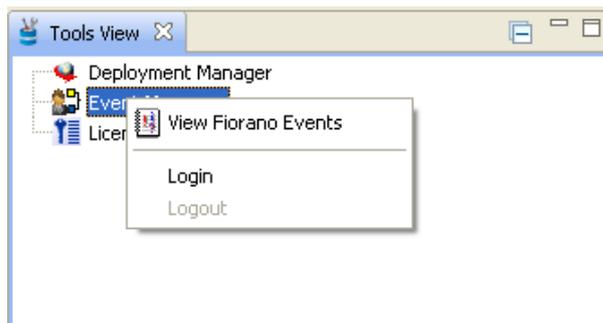


Figure 14: Tools View, Event Manager Node

2. After Login, an Events View is opened adjacent to the Tools View. All the Event Process events, Business Service events, States of tracked documents, Business Components for event processes etc can be viewed in this view. To explicitly open Events View, Right click on Event Manager Node and select “View Fiorano Events”.
3. The Event Manager Node contains an Event Process Repository Node, System Events Node and Security Events Node. The Event Process Repository Node in-turn contains a tree structure of various event processes in the Enterprise server. Each Event Process Node contains the following nodes:
 - Business Components Node
 - Process Events Node
 - Business Component Events Node
 - Tracked Documents Node

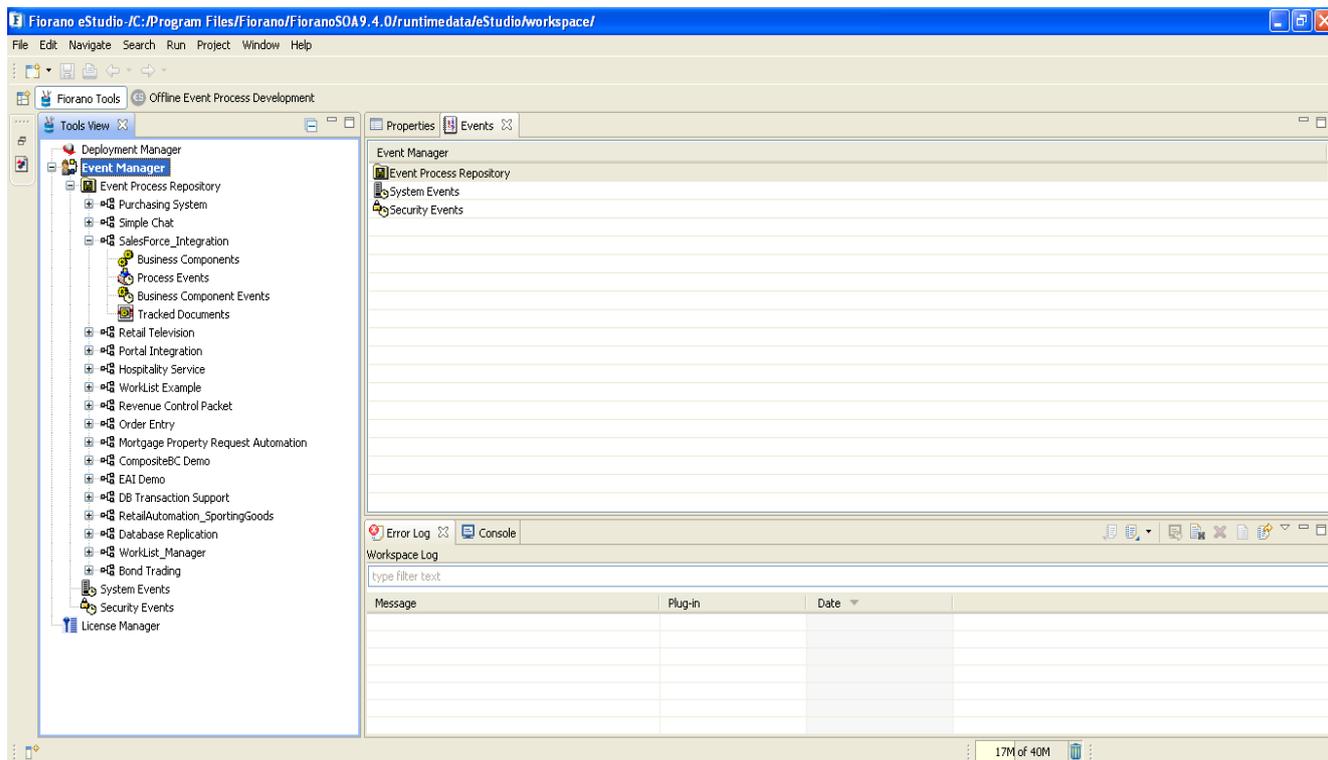


Figure 15: Events View

20.3.1 Managing Services

The various Business services in an Event Process and their current states can be viewed here. Also, Output and Error logs can be viewed for each service.

To view the details of a service:

1. Click Event Process Repository -> [Event Process Name] in Tools View
2. Select Business Components Node under the Event Process Node

Each Business service contains the following details:

- **Business Service Instance Name:** Name of the business service instance
- **Business Service GUID:** GUID (Global Unique Identifier) is used to uniquely identify the business service on the Fiorano network. This ID cannot hold any special characters or spaces. Business service GUID represents the GUID for the business service.
- **Version:** This is the version of the business service, and must be a valid floating point number.
- **Status:** Represents the status of the business service. Its value is set to “Not running” if the business service handle does not have the value SERVICE_HANDLE_BOUND or SERVICE_HANDLE_UNBOUND.
- **Node Name:** Name of the node where the business service is running.
- **Launch Time:** The time when the business service was launched.
- **Kill Time:** The time when the business service was killed.

The screenshot shows the Fiorano eStudio interface with the 'Events' view selected. The main window displays a table of Business Services with the following data:

Business Component Instance Name	Business Component GUID	Version	Status	Node Name	Launch Time	Kill Time
HttpReceive	HttpReceive	4.0	SERVICE_HANDLE_UNBOUND	fps	Aug 17 2011 01:01:28 PM	Aug 17 2011 01:01:44 PM
PO_Database	DB	4.0	SERVICE_HANDLE_BOUND	fps	Aug 17 2011 01:01:28 PM	Not set
Transform_PO_Details	Xslt	4.0	SERVICE_HANDLE_BOUND	fps	Aug 17 2011 01:01:27 PM	Not set
Extract_PO_Details	Xslt	4.0	SERVICE_HANDLE_BOUND	fps	Aug 17 2011 01:01:28 PM	Not set
Get_FinalDetails	Xslt	4.0	SERVICE_HANDLE_BOUND	fps	Aug 17 2011 01:01:28 PM	Not set
ExtractErrorResponse	Xslt	4.0	SERVICE_HANDLE_UNBOUND	fps	Aug 17 2011 01:01:28 PM	Aug 17 2011 01:01:43 PM

The interface also shows a tree view on the left with 'Business Components' expanded, and a console window at the bottom.

Figure 16: Business Components in Events View

To view the logs of a service:

1. Select a service in Events View
2. Right click on the service and select 'View Logs'

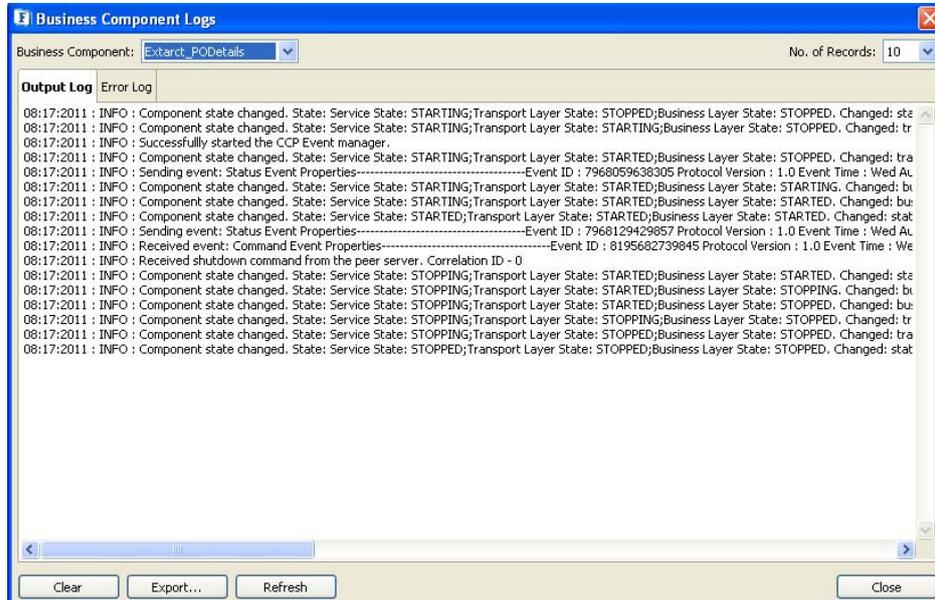


Figure 172: Business Component Logs

20.3.2 Managing Events

Fiorano Event Manager allows you to log and monitor events generated by event processes at the FES layer. The various events that are logged by the event viewer are as follows:

1. Process Events: System Events pertaining to an event process. For example, on launching an event process APPLICATION_LAUNCH_STARTED event is generated.

To view Process Events:

- Select Event Process Repository Node -> [Event Process Name] in Tools View
- Select Process Events Node under the Event Process Node

2. Business Service Events: System Events that are related to services are Service Events.

To view Service Events:

- Select Event Process Repository Node -> [Event Process Name] in Tools View
- Select Business Component Events Node under Event Process Node

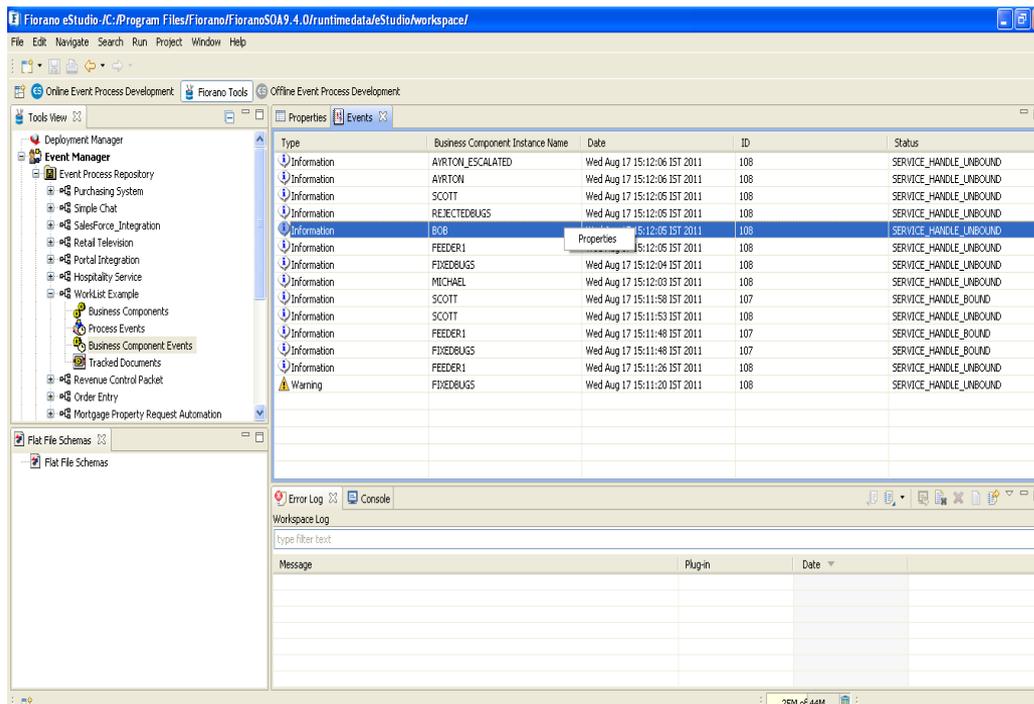


Figure 18: Business Component Events in Events View

Event Properties: To view the information pertaining to a specified Event, in detail:

- Select Event Process Repository Node -> [Event Process Name] -> [Events Node] in Tools View.
- Select an Event, right-click on the event and select Properties.

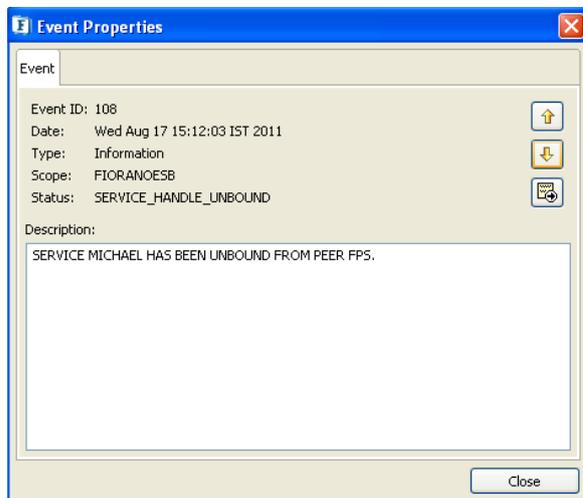


Figure 19: Event Properties dialog

20.3.3 Managing Document Tracking

The documents flowing from one Service to another in an Event Process can be tracked by enabling Document Tracking in FEPO. All such Tracked Documents and their properties can be viewed in Fiorano Event Manager.

To view Tracked Documents in an Event Process:

- Click Event Process Repository -> [Event Process Name] in Tools View

- Select Tracked Documents Node under the Event Process Node

WorkFlow Information:

When you click on the Tracked Documents Node in the Tools View, a detailed table is displayed in the Events View. This is termed as the Workflow table. It contains information such as workflow Instance ID and the current status of the workflow. Each Event Process is viewed as a Workflow area, within which a large number of documents and messages flow.

The Workflow table for a Tracked Document shows the following details:

- **Instance ID:** An Event Process can have many of its instances at the same time. These are called as Workflow Instances. Every time a new document enters the workflow, a new workflow instance is generated with a new Workflow Instance ID.
- **Status:** The current execution status of the Workflow Instance.
- **Last Business Service:** Represents the last business service that was tracked in the Workflow.
- **Cycle Time:** Time spent by the document in traversing the Workflow.

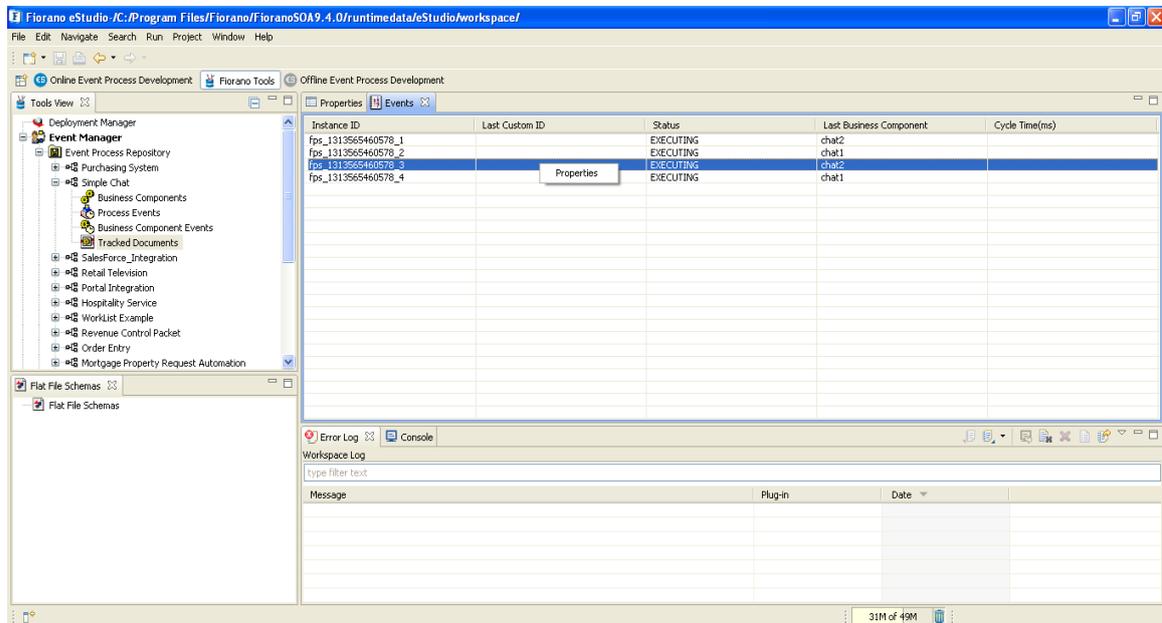


Figure 20: Tracked Documents in Events View

Document Properties:

To view the information pertaining to a Specific Document like the various Document States Traversed etc, in detail:

- Select Tracked Documents Node under the Event Process Node
- Select a Document, Right-click on the document and select properties.

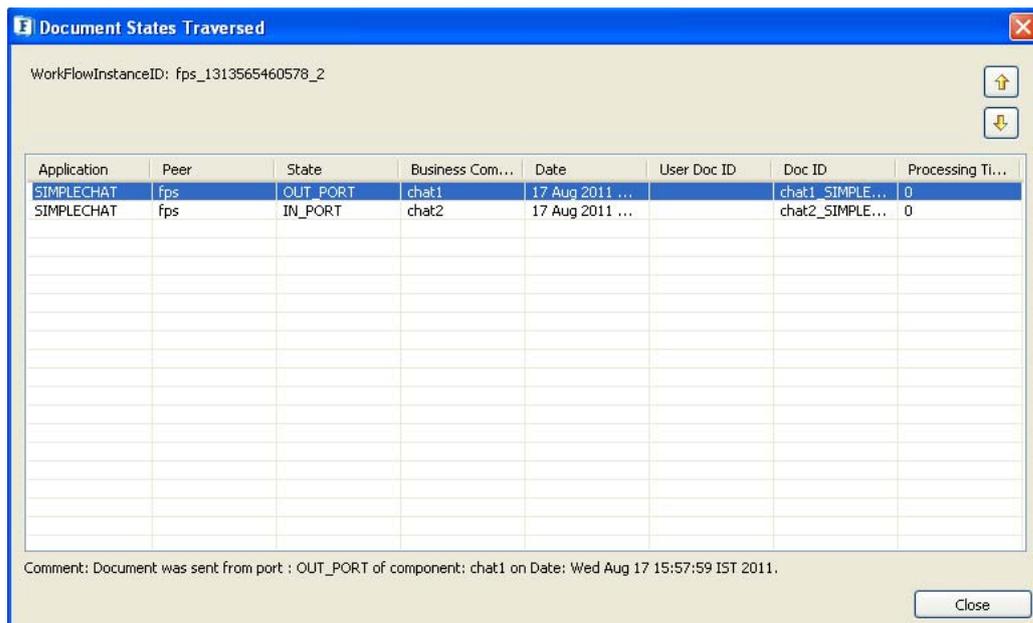


Figure 21: Document Stated Traversed dialog

Document State Properties:

To view Properties of the Document States Traversed, Double-click on the Document State. If the Tracked Document contains any file attachments, go to Attachments tab, select the file attachment and use “Save As...” to save the attachment on the local disk.

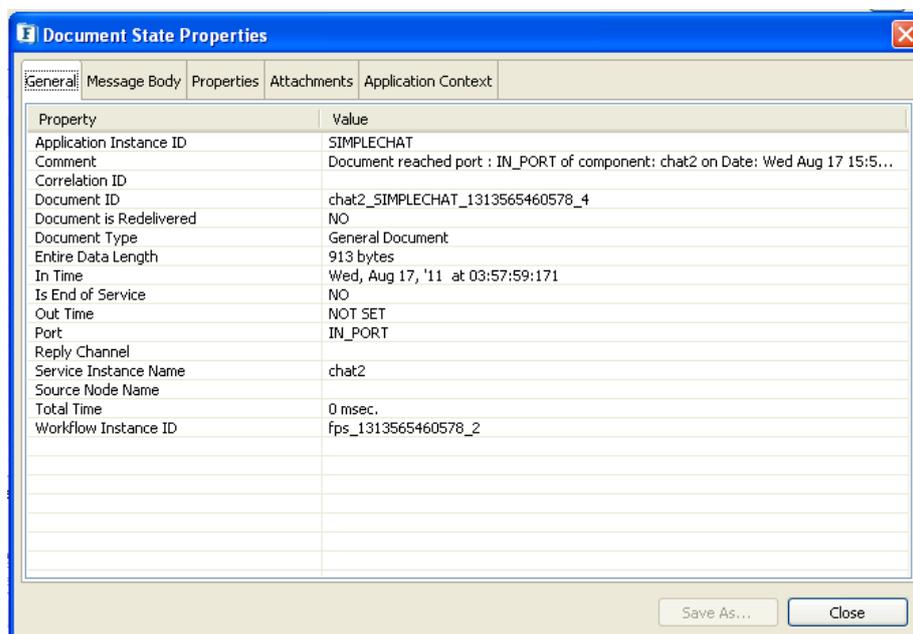


Figure 22: Document State Properties dialog

In addition to above, the following actions can be performed with respect to Event Manager Tool.

1. Right-click on Event Process Repository node and select Refresh to refetch all the event processes, their components, events and tracked documents from server.
2. Select Event Process Node and right-click.
 - a. Select Clear EventProcess Logs to clear all the event process logs.

- b. Select Clear all Events to clear the process and component events of the selected Event Process
 - c. Select Refresh to fetch Business components, Process and Component events and tracked documents from the server.
- 3. Select Business components node and Right-click.
 - a. Select Export List to export the Business Components with their current execution states and other details as a text file.
 - b. Select Refresh to fetch the business components from server.
- 4. Select Process Events Node \ Component Events Node \ Tracked Documents Node and right-click
 - a. Select Export List to export the events or tracked documents as a text file.
 - b. Select Clear to clear all the events or tracked documents of the selected node.
 - c. Select Refresh to fetch events or tracked documents for the selected node from server.

20.4 License Manager

The License Manager Tool enables you to manage Fiorano SOA Platform licenses. This tool can also be used to gather machine information and request for additional licenses.

20.4.1 Fiorano Licenses Overview

Fiorano SOA Platform has been segregated into various modules. The product modules and the license files required to enable these modules are listed below. These License files can be managed using License Manager Tool.

Module	License File
Fiorano Prebuilt Components(BCs and EDBCs)	fiorano-soa9.lic
Fiorano ESB Server	fiorano-soa9.lic
Fiorano MQ 9 Server The license for FioranoMQ 9 Server offered with Fiorano SOA Platform is included in the Fiorano-soa9.lic file.	fiorano-mq9.lic
Fiorano Studio Fiorano eStudio	fiorano-soa9.lic
Fiorano Tools Fiorano Deployment Manager Fiorano Event Manager Fiorano Mapper Fiorano Event Process Orchestrator	fiorano-soa9.lic

20.4.2 Tool Environment

Select License Manager Node from Tools View, right-click and select Load to load license files as shown in Figure 23.

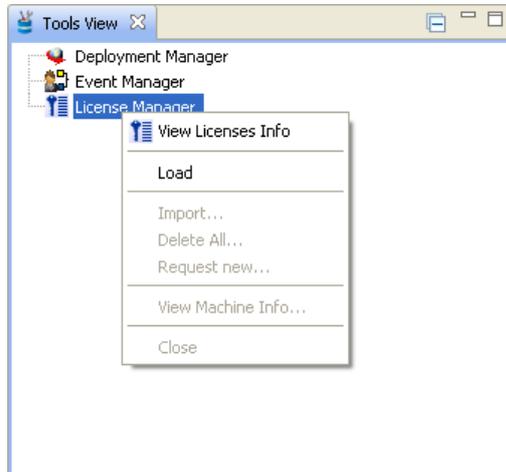


Figure 23: License Manager Node, Tools View

After the licenses are loaded successfully, a Licenses View is opened adjacent to the Tools View. All the license related information can be viewed in the Licenses View. To explicitly open Licenses View, Right click on License Manager Node and select View Licenses Info.

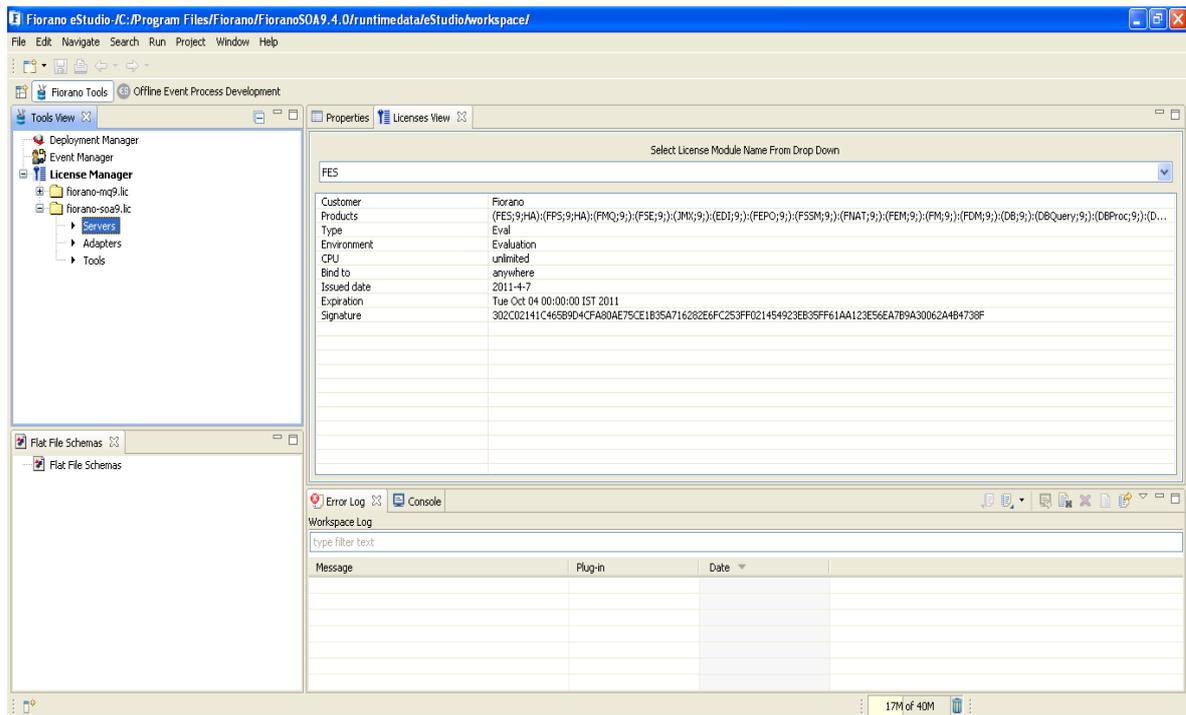


Figure 24: Licenses View

All the Licenses present in “FIORANO_HOME\ licenses” folder are shown as License Nodes under the License Manager Node in the Tools View. Each License Node in-turn contains Servers, Adapters and Tools nodes. Each of these nodes on selection shows the Servers, Adapters or Tools respectively in the drop down along with the license configuration details in a table below.

20.4.3 License Configuration

To view License Configuration, select a license node under the License Manager Node. The Configuration for the selected license is shown in key-value pairs in the Licenses View. The Configuration parameters and their respective descriptions are shown below:

Parameter	Description
Customer	Displays the name of the customer.
Products	Displays the name of the enabled product modules.
Type	Displays the type of license used.
Environment	Displays the environment where the product is used.
CPU	Displays the number of CPUs supported by the installed license
Bind To	Displays whether the license is bound to the Host Name of your system or to the IP address of your system.
Issued Date	Displays the issue date of the installed license
Expiration	Displays the expiry date of the installed license
Signature	Displays the encrypted signature of the installation.

20.4.4 Managing Fiorano Licenses

This section describes the following tasks that enable you to manage Fiorano licenses using the License Manager Tool:

- Viewing and saving system Information
- Acquiring, Validating and Adding a license.
- Removing a license.

Viewing and saving Information:

The following steps enable you to view license specific information:

1. Right-click on License Manager Node and select Machine Info. A machine information pop-up appears and displays license specific machine information as name-value pairs
2. Save the displayed information by clicking in the Save button on the top right hand corner on the machine pop-up.

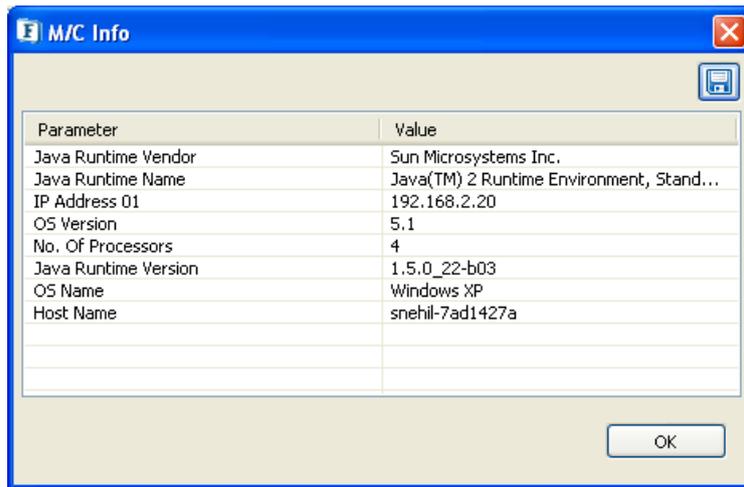


Figure 25: Machine Info dialog

Acquiring, Validating and Adding a License:

To acquire a new License:

1. Right-click on License Manager Node and select Request new. This redirects you to Fiorano License Portal so that a new license request can be registered with us.
2. You will receive the requested license(s) after successful execution of your license request. These licenses will be delivered to you in as Fiorano<product_name>.lic files which can be imported, validated and added to the Fiorano SOA Platform using the License Manager Tool.

To import, validate and add a license:

1. Right-click on License Manager Node -> select Import, to import a new License File into the License Manager tool. The imported license will be added as a License Node under the License Manager Node.
2. Right-click on any License Node -> select Validate to validate the license with respect to your Fiorano SOA Platform.
3. Upon importing a license from the License Manager tool, the license is successfully added to the Fiorano SOA Platform.

Removing a License:

1. Right-click on a License Node -> select Delete to delete the selected license file from Fiorano SOA Platform
2. To delete all licenses, right-click on License Manager Node -> select Delete All. All license files are deleted from Fiorano SOA Platform.

In addition to above, the following actions can be performed with respect to License Manager Node:

Select the License Manager Node and right-click:

- View Fiorano Licenses: Opens the Licenses View. All the license related information can be viewed here.
- Load: Load licenses present in "FIORANO_HOME\licenses" folder.
- Close: Close all previously loaded licenses.