



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) 2008-2010, Fiorano Software Pte Ltd. and Affiliates

Contents

Chapter 1: Introduction to Fiorano eStudio 22

- 1.1 Key Features 22
- 1.2 Getting started with Fiorano eStudio 23
 - 1.2.1 Launching eStudio using 64-bit JVM 24

Chapter 2: Offline Event Process Development 26

- 2.1 Fiorano Views 27
 - 2.1.1 Event Process Repository View 27
 - 2.1.2 Fiorano Orchestration 27
 - 2.1.3 Service Palette 28
 - 2.1.4 Properties 29
 - 2.1.5 Problems 29
 - 2.1.6 Error Log 30
 - 2.1.7 Service Repository (Offline) 30
 - 2.1.8 Project Explorer 31
 - 2.1.9 Service Descriptor Editor 32
- 2.2 Event Processes 33
 - 2.2.1 Creating New Event Process 33
 - 2.2.2 Opening Sample Event Process 34
 - 2.2.3 Import and Export Event Processes 35
 - 2.2.3.1 Exporting an Event Process 35
 - 2.2.3.2 Importing an Event Process 37
 - 2.2.4 Importing nStudio Event Processes 39
- 2.3 Service Repository (Offline Event Process Development) 40
 - 2.3.1 Deploying Services to Server 41
 - 2.3.2 Fetching Services from Server 42
 - 2.3.3 Exporting Services to Local Disk 43
 - 2.3.4 Importing Services from Local disk 43

Chapter 3: Online Event Process Development Perspective 44

- 3.1 Fiorano Views 46
 - 3.1.1 Server Explorer 46
 - 3.1.2 Fiorano Debugger View 46
- 3.2 Service Repository (Online Event Process Development) 47
 - 3.2.1 Exporting Services to Local Disk 47
 - 3.2.2 Importing Services from Local disk 48

Chapter 4: Mapper Perspective **50**

Chapter 5: Composing Event Processes **51**

5.1 Adding Components	51
5.2 Connecting Routes	52
5.3 Configuring Components	52
5.4 Configuring Component Properties	54
5.5 Adding Remote Service Instance	56
5.6 Adding External Event Process (Subflow)	58
5.7 Document Tracking	60
5.8 Defining Route Transformations	62
5.9 Configuring Selectors on Routes	65
5.10 Configuring Application Context	66
5.11 Check Resource and Connectivity	68
5.12 Running Event Process	69
5.13 Stopping an Event Process	69
5.14 Synchronizing an Event Process	70

Chapter 6: Event Process Life Cycle Management **72**

6.1 Setting Properties of Service Instances for Different Environments	72
6.2 Running Event Process on an Environment	73

Chapter 7: Debugging Event Process **74**

7.1 Adding Breakpoint	74
7.1.1 Context Menu option	74
7.1.2 Debugger View	75
7.2 Viewing Messages at Breakpoint	76
7.3 Editing Messages at Breakpoint	76
7.4 Inserting Messages into Breakpoint	77
7.5 Releasing Messages from Breakpoint	78
7.6 Discard Messages from Breakpoint	79
7.7 Remove Breakpoint	80

Chapter 8: Services **81**

8.1 Service Descriptor Editor	81
8.1.1 Overview Section	83
8.1.2 Execution Section	84
8.1.2.1 Port Information	85
8.1.2.2 Support	85
8.1.2.3 Launch Configuration	86

8.1.2.4 Log Modules	86
8.1.2.4 Runtime	87
8.1.3 Deployment Section	87
8.1.3.1 Resource	88
8.1.3.2 Service Dependencies	88

Chapter 9: Service Creation **90**

9.1 Service Generation	90
9.1.1 Service Location	90
9.1.2 Basic Details	91
9.1.3 Ports Information	92
9.1.4 Resources	93
9.1.5 Dependencies	94
9.2 Building and Deploying Services	94

Chapter 10: eMapper **96**

10.1 Key Features of Fiorano eMapper	96
10.2 Fiorano eMapper Environment	96
10.2.1 eMapper Projects.	97
10.2.2 eMapper Editor	98
10.2.2.1 Map View	98
10.2.2.2 MetaData tab	99
10.2.3 Funclet View	99
10.2.4 eMapper Console	100
10.2.5 MetaData Messages View	100
10.2.6 Node Info View	101
10.3 Working with Input and Output Structures	101
10.3.1 Loading Input/Output Structure	101
10.3.1.1 Load Input/Output Structure From an XSD document	101
10.3.1.2 Load Input/Output Structure from a DTD document	104
10.3.1.3 Load Input/Output Structure from an XML document	104
10.3.2 Delete Structure	105
10.3.3 Edit Structure	106
10.4 Working with the Visual Expression Builder	106
10.4.1 Function Palette	107
10.4.1.2 Math Functions	110
10.4.1.3 String Functions	112
10.4.1.4 Control Function	115
10.4.1.5 Conversion Functions	115
10.4.1.6 Advanced Functions	117
10.4.1.7 Date-Time Functions	119
10.4.1.8 NodeSet Functions	125
10.4.1.9 Boolean functions	128

10.4.1.10 Lookup functions	139
10.4.1.11 JMS Message Functions	141
10.4.1.12 User Defined functions	142
10.4.2 Funclet Easel	143
10.4.2.1 Source Node	144
10.4.2.2 Destination Node	144
10.5 Creating Mappings	147
10.5.1 Understanding Types of Nodes	147
10.5.2 Types of Mappings	149
10.5.2.1 Name-to-Name Mapping	149
10.5.2.2 For-Each Mapping	150
10.5.3 Duplicating a For-Each Mapping	151
10.5.4 Linking Nodes to Define Mappings	153
10.5.4.1 Using the Automatic Mapping option to Define Mappings	153
10.5.4.2 Using the Visual Expression Builder to Define Mappings	154
10.5.5 Mapping XML Formats	157
10.6 Adding User XSLT	157
10.7 Working with derived types	160
10.8 Create/Edit User Defined Function(s)	162
10.9 Testing the Transformation	165
10.10 Managing Mappings	170
10.10.1 Exporting eMapper Project	170
10.10.2 Importing Project from the File	171
10.10.3 Copying functions in a Mapping	171
10.10.4 Clearing All Mappings	171
10.10.5 Managing XSLT Properties	172

Chapter 11: Working With Multiple Servers And Perspectives 173

11.1 Active Server Node	173
11.2 Switching of Active Server	174
11.3 Switching Between Perspectives	176

Chapter 12: Fiorano Preferences 179

12.1 ESB Connection Preferences	179
12.2 SOA Orchestration	180
12.2.1 General Options	180
12.2.2 Workflow Options	180
12.2.3 Service Options	180
12.2.3.1 Default JVM Configurations	181
12.2.3.2 Connection Factory Preferences	182
12.2.4 CPS Options	182
12.3 SOA Orchestration Online	184

12.3.1 General Options	184
12.3.2 Application Options	185
12.3.3 Service Options	185
12.3.4 Peer Options	185
12.4 Key Board Short Cut Preferences	186

Chapter 13: Schema Repository 189

Chapter 14: SCM Integration 191

14.1 Downloading and integrating SCM plugins in Fiorano eStudio	191
14.2 Specifying SCM repository	191
14.3 Creating a project for version control	192
14.4 Adding the Project to Repository	195
14.5 Updating the project into the Repository.	197
14.6 Updating an Event Process with older version from Repository	197

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.

The following steps have to be followed to launch eStudio using 64-bit JVM.

1. SWT binary and source distribution for all platforms is available for download at <http://archive.eclipse.org/eclipse/downloads/drops/R-3.5.2-201002111343/#SWT>. Download the distribution for the required platform.
2. Unzip the directory and copy the **swt.jar** file.
3. Replace the SWT jar file located at FIORANO_HOME/eStudio/plugins (example, for linux org.eclipse.swt.gtk.linux.x86_3.5.2.v3557f.jar) with the 64 bit swt.jar file.

Note: The new swt.jar should be renamed to the name of the actual swt jar file located at FIORANO_HOME/eStudio/plugins and the original one has to be replaced.

Fiorano eStudio has three perspectives:

1. Offline Event Process Development Perspective
2. Online Event Process Development Perspective
3. Mapper Perspective

A perspective defines the initial set and layout of views in the Workbench window. Within each window, the perspective has a set of views and editors. Each perspective provides a set of functionality aimed at accomplishing a specific type of task or works with specific types of resources.

For example, the Java perspective contains views that are commonly used while editing Java source files, while the Debug perspective contains the views used while debugging Java programs. User can switch from one perspective to another.

An icon added to the shortcut bar allows you to switch to other perspectives.

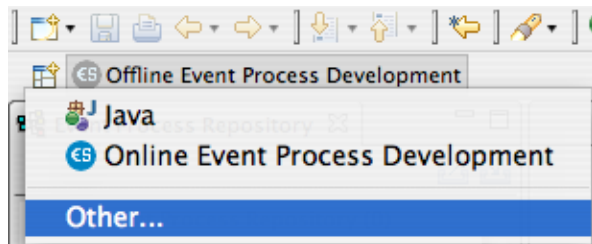


Figure 1.2.1: Perspective switch

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.

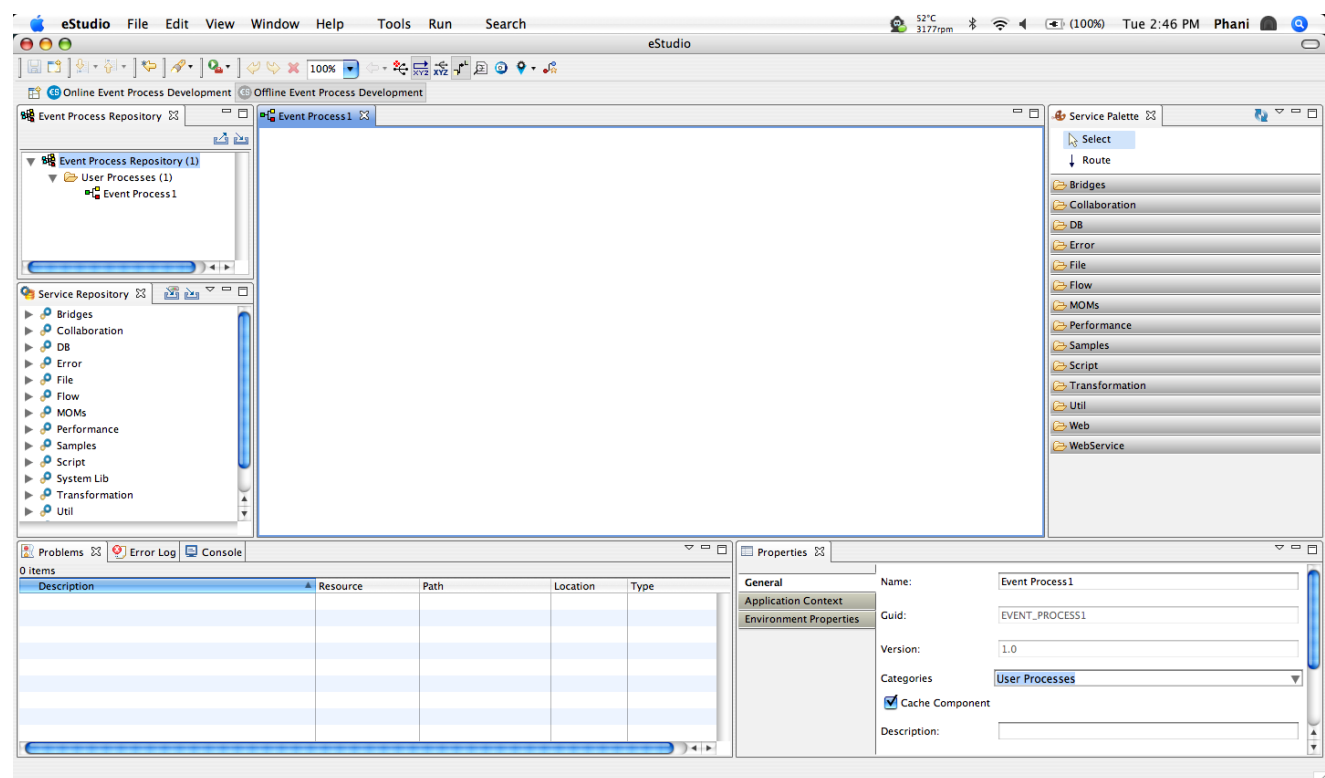


Figure 2.1: Offline Event Process Development perspective

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.

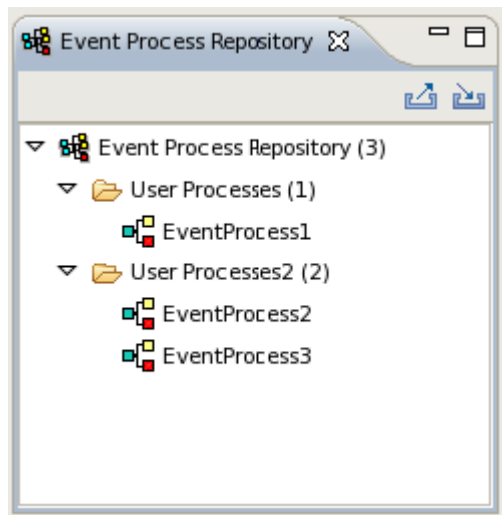
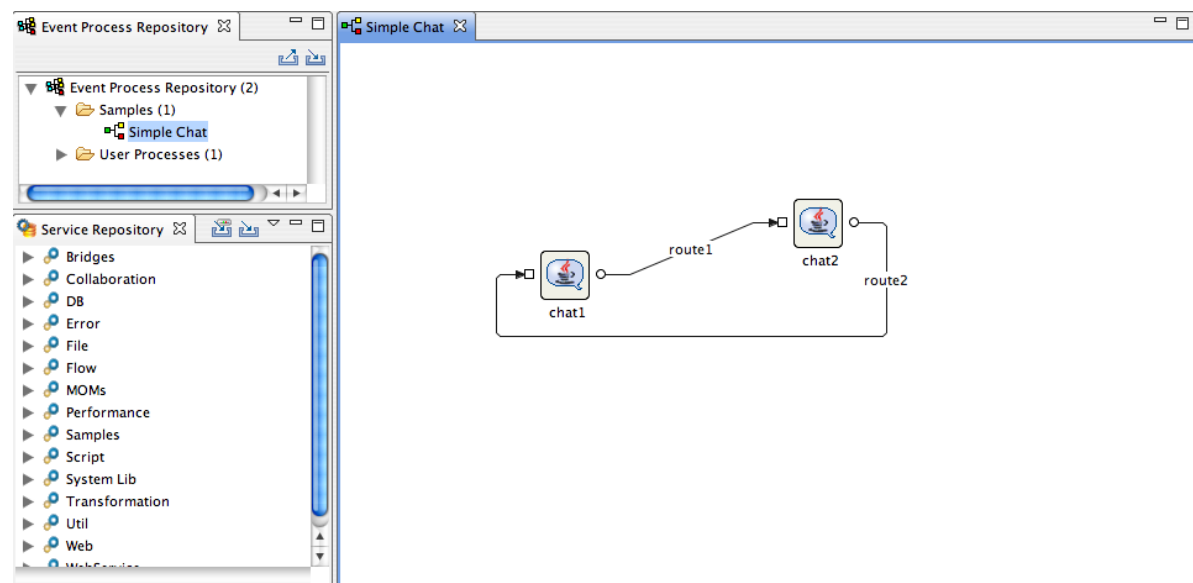


Figure 2.1.1: Event Process Repository

2.1.2 Fiorano Orchestration

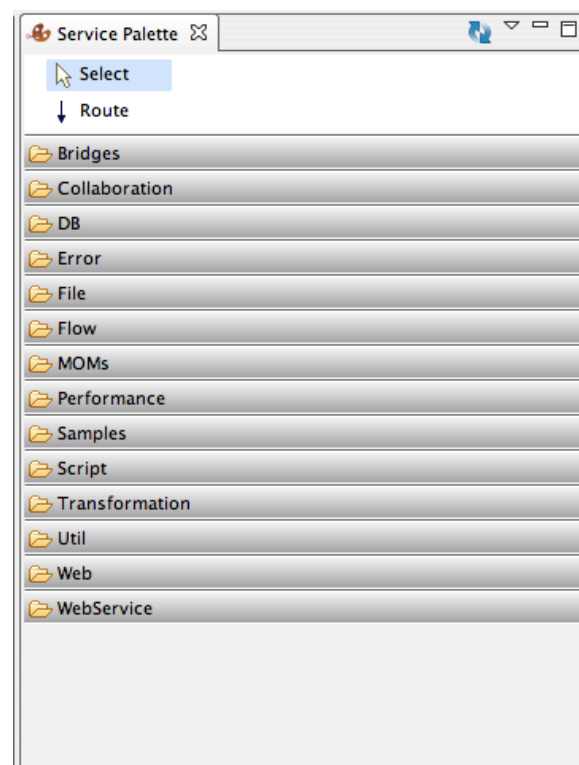
Offline and Online Event Process Development perspectives are comprised of an editor area Fiorano Orchestrator.

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.

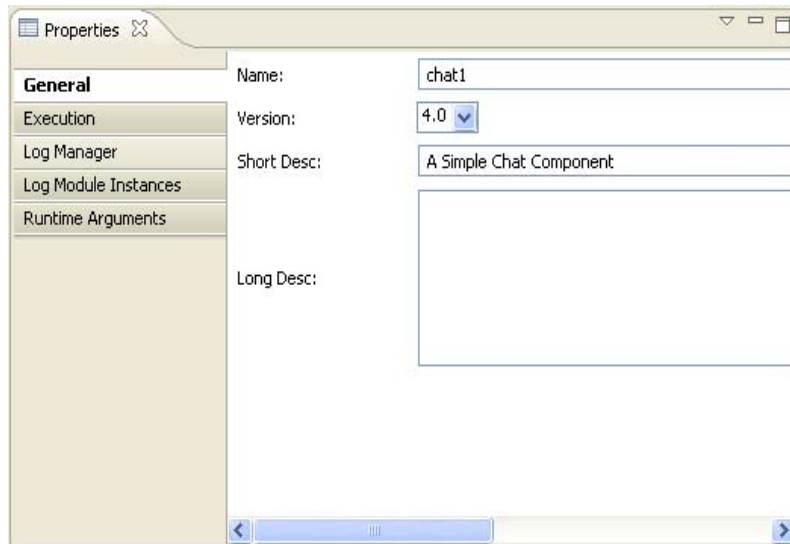


Figure 2.1.4: Properties view

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.

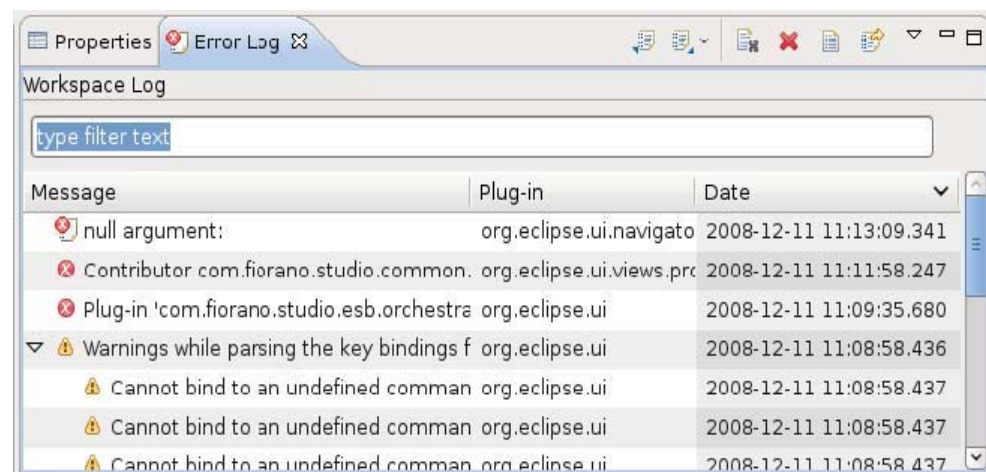


Figure 2.1.6: Error Log view

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.

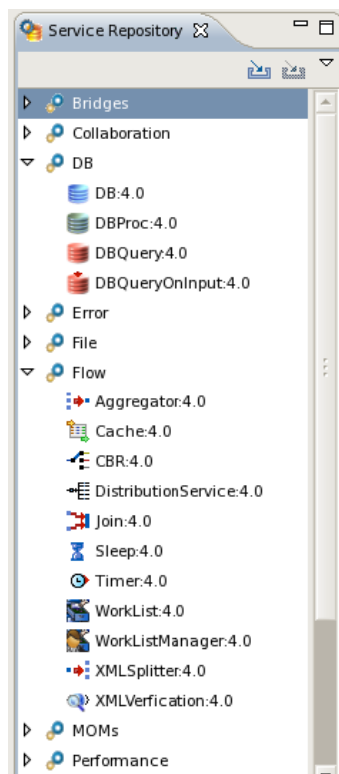


Figure 2.1.7: Service Repository

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.

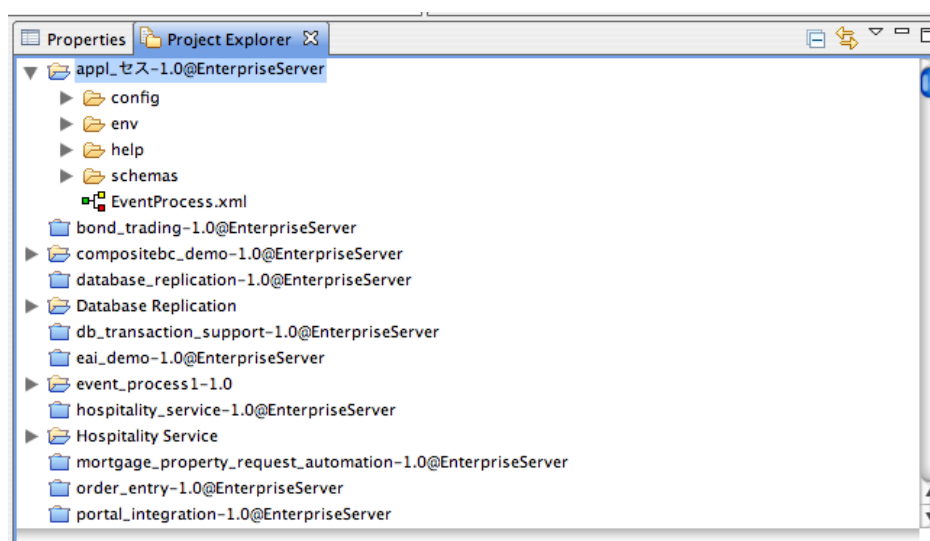


Figure 2.1.8: Project Explorer

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.

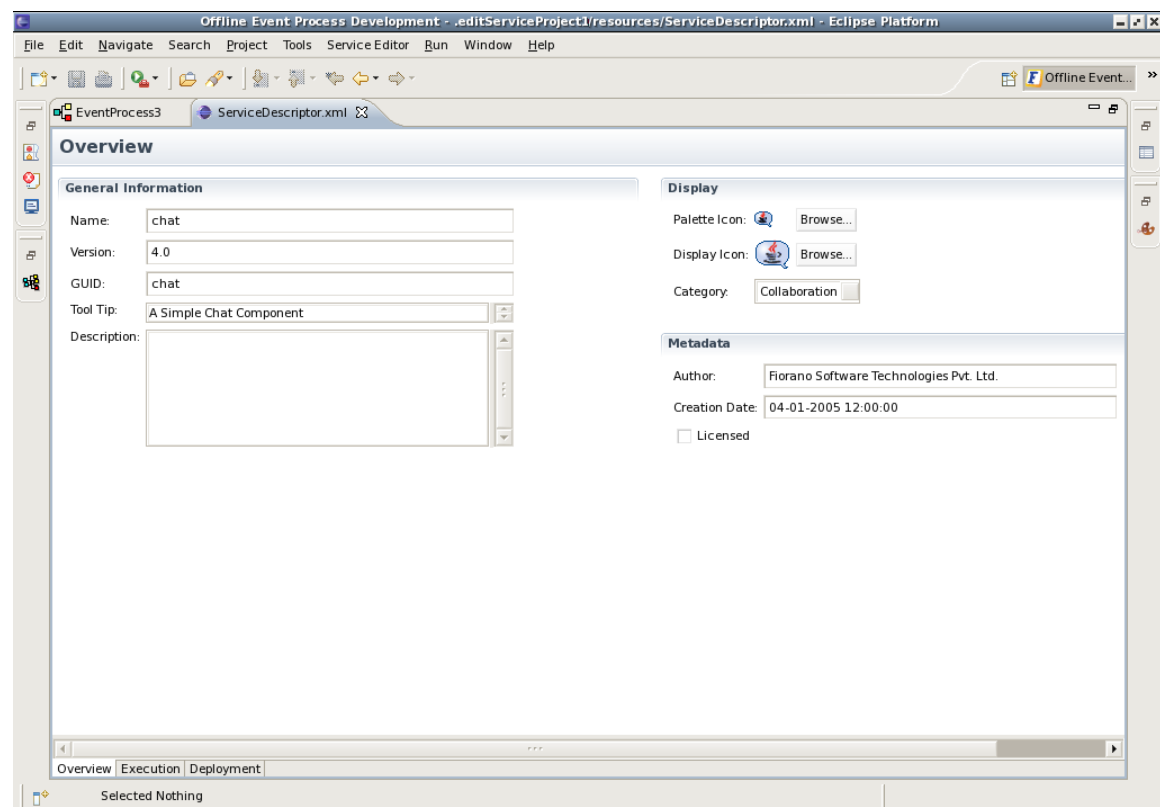


Figure 2.1.9: Service Descriptor Editor

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.

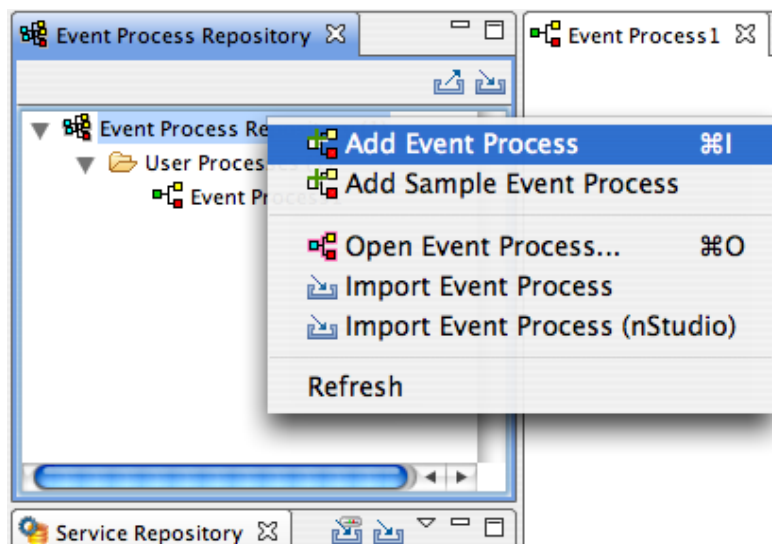


Figure 2.2.1: Creating new Event Process

- 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.

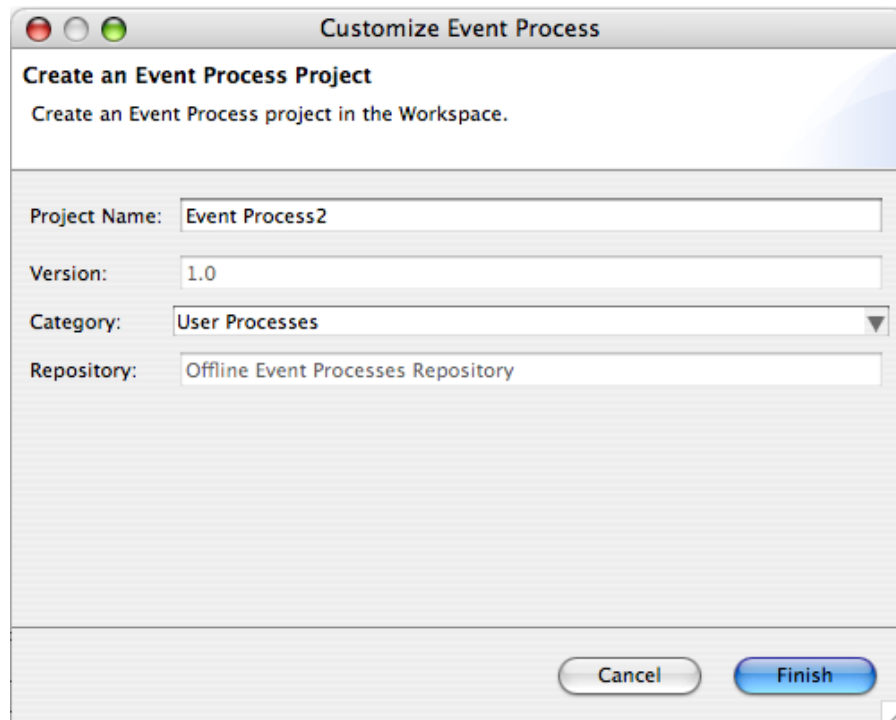


Figure 2.2.2: Customize Event process

- 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.

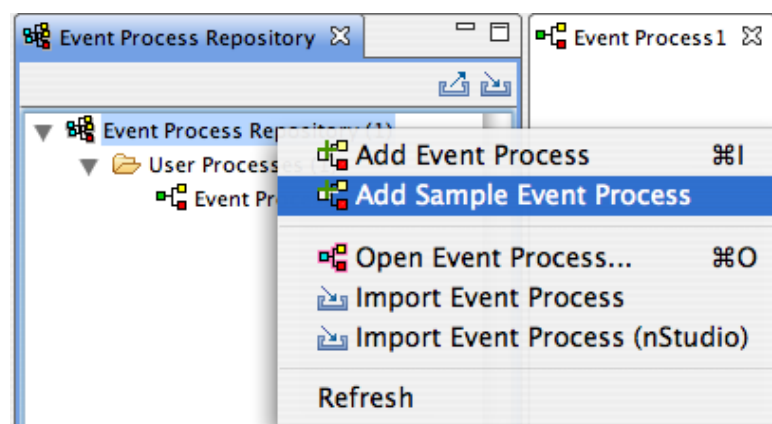


Figure 2.2.3: Add Sample Event Processes

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.

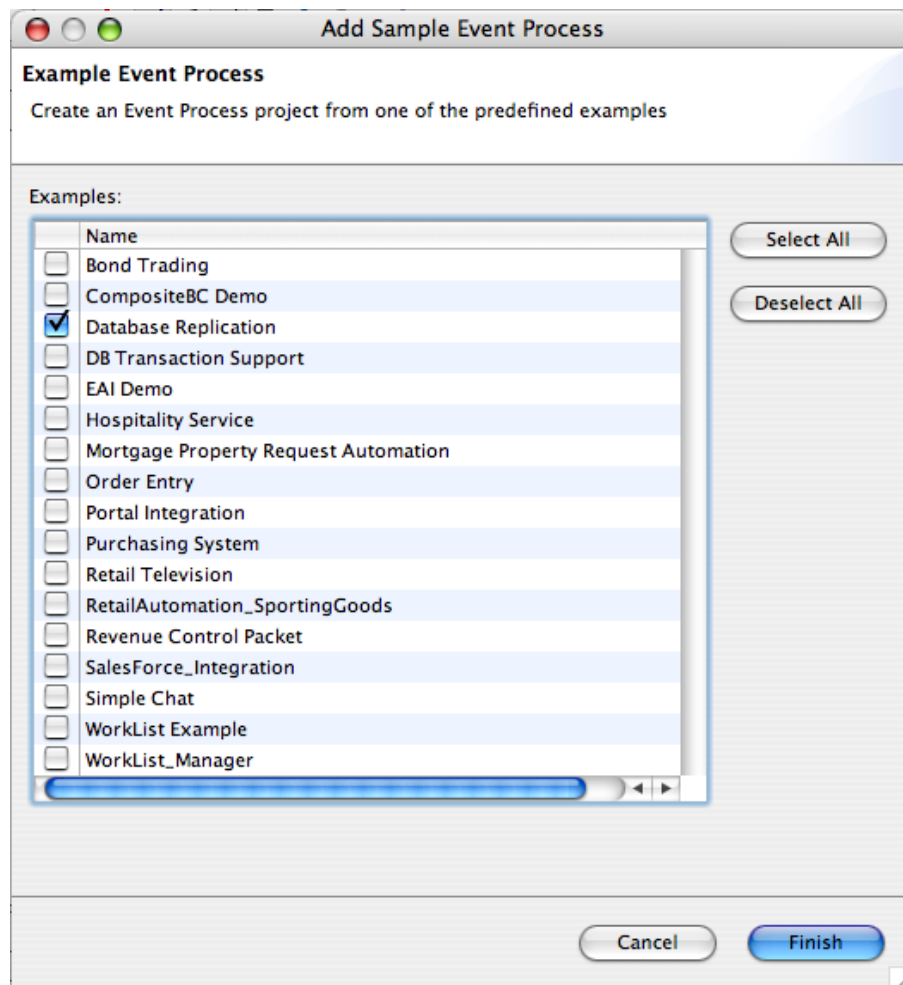


Figure 2.2.4: Sample Event Processes

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 in 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). The Export dialog box appears.

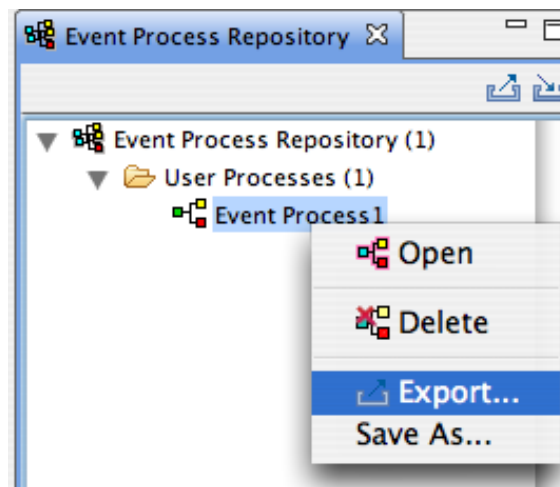


Figure 2.2.5: Export Event Process to local disk

2. Specify the file name and location to save and click **OK**. The Event Process project will be saved as a .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 2.2.6. 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](#).

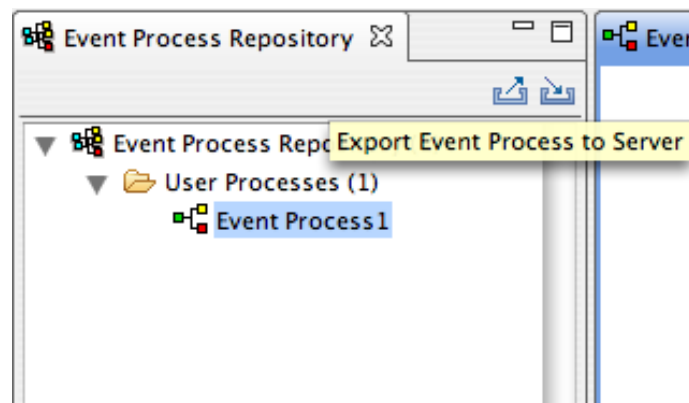


Figure 2.2.6: Export Event Process to Server

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

Select **Overwrite if exists** option if the Event Process with same GUID is already present in the Server.

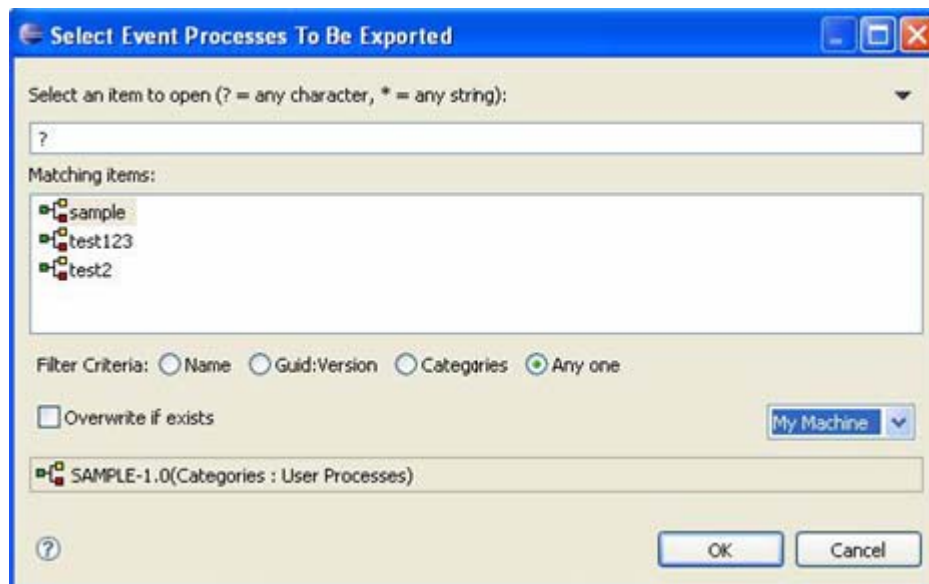


Figure 2.2.7: Select Event Process to be exported

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 2.2.8.

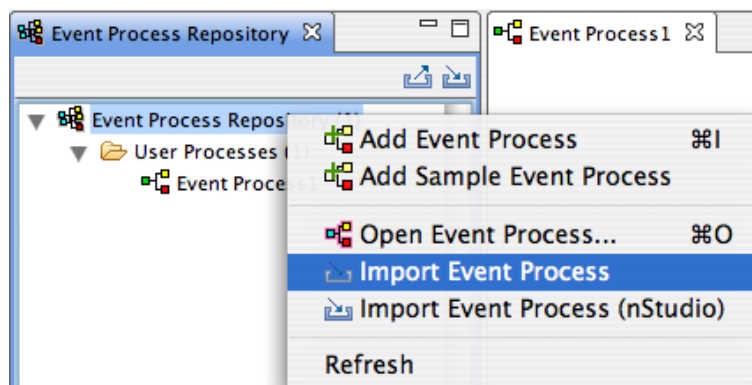


Figure 2.2.8: Import Event Process from local disc

2. Specify the location of Event Process zip file and click **OK**. Event Process project will be imported to the Event Process Repository.

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 2.2.9.

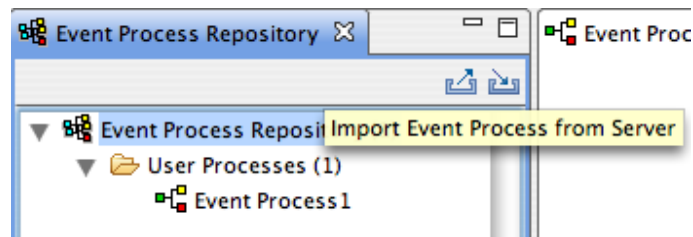


Figure 2.2.9 Import Event Process from Server

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

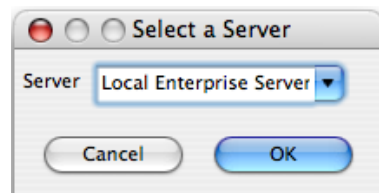


Figure 2.2.10: Select Enterprise Server

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 2.2.11.

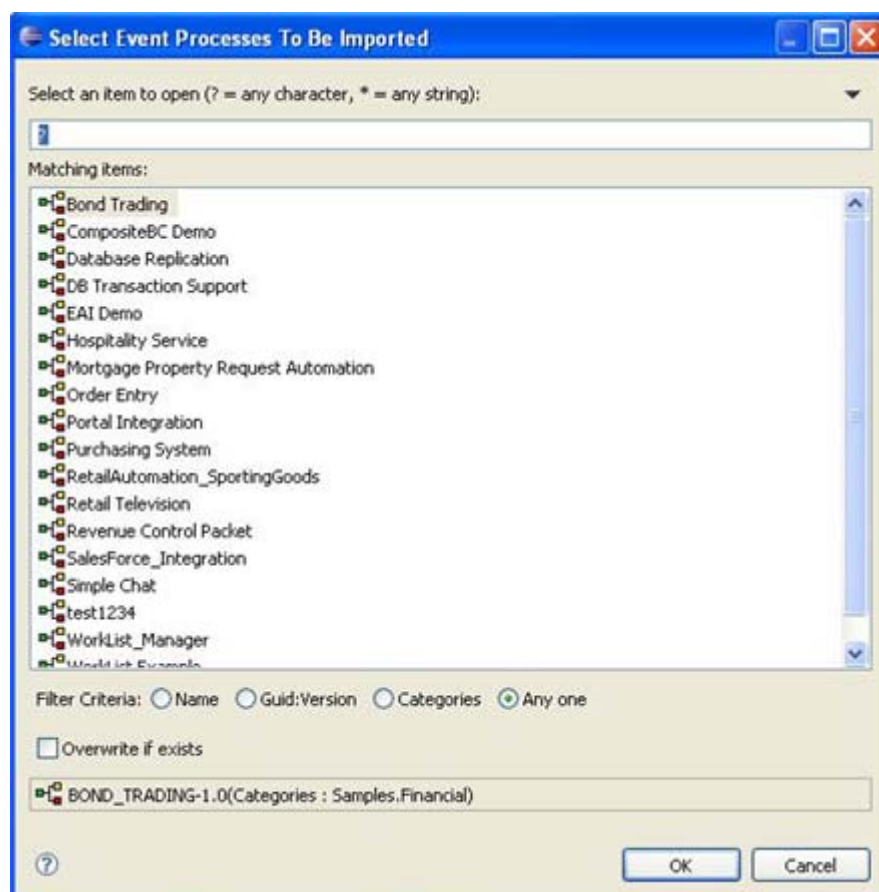


Figure 2.2.11: List of Event Process in server

3. Select the Event Process to be imported and click **OK**. Event Process project will be imported to the Event Process Repository.

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.

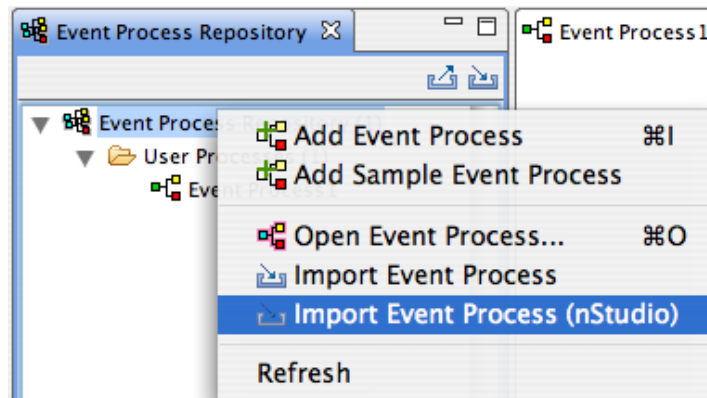


Figure 2.2.12: Import nStudio Event Process

Selecting the import option opens an Import Wizard as shown in Figure 2.2.13.

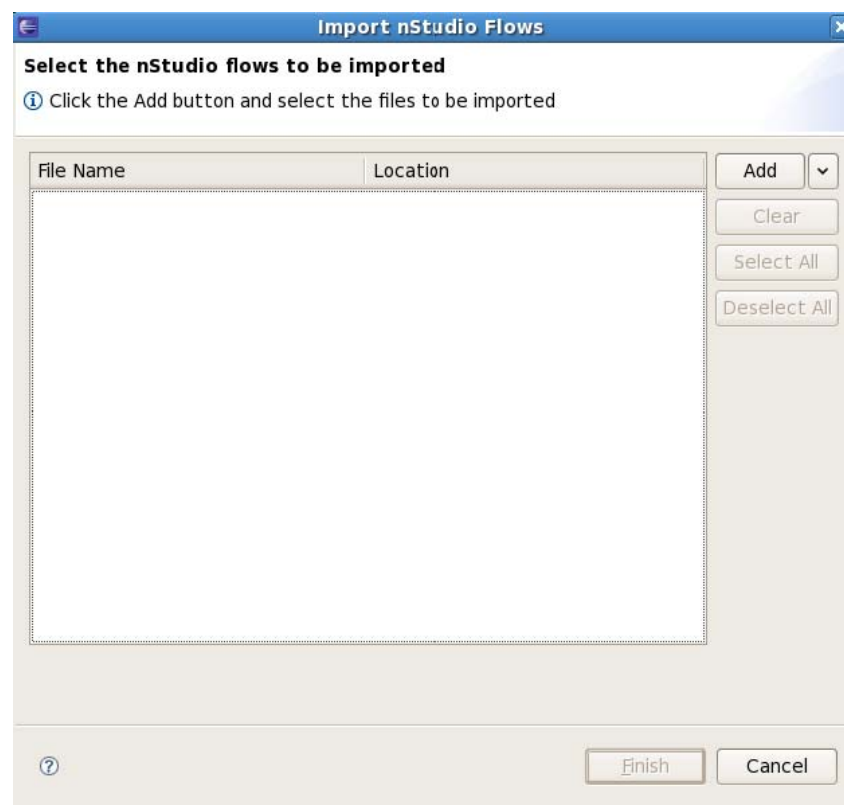


Figure 2.2.13: Import nStudio Flows

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 2.2.14:

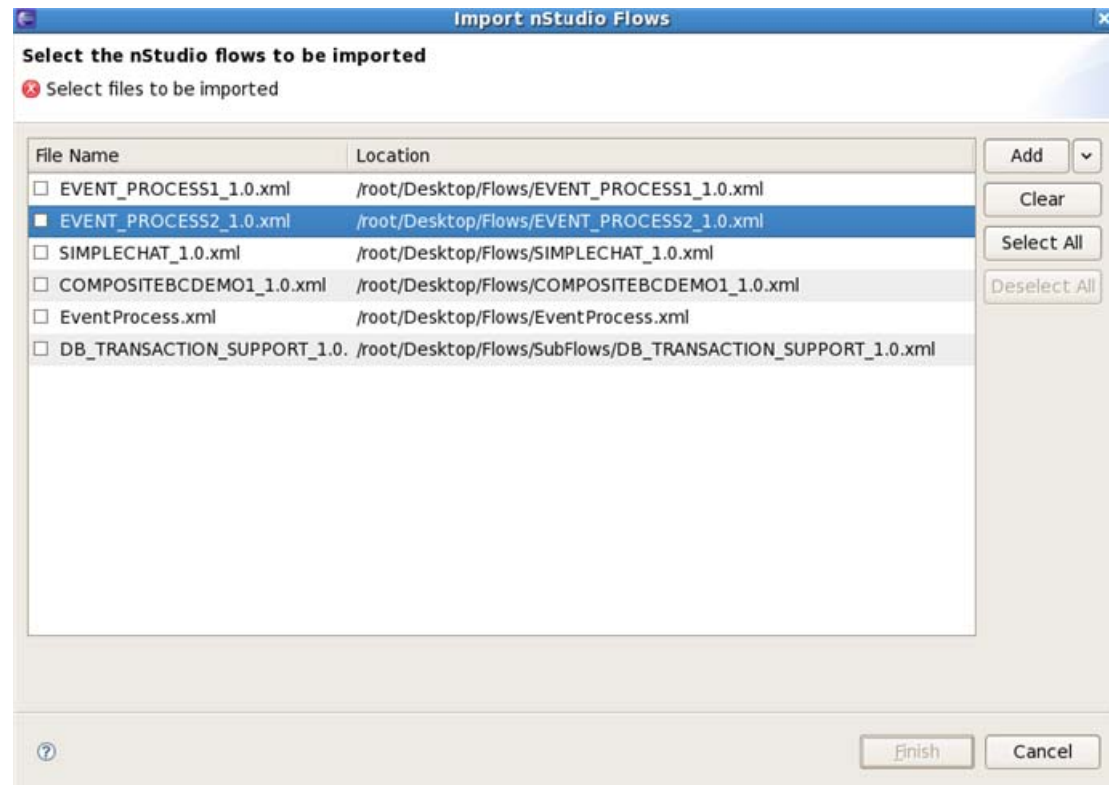


Figure 2.2.14: Select nStudio flows to import

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.

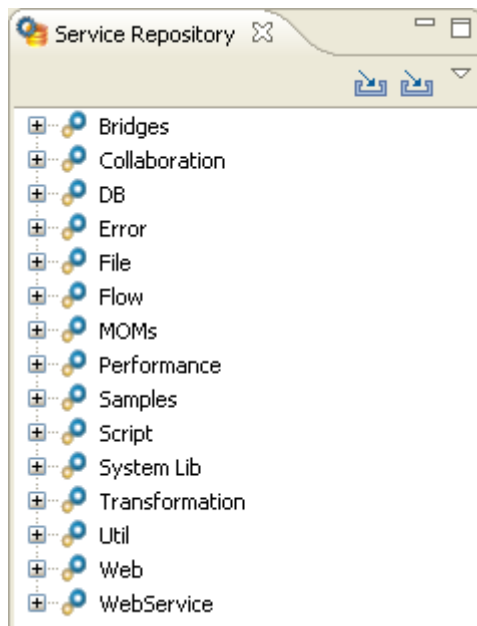


Figure 2.3.1: Service repository

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.

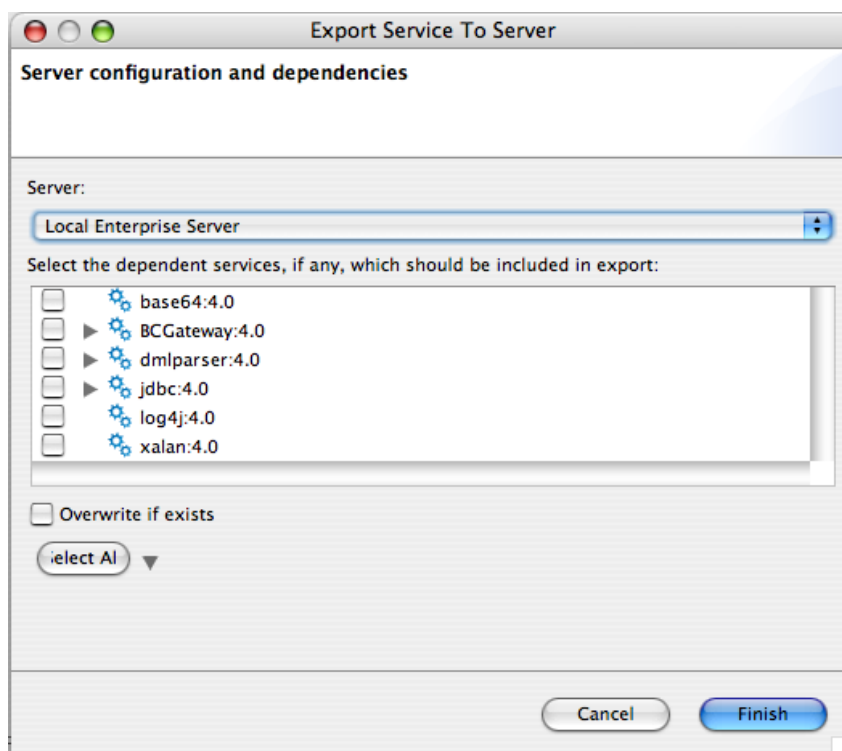


Figure 2.3.2: Export service to server

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.

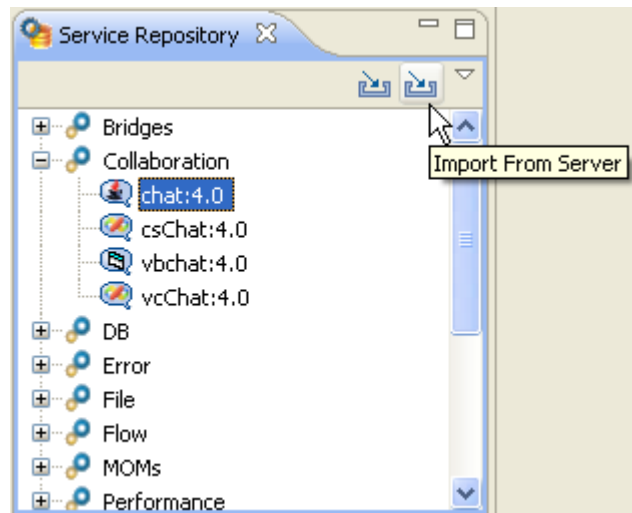


Figure 2.3.3: Import from Server option

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

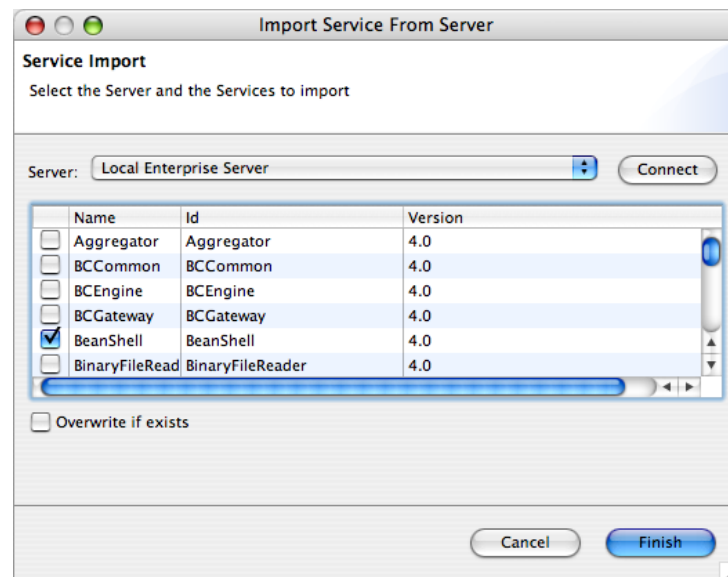


Figure 2.3.4: Import service from server

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.

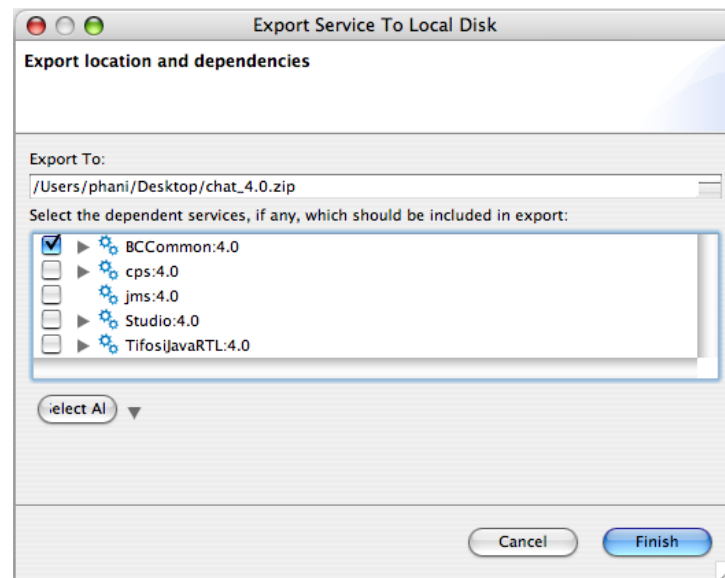


Figure 2.3.5: Export service to Local Disk

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.

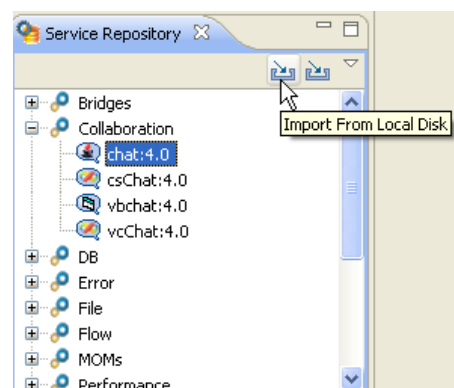


Figure 2.3.6: Import from Local Disk button

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.

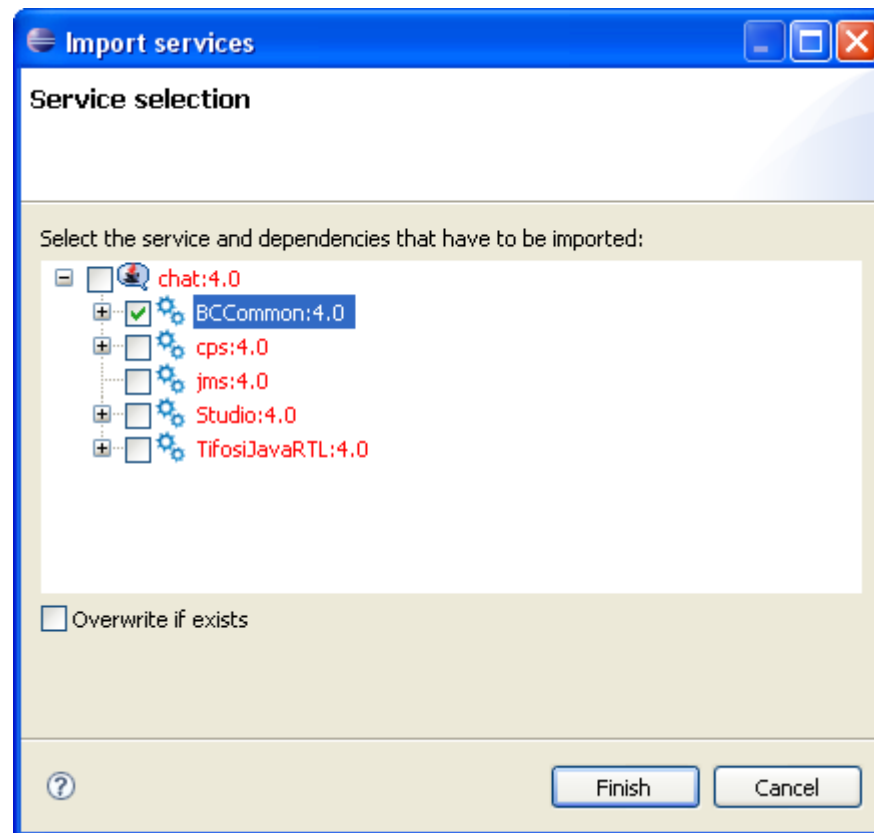


Figure 2.3.7: Import services dialog box

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.

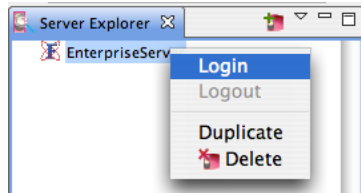


Figure 3.1.1:Enterprise Server node

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

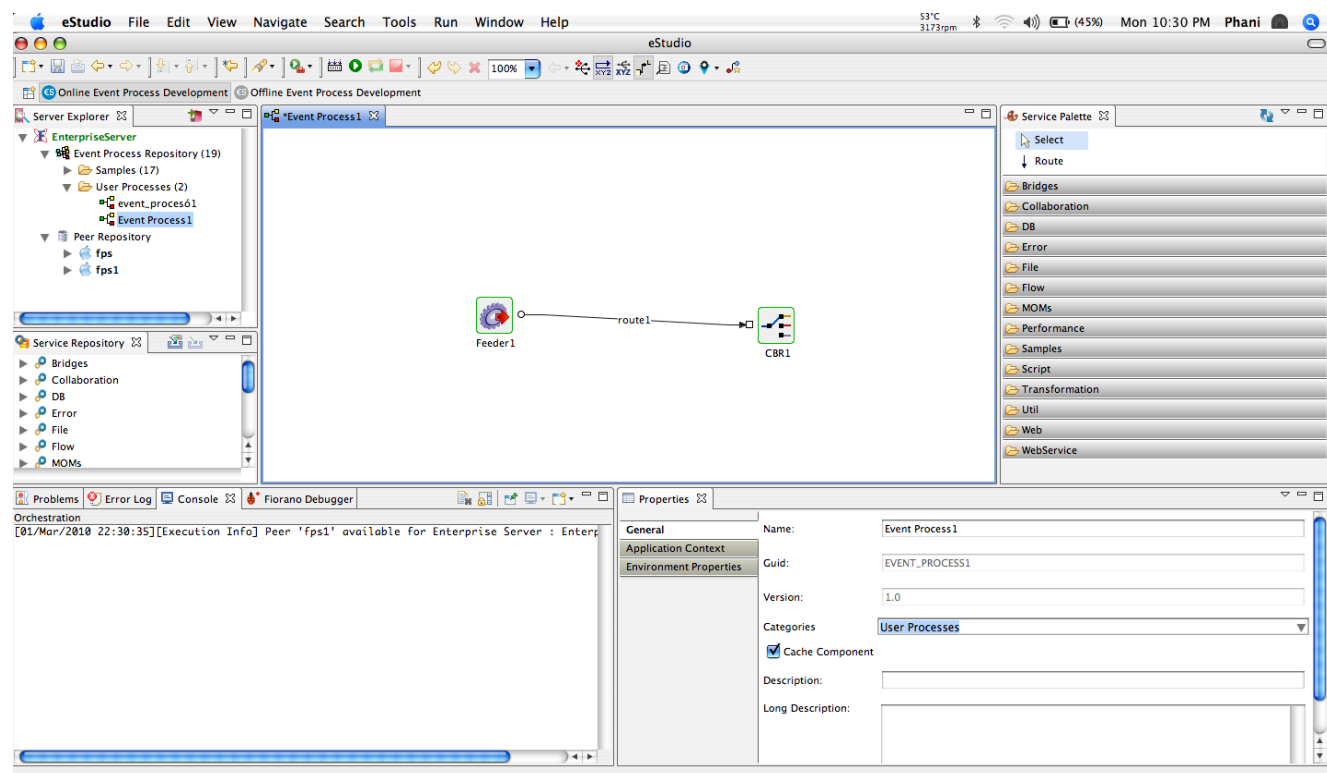


Figure 3.1.2: Online Event Process Development perspective

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 to 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.

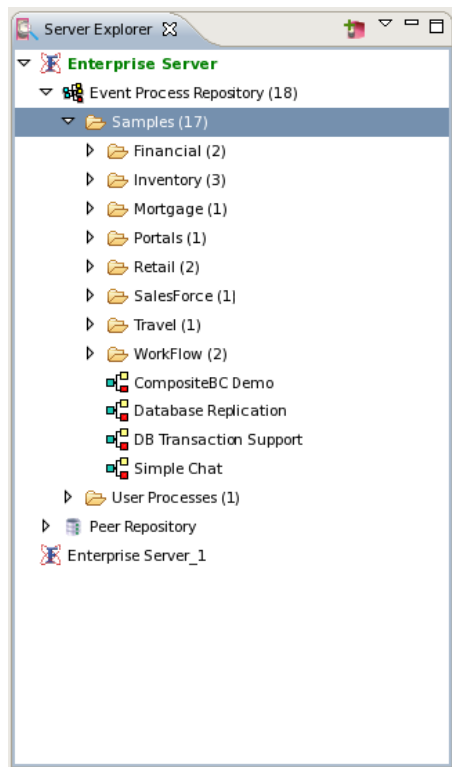


Figure 3.1.3: Server Explorer

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.

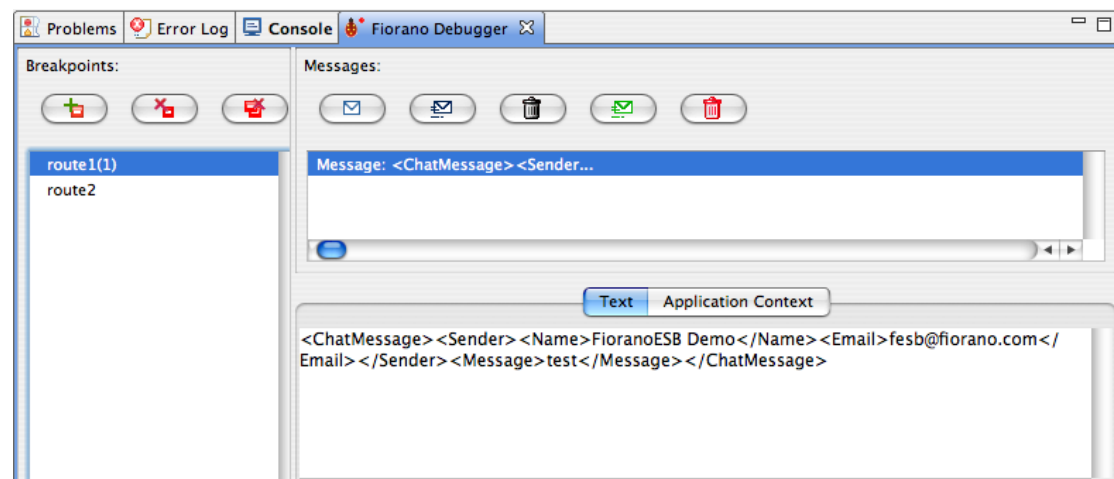


Figure 3.1.4: Fiorano Debugger view

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.

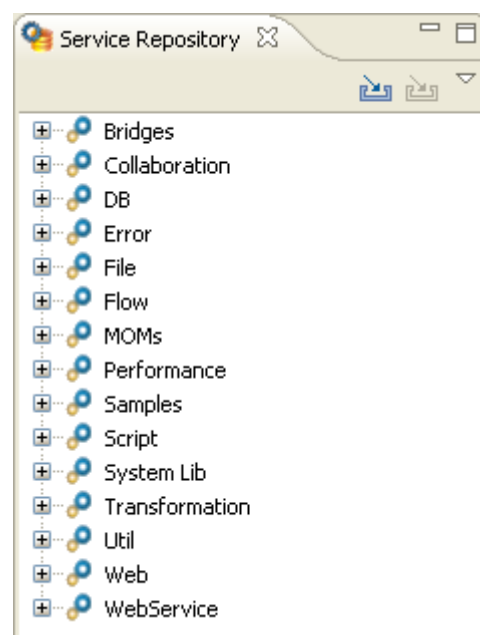


Figure 3.2.1: service repository (Online Event Process Development perspective)

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.

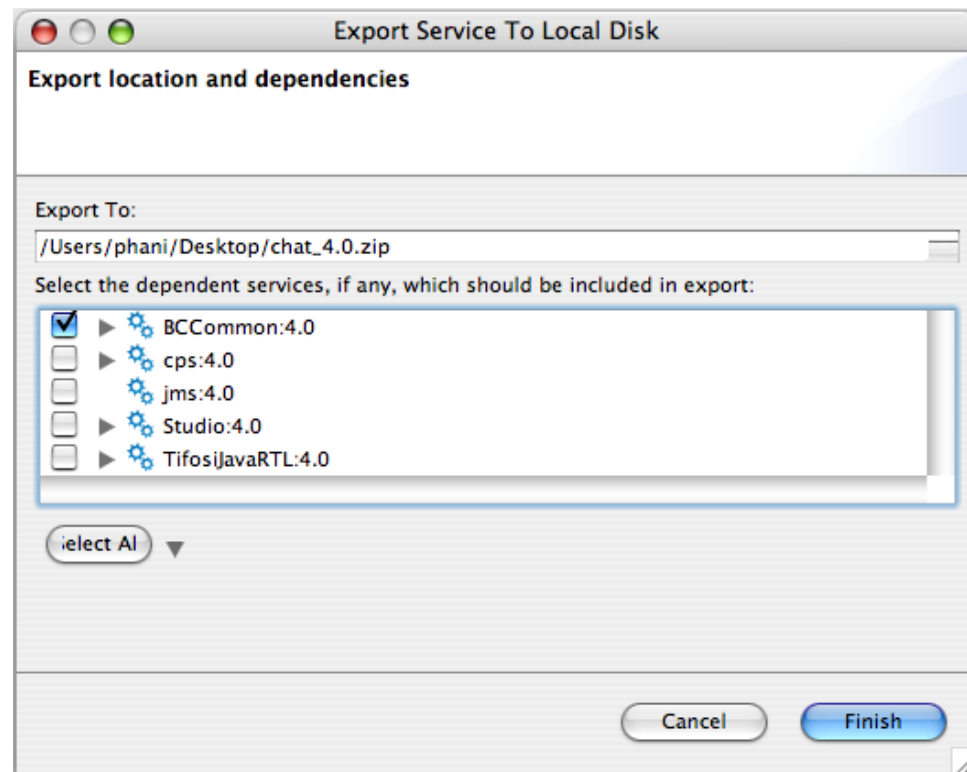


Figure 3.2.2: Export service to Local Disk

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.

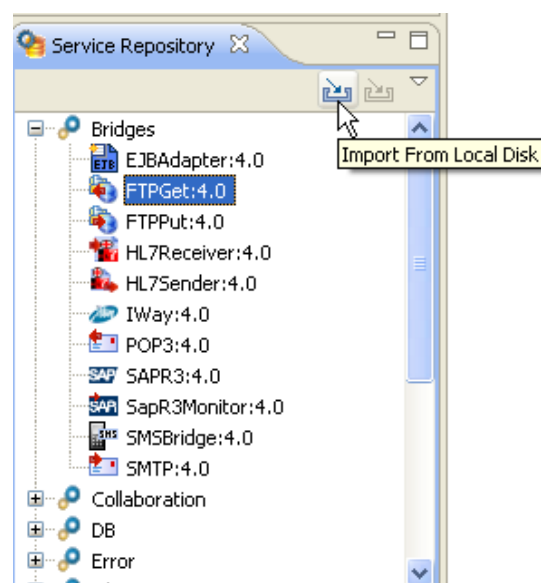


Figure 3.2.3: Import from Local Disk

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.

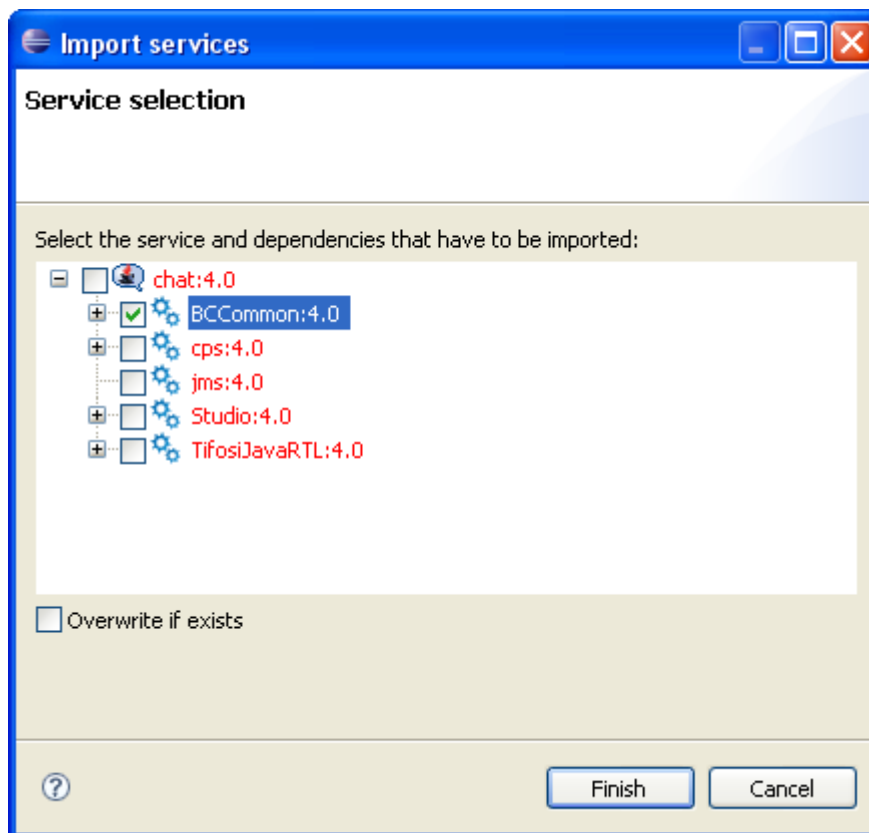


Figure 3.2.4: Import services dialog

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.

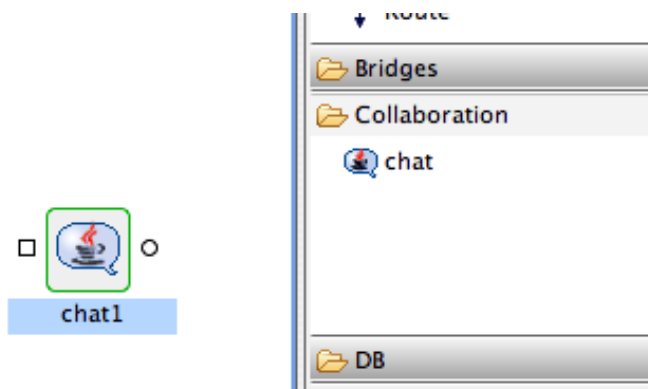


Figure 5.1.1: Adding components

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.

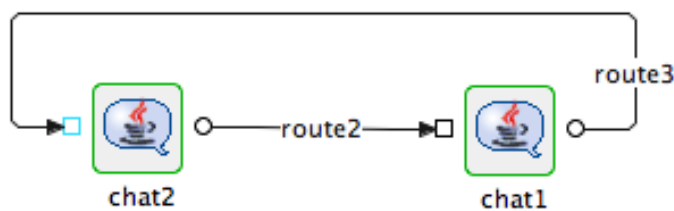


Figure 5.2.1: Connecting components through routes

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.

Managed Connection Factory

Configuration is Valid

Show Expert Properties

Connection Properties

Database: Oracle

Driver: oracle.jdbc.driver.OracleDriver

URL: jdbc:oracle:thin:@<hostname>:1521:orcl

User name: scott

Password: *****

Connection Properties: Click ... to edit

Auto Commit: ☐

Connection Pool Params: Click ... to edit

Advanced Settings

Query timeout (in seconds): 60 second(s)

Fetch size: 200 fetch size

Connection ping sql: select * from dual

Enable JDBC Driver Logging: ☐

Description: Place cursor on a property to view its description

Test

< Back Next > Finish Cancel Save and Close

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

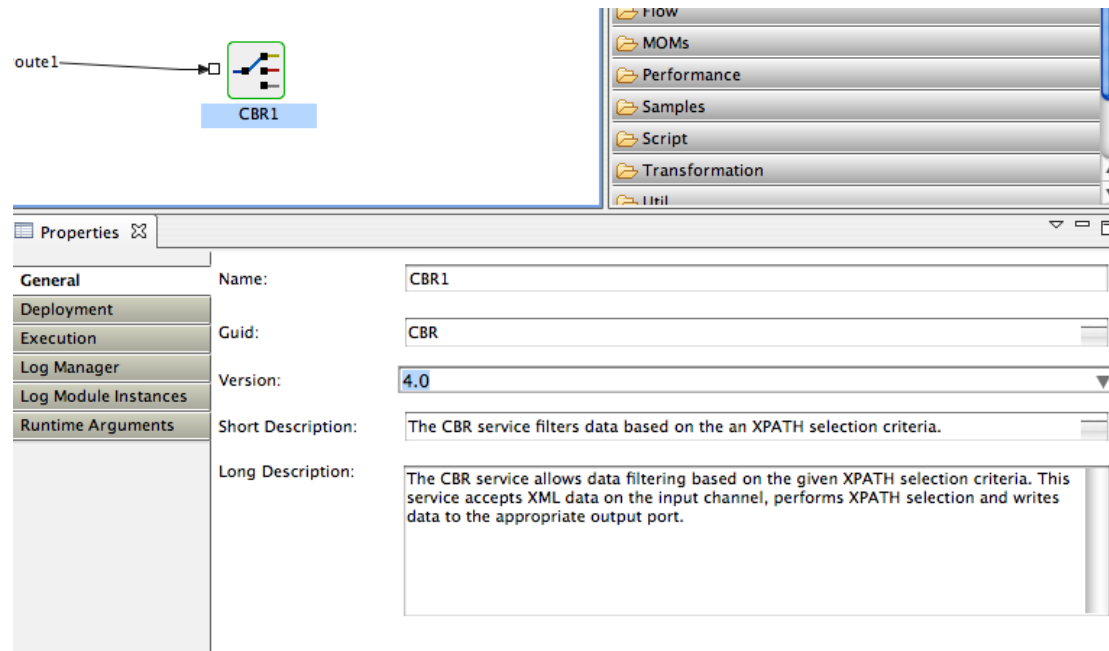


Figure 5.4.1: Component properties

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.

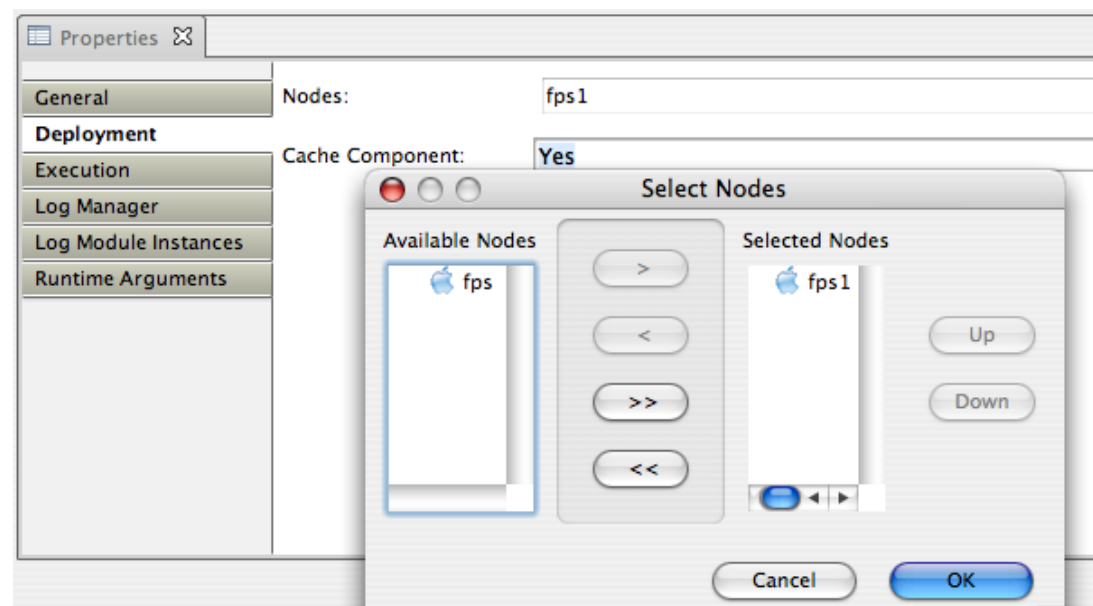
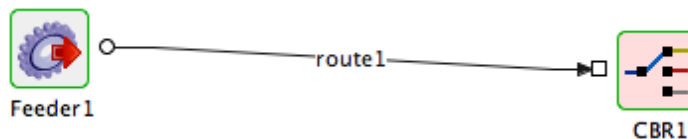


Figure 5.4.2: Component ports

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.

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.

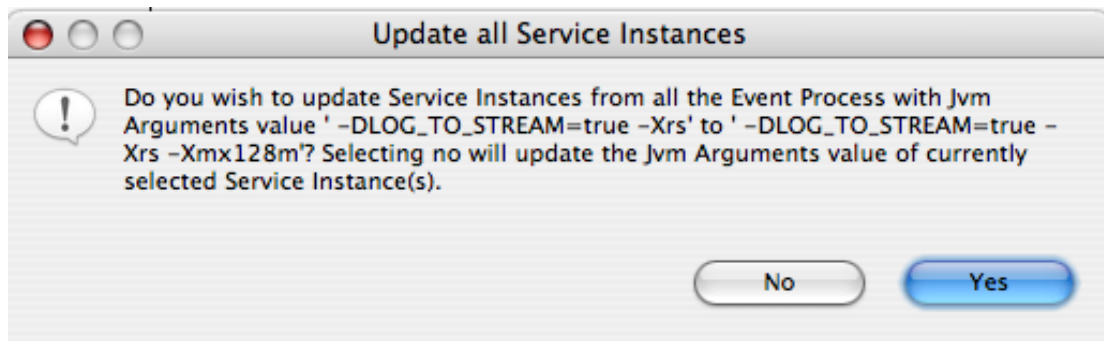


Figure 5.4.4: Update all Service Instances dialog

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.

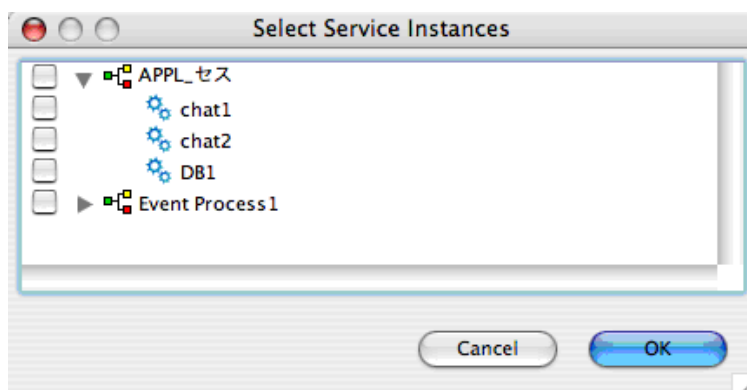


Figure 5.4.5: Select Service Instances

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**.

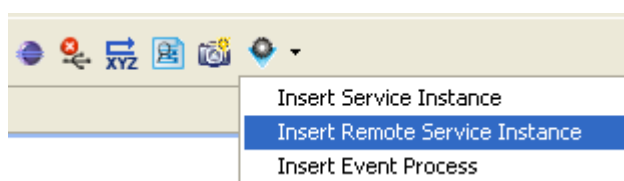
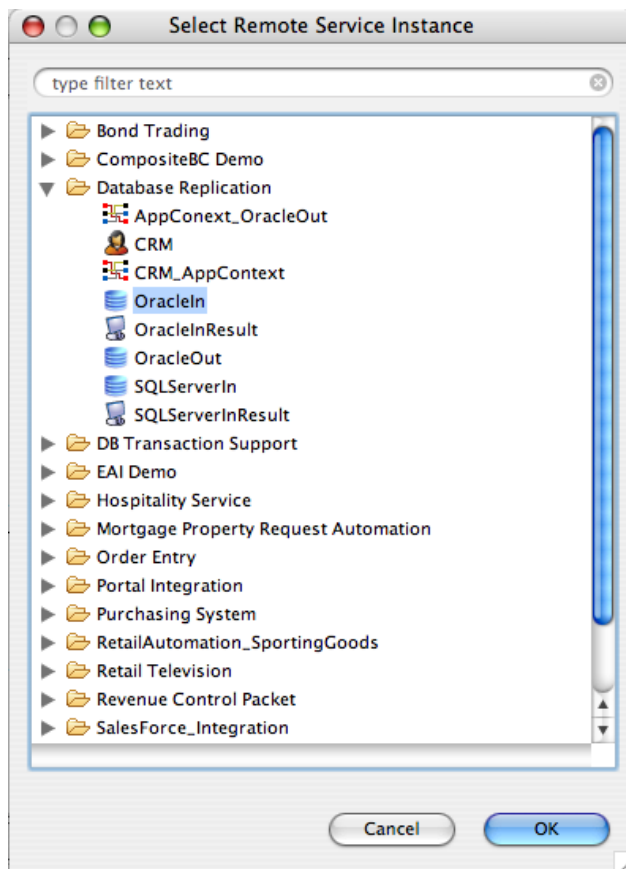


Figure 5.5.1: Insert Remote Service Instance option

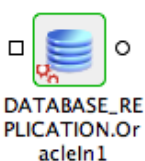
The **Select Remote Service Instance** wizard starts, as shown in Figure 5.5.2.

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.

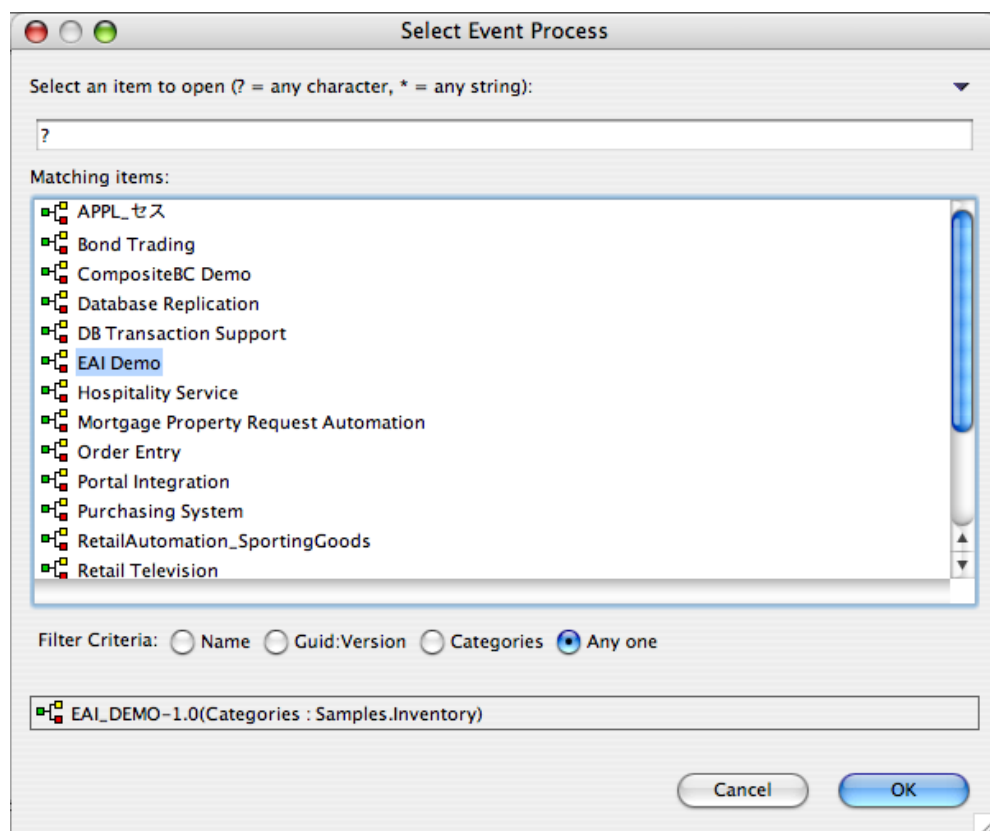


Figure 5.6.2: Select Event Process dialog

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

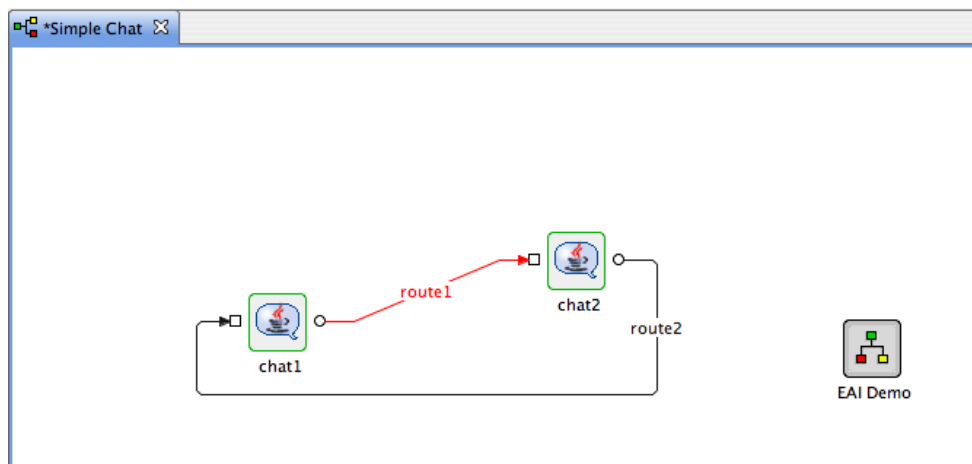


Figure 5.6.3: Inserted Event process

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.

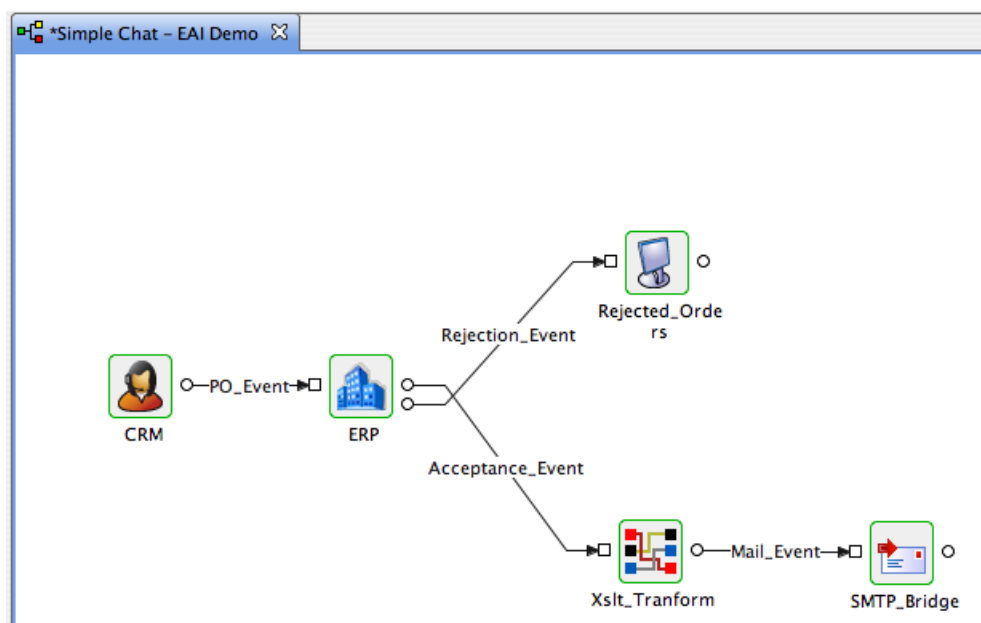


Figure 5.6.4: Component ports

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.

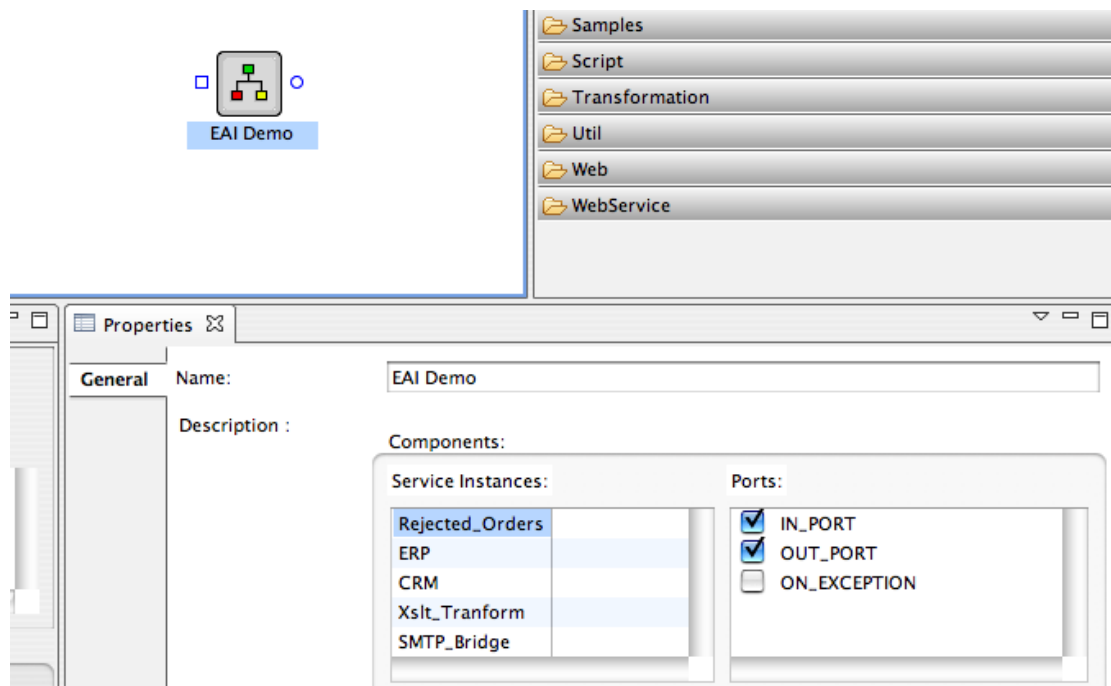


Figure 5.6.5: Properties tab

- 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.

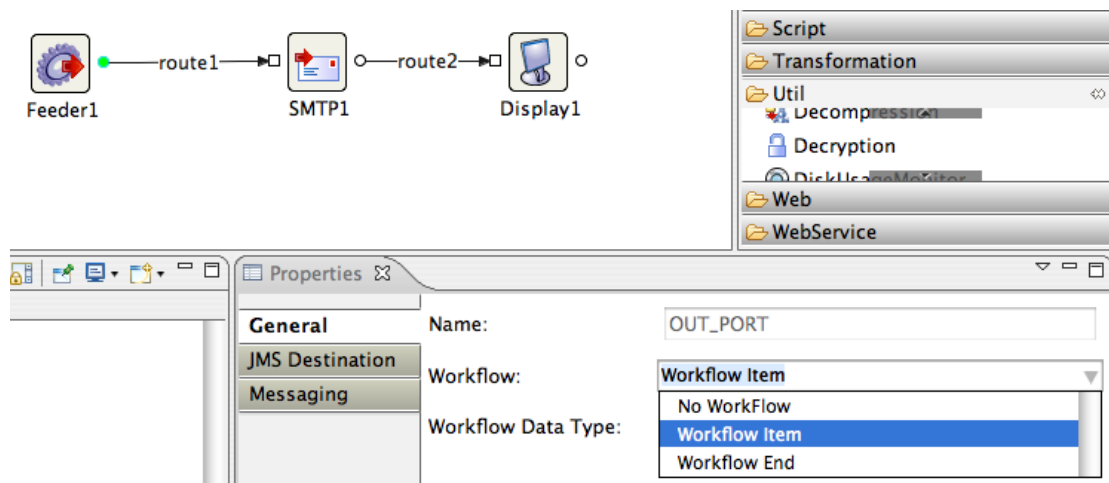


Figure 5.7.1: Enabling Document Tracking

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.

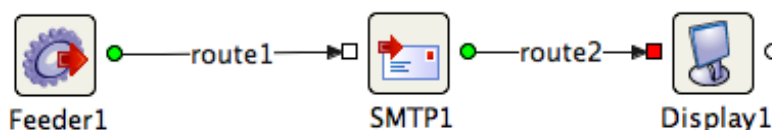


Figure 5.7.2: Event Process with Document Tracking enabled

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.

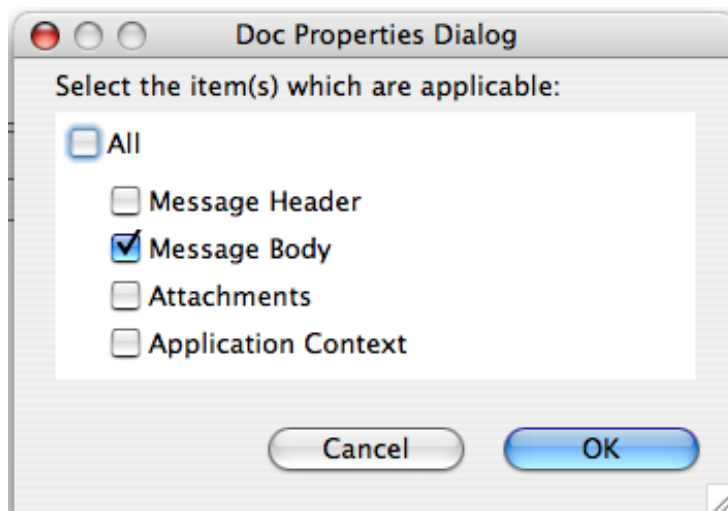


Figure 5.7.3: Document Tracking Properties

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 two ways, defining a Mapper Project or by providing a Custom XSL. 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.

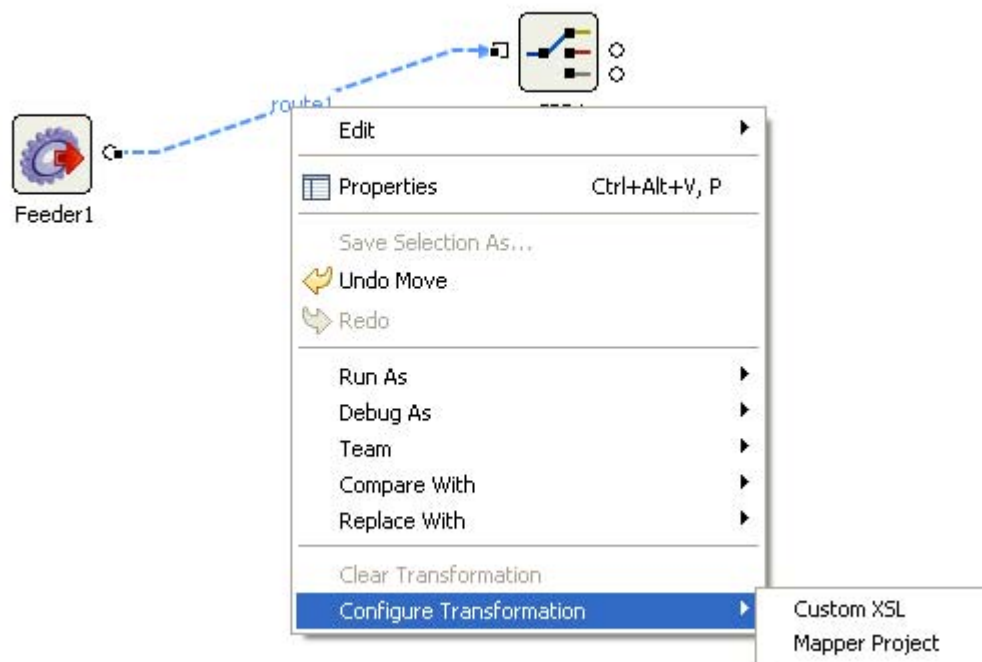


Figure 5.8.1: Route Transformation

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 Funclet view.

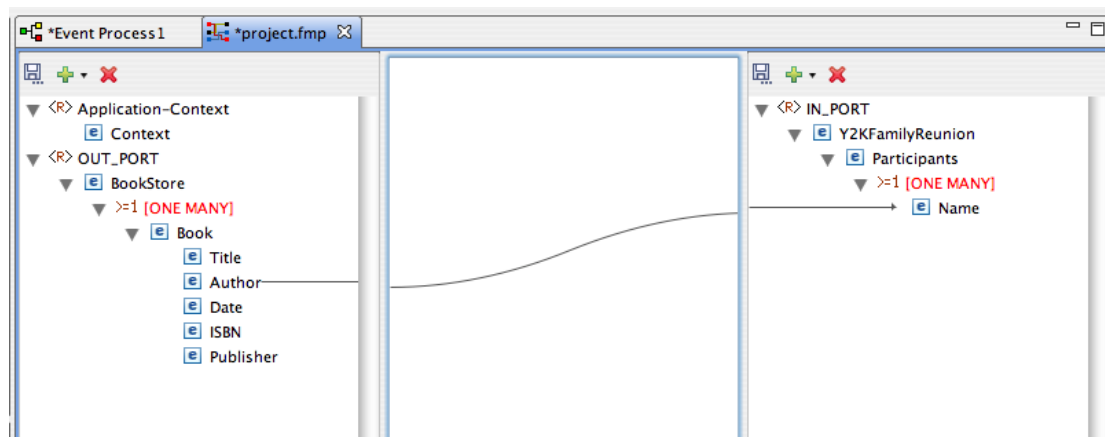


Figure 5.8.2: Route transformation editor

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.

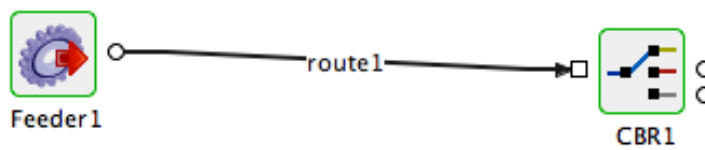


Figure 5.8.3: Route with transformation set

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) can be specified from the drop-down in the top right corner.
3. The XSL provided can be tested by clicking the **Test** button.

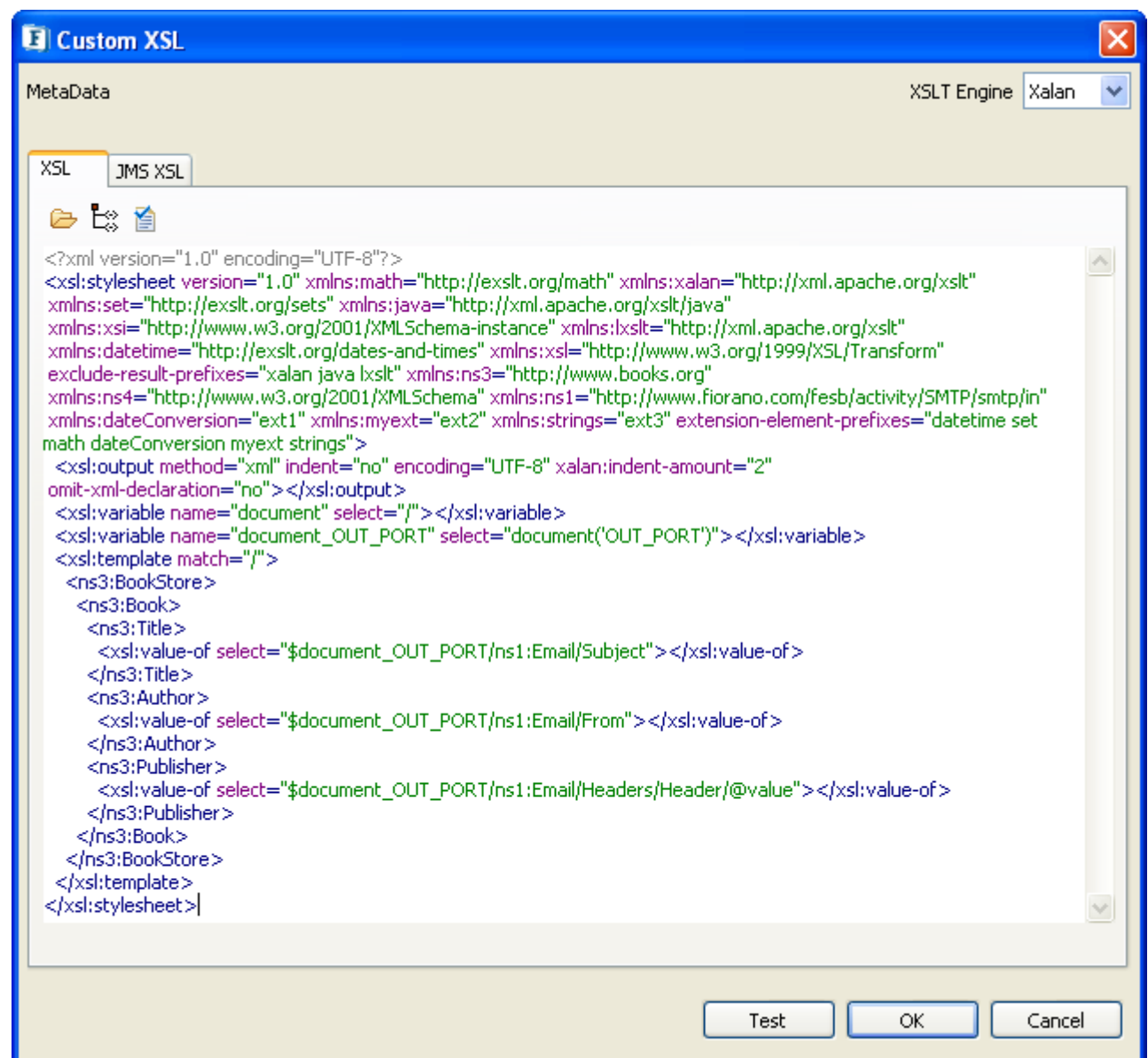


Figure 5.8.4: Custom XSL Dialog

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.

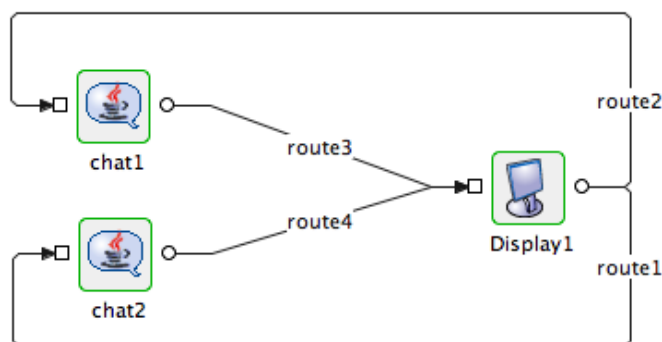


Figure 5.9.1: Out-port of chat1 and chat 2 to display in-port, out-port of display to in-port of Chat1 and chat2

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.

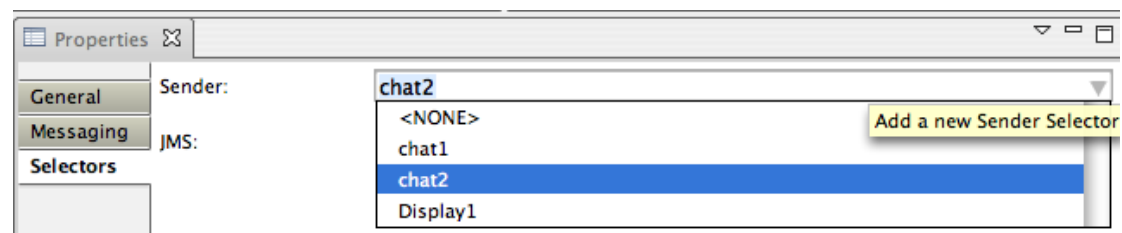


Figure 5.9.2: Configuring Selectors on route

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.

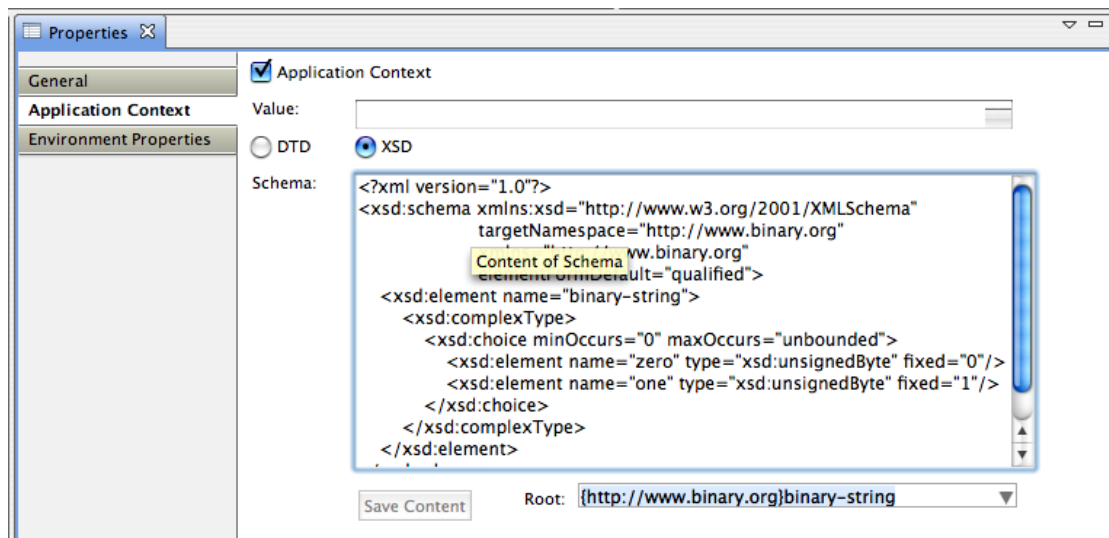


Figure 5.10.1: Application Context option

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.

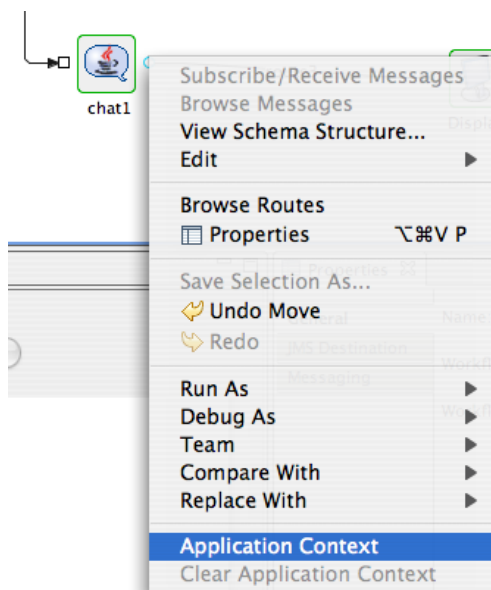


Figure 5.10.2: Application Context on output port

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.

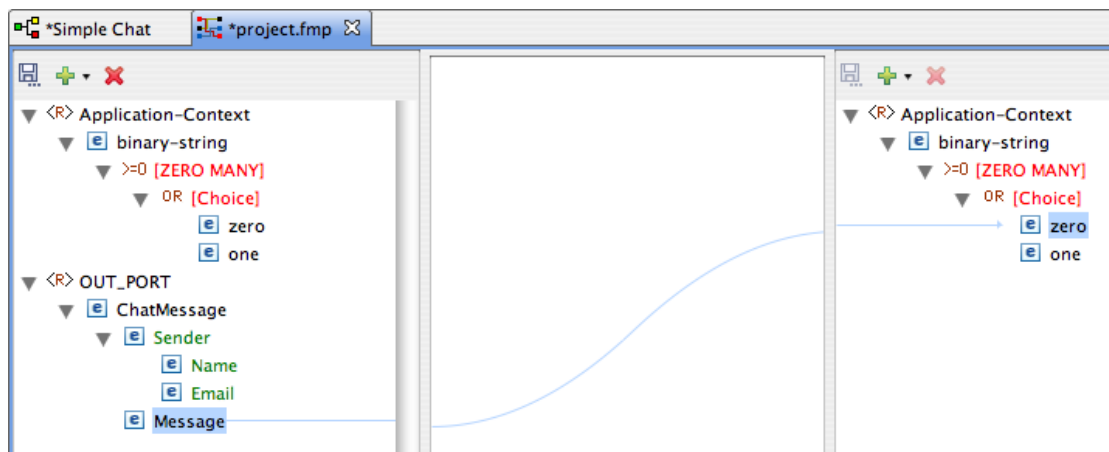


Figure 5.10.3: Mapper editor

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.

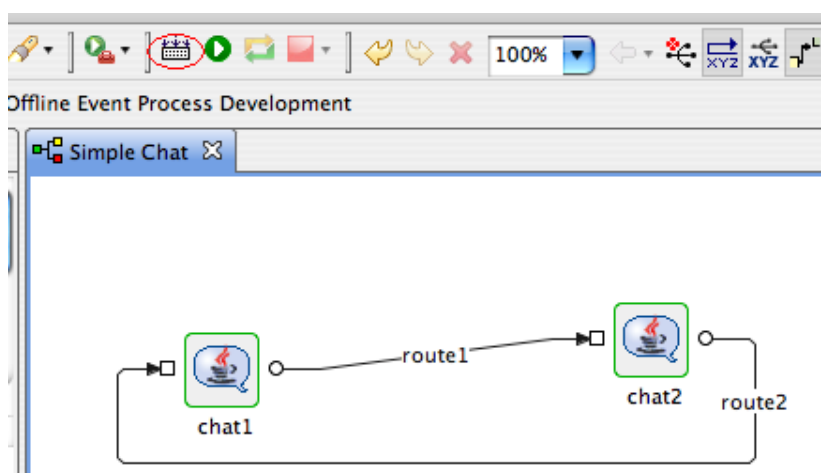


Figure 5.11.1: Check Resources and Connectivity

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.

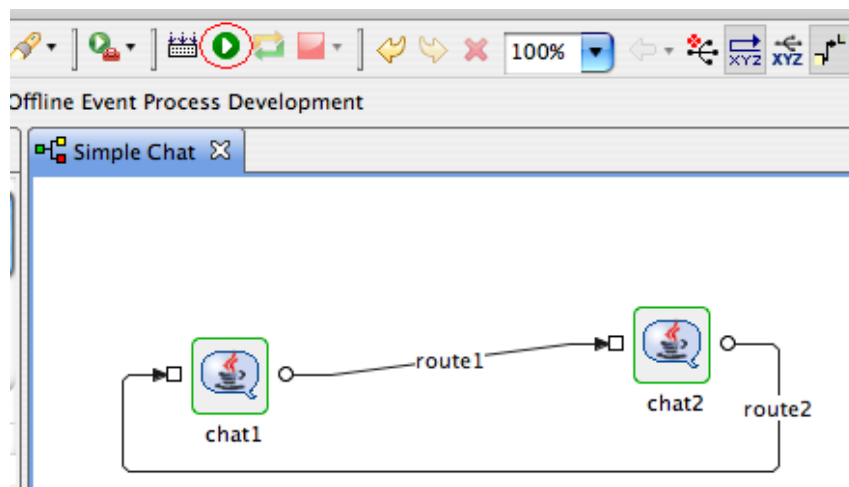


Figure 5.12.1: Launching an Event Process

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.

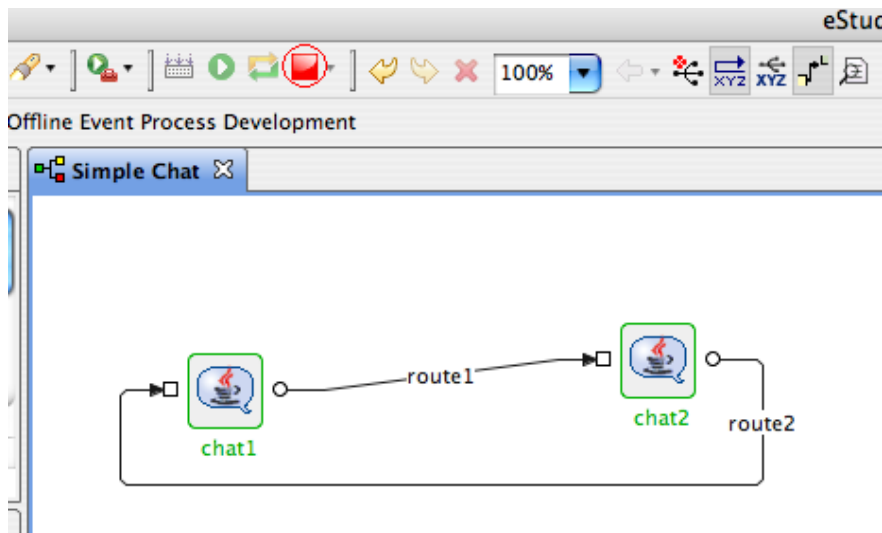


Figure 5.13.1: Stopping an Event Process

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.

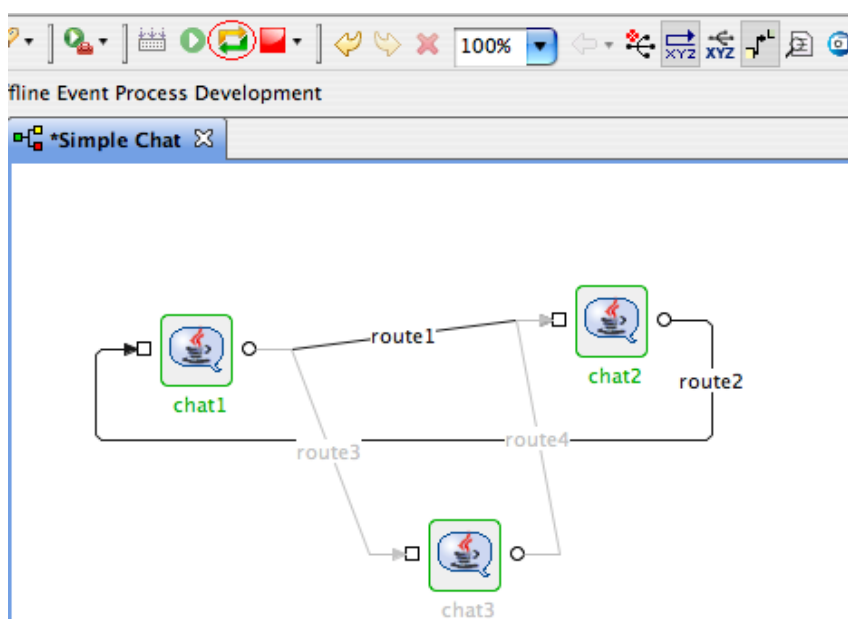


Figure 5.14.1: Event Process Synchronization

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.

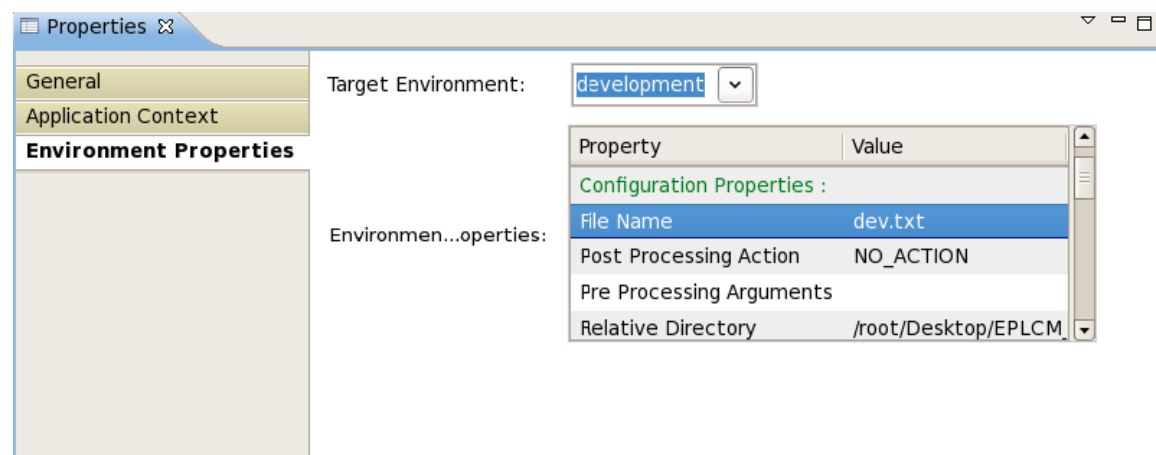


Figure 6.1.1: Environment Properties Tab

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:

4. 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.
5. 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.

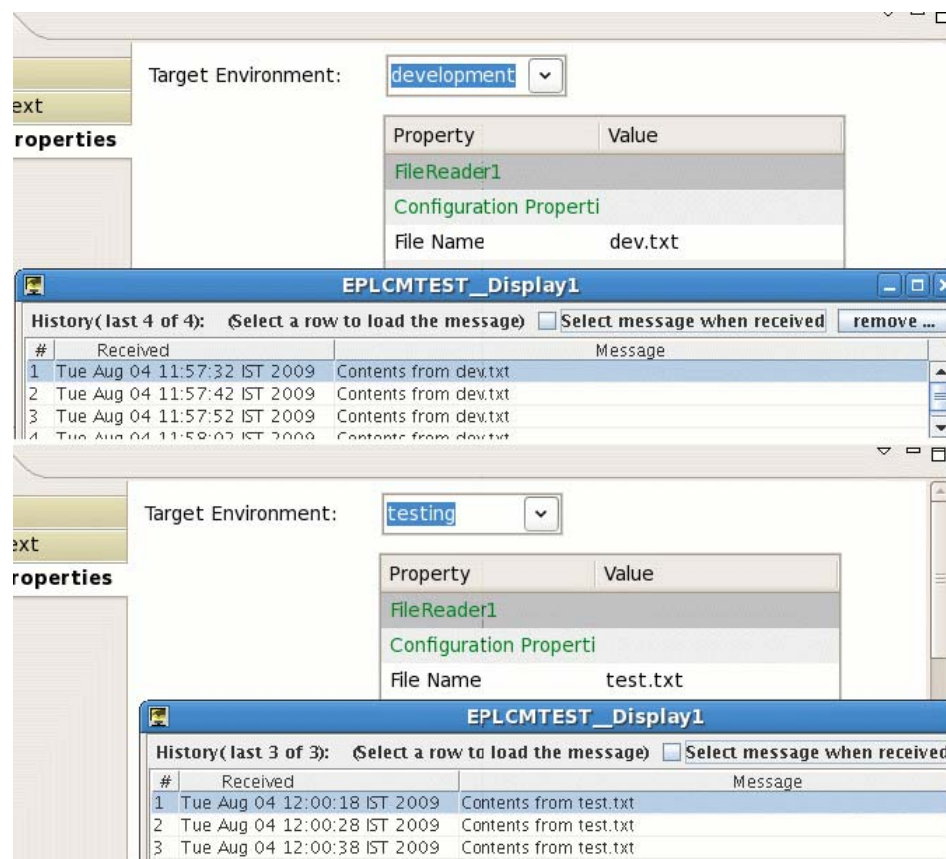


Figure 6.2.1: Display showing messages received from file reader in different environments

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.

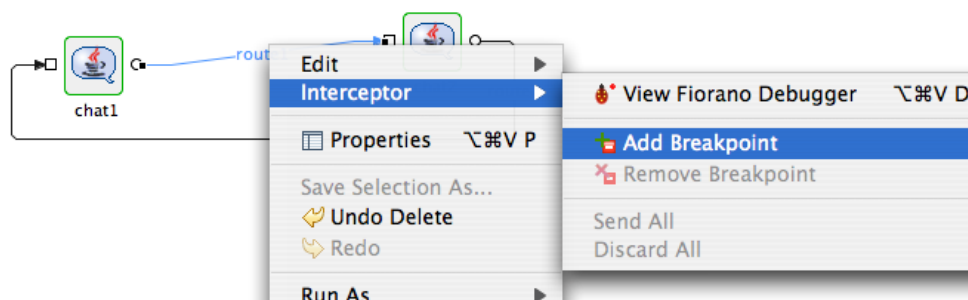


Figure 7.1.1: Adding breakpoint from context menu

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.

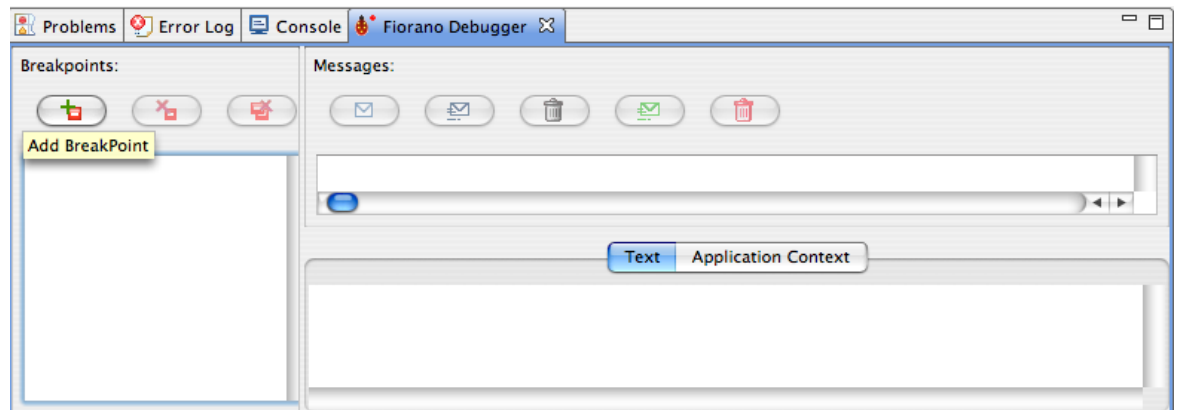


Figure 7.1.2: Adding break point from debugger view

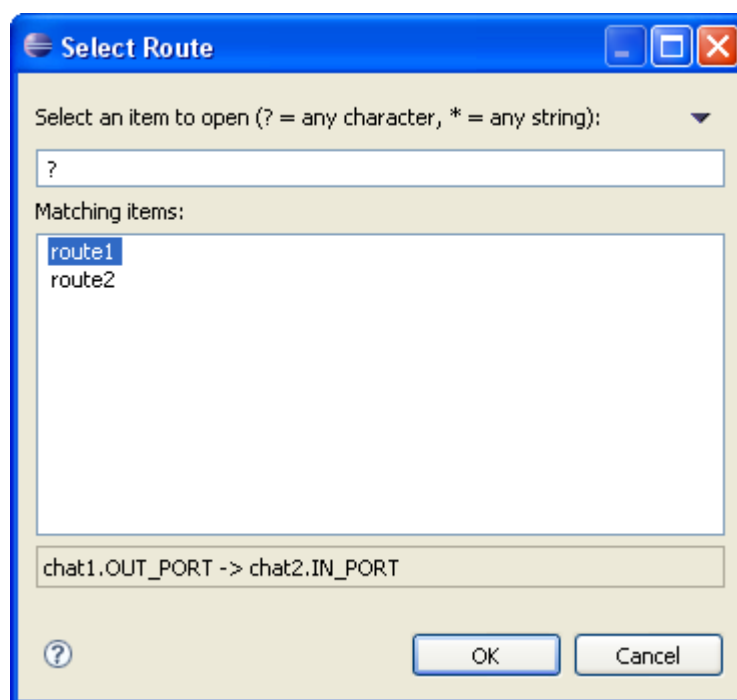


Figure 7.1.3: Select route to add breakpoint

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.

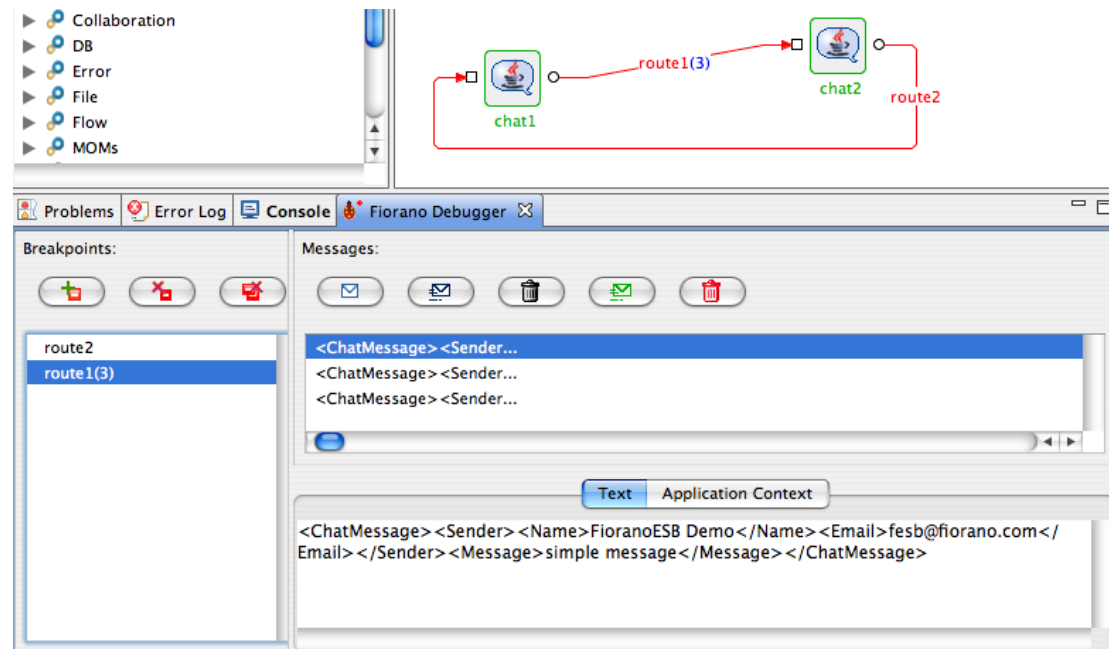


Figure 7.2.1: Message at breakpoint in Fiorano Debugger

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.

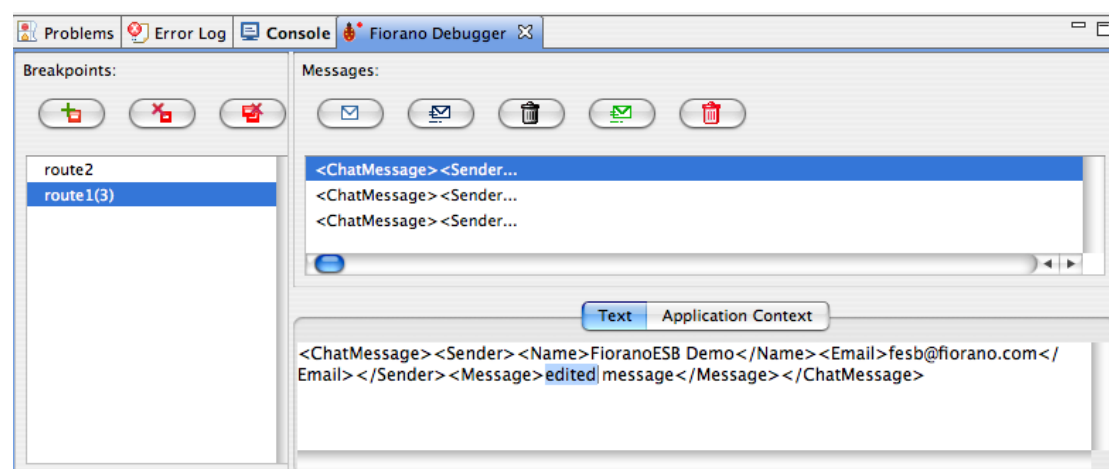


Figure 7.3.1: Edit message in Fiorano debugger

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.

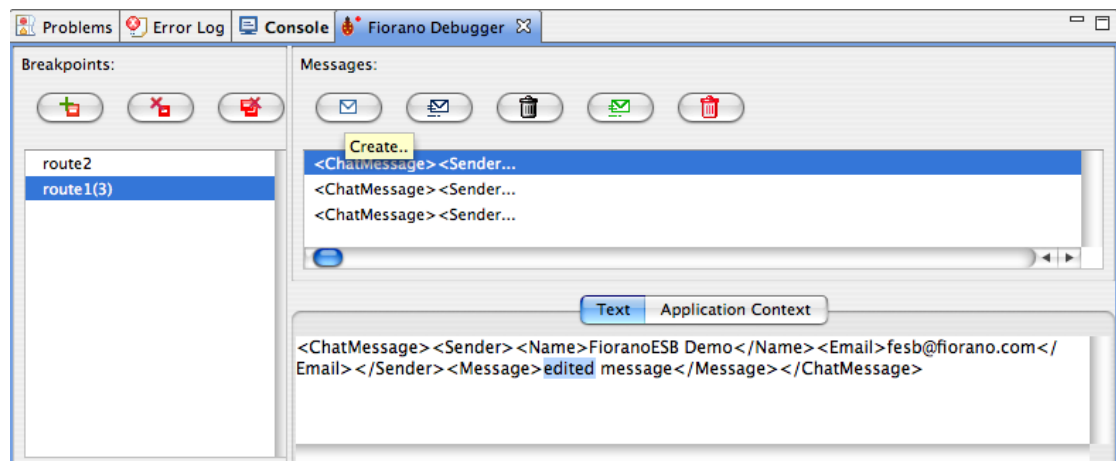


Figure 7.4.1: Create message in Fiorano debugger

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.

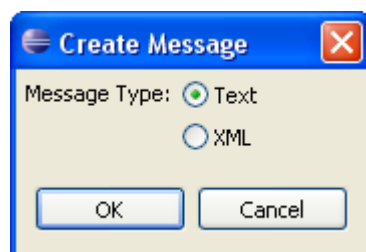


Figure 7.4.2 select type of new message

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.

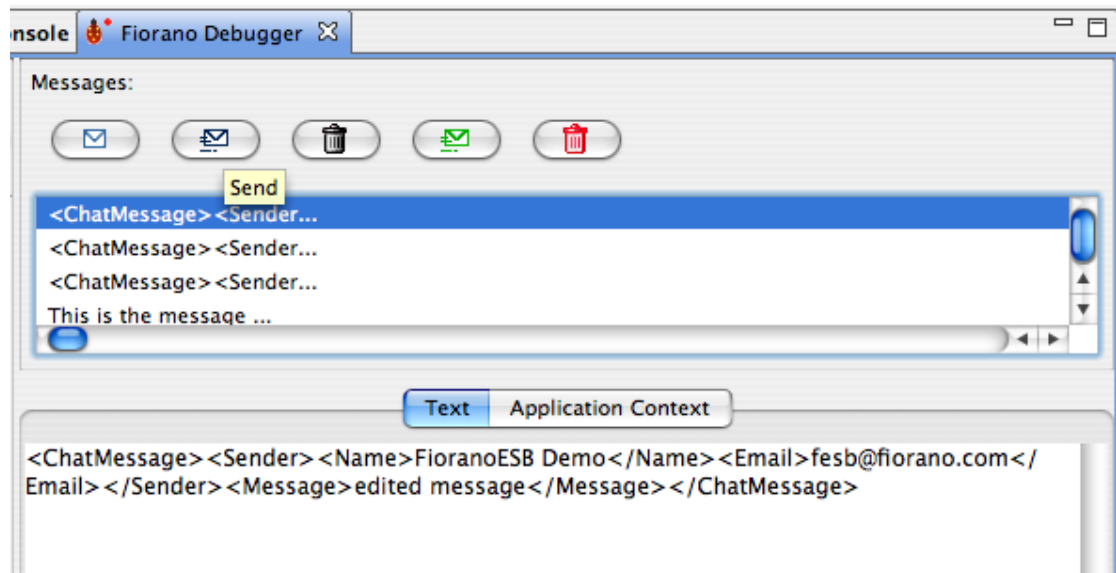


Figure 7.5.1: Send message in Fiorano debugger

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.

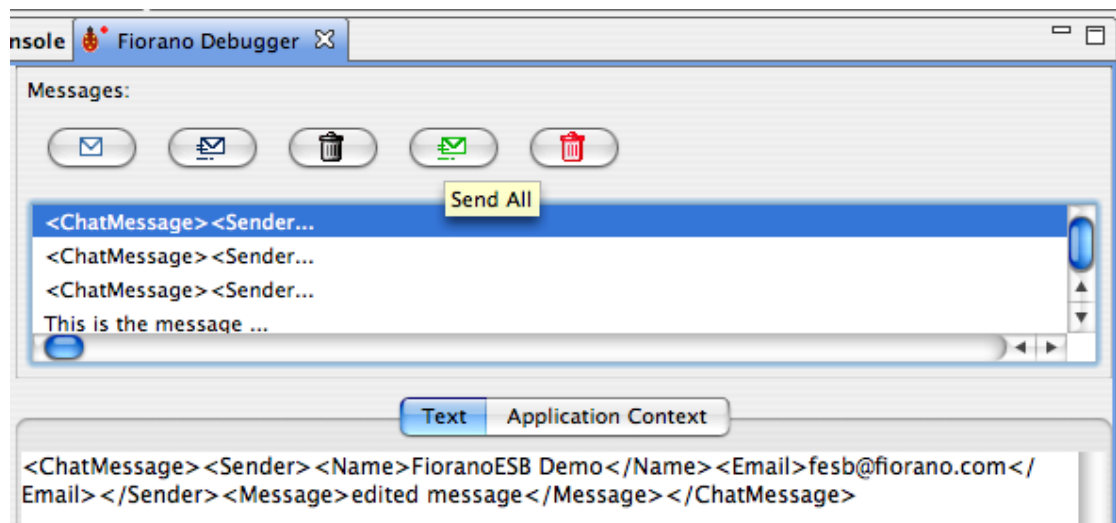


Figure 7.5.2: Send all messages in Fiorano debugger

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.

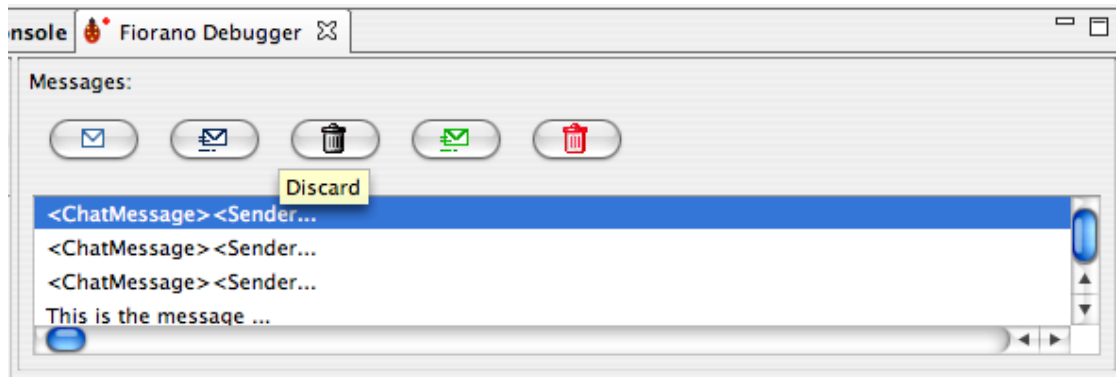


Figure 7.6.1: Discard message in Fiorano debugger

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.

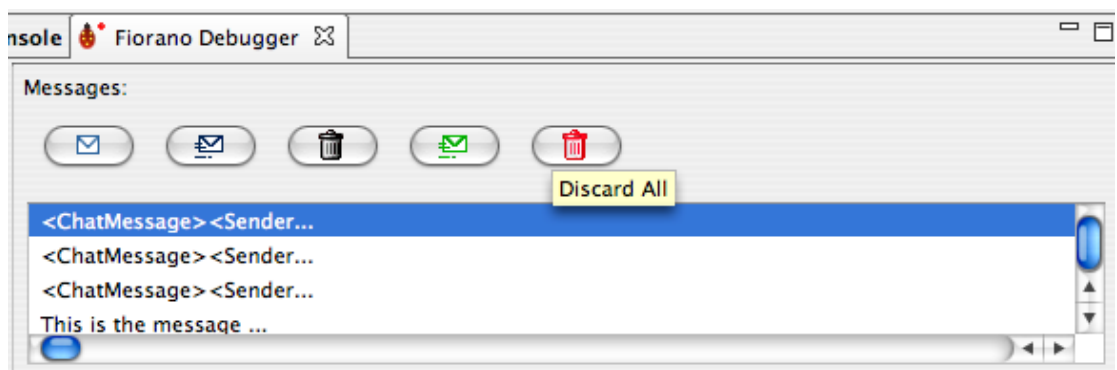


Figure 7.6.2: Discard All messages in Fiorano debugger

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.

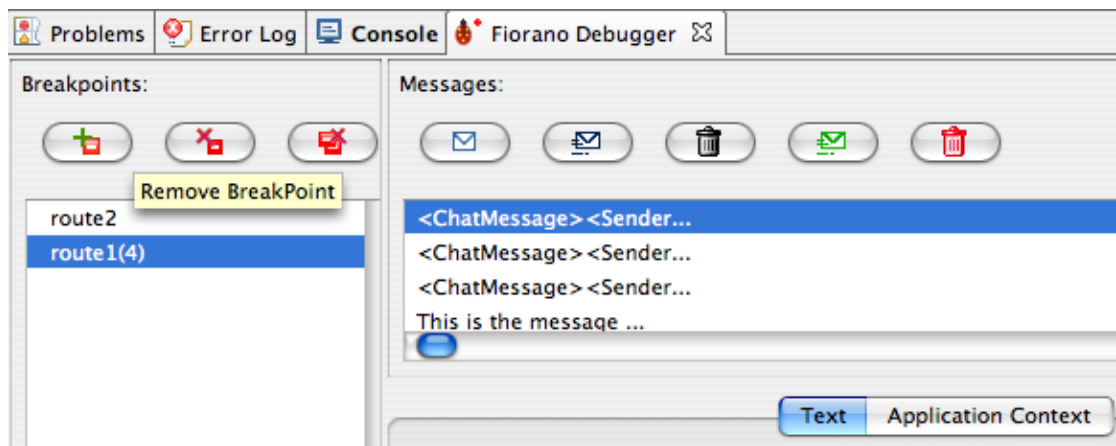


Figure 7.7.1: Remove Breakpoint in Fiorano debugger

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.

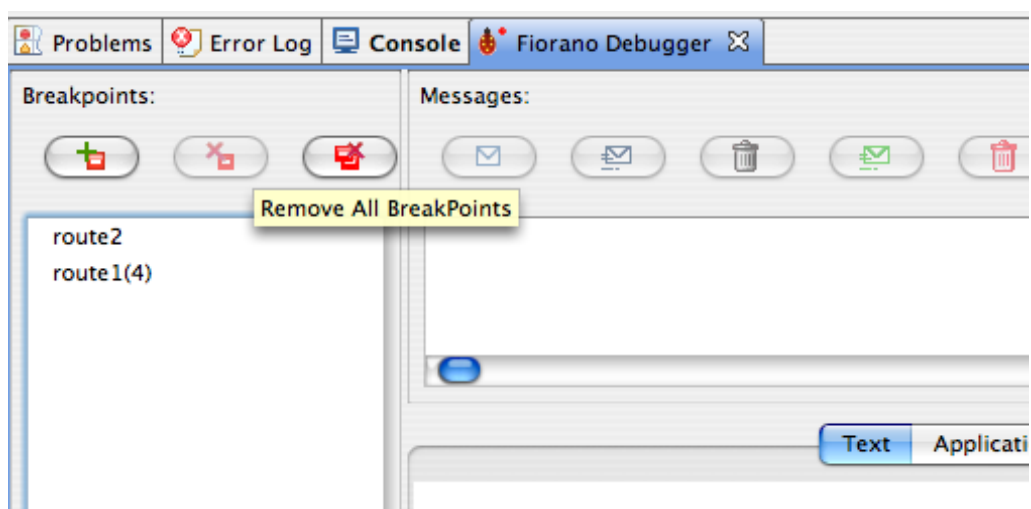


Figure 7.7.2: Remove All BreakPoints in Fiorano debugger

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.

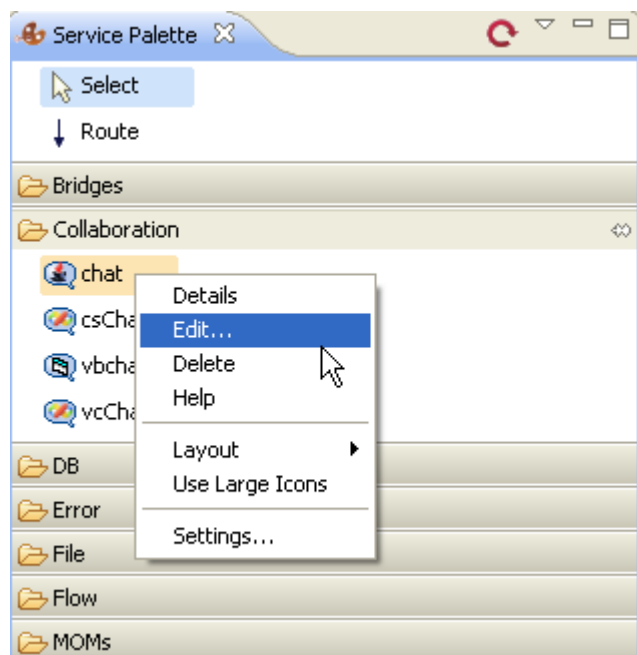


Figure 8.1.1: Edit option

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

The screenshot shows the 'ServiceDescriptor.xml' editor window. The 'Overview' tab is active, displaying the following sections:

- General Information:**
 - Name: chat
 - Version: 4.0
 - GUID: chat
 - Tool Tip: A Simple Chat Component
 - Description: (Empty text area)
- Display:**
 - Palette Icon: (Icon) [Browse...]
 - Display Icon: (Icon) [Browse...]
 - Category: Collaboration ☐
- Metadata:**
 - Author: Fiorano Software Technologies Pvt. Ltd.
 - Creation Date: 04-01-2005 12:00:00
 - ☐ Licensed

At the bottom, there are three tabs: Overview, Execution, and Deployment. The 'Overview' tab is currently selected.

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.

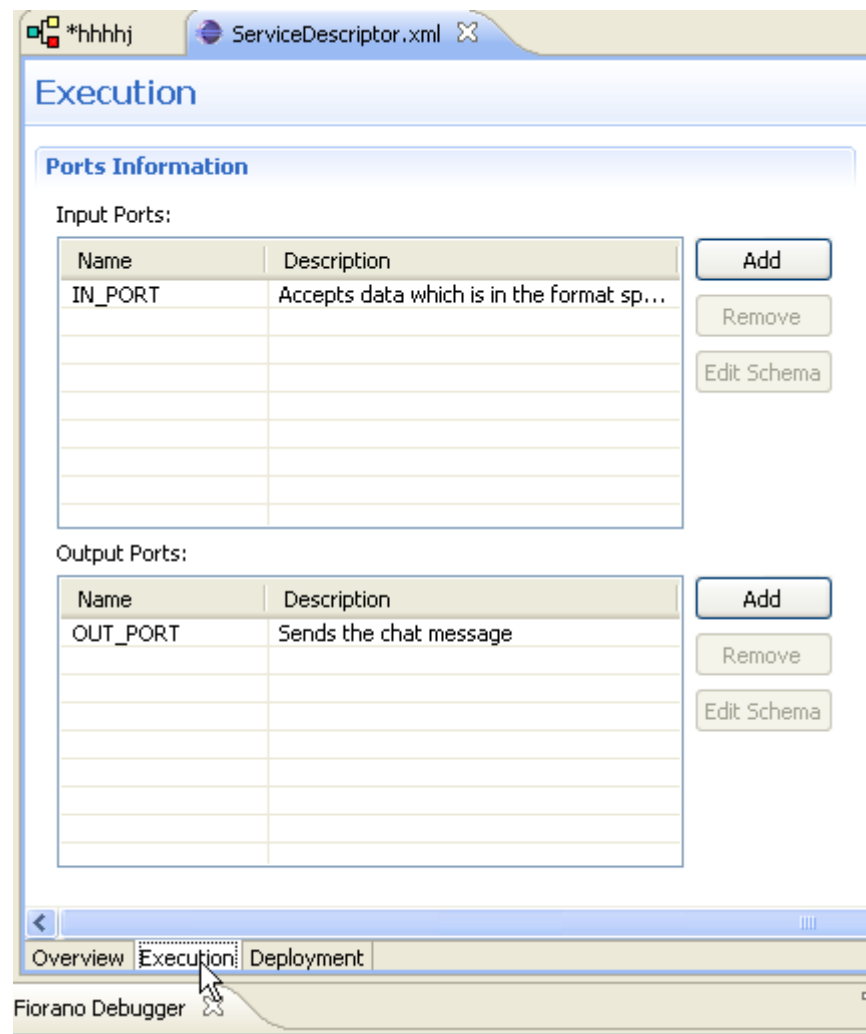


Figure 8.1.3: Sections under Service Descriptor

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.

Overview

General Information

Name:

Version:

GUID:

Tool Tip:

Description:

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.

Display

Palette Icon:

Display Icon:

Category:

Metadata

Author:

Creation Date:

☐ Licensed

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

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.

Run Time

Executable:

In Memory Executable:

Working Directory:

Runtime Arguments:

Name	Type	Mand...	Value
JVM_PARAMS	String	false	-DLOG_TO_...
	String		
	Integer		
	Double		
	Boolean		
	String[]		

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.

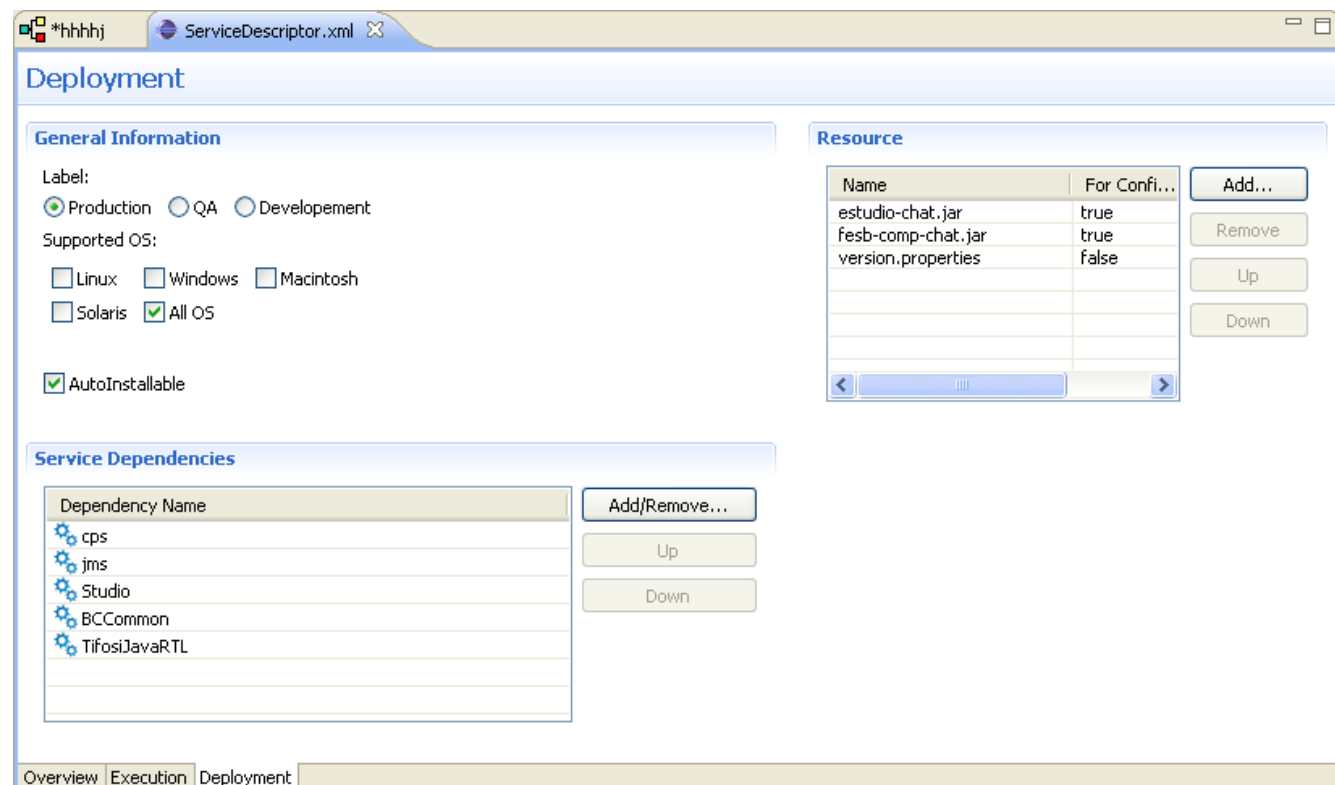


Figure 8.1.11: Deployment page

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.

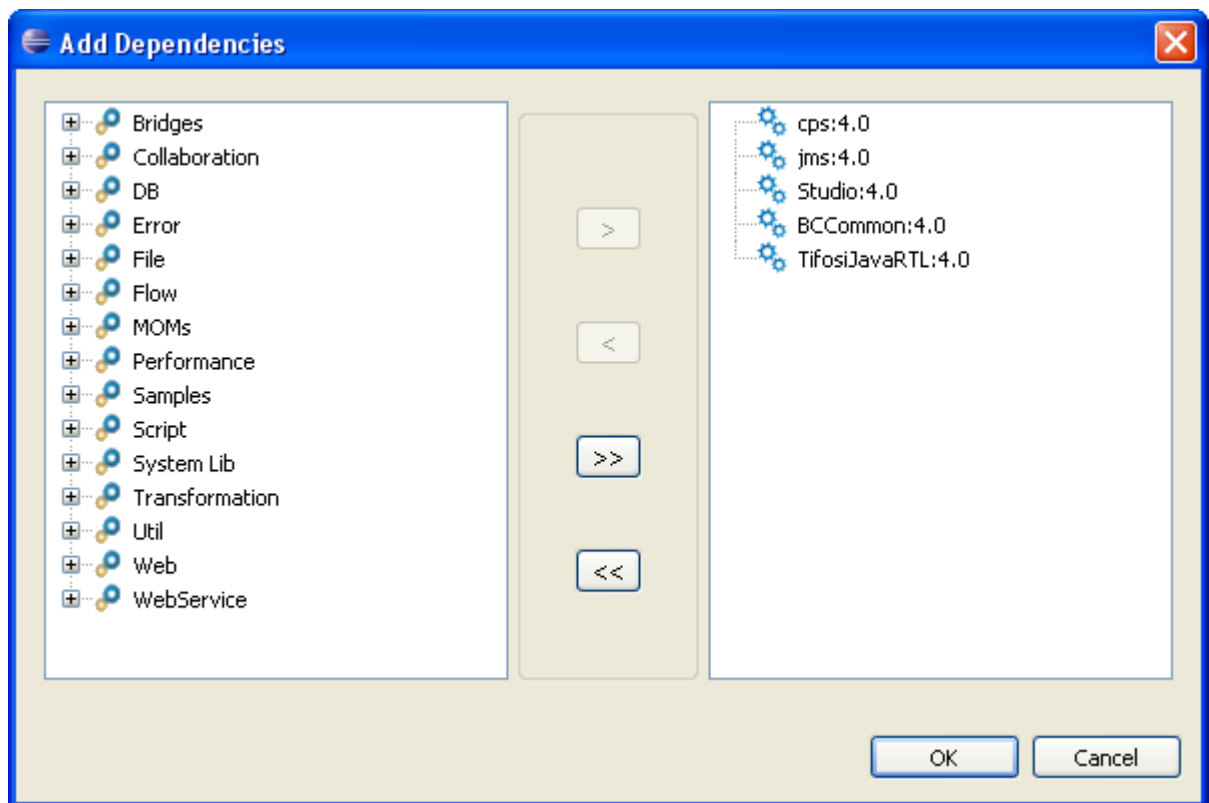


Figure 8.1.12: Service Dependencies section

- **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.

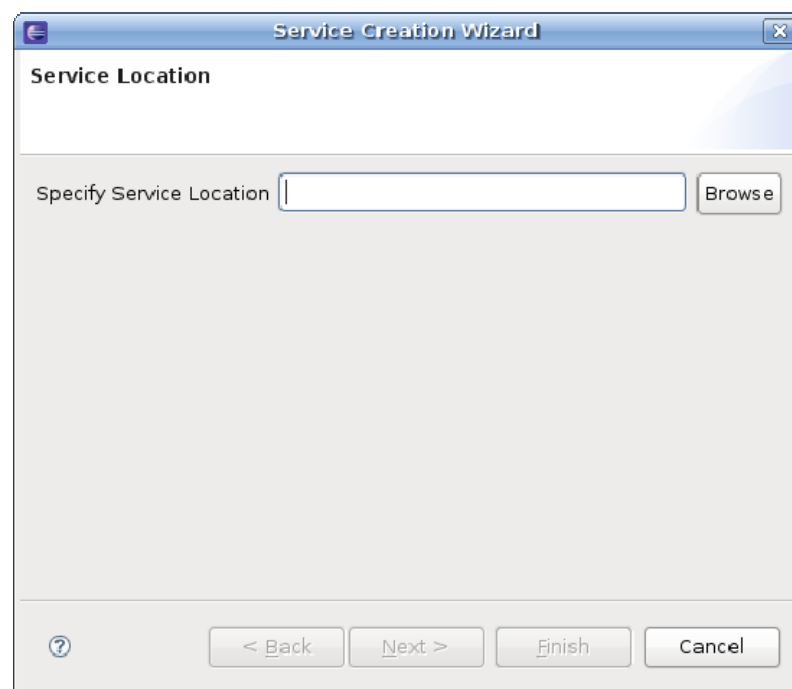
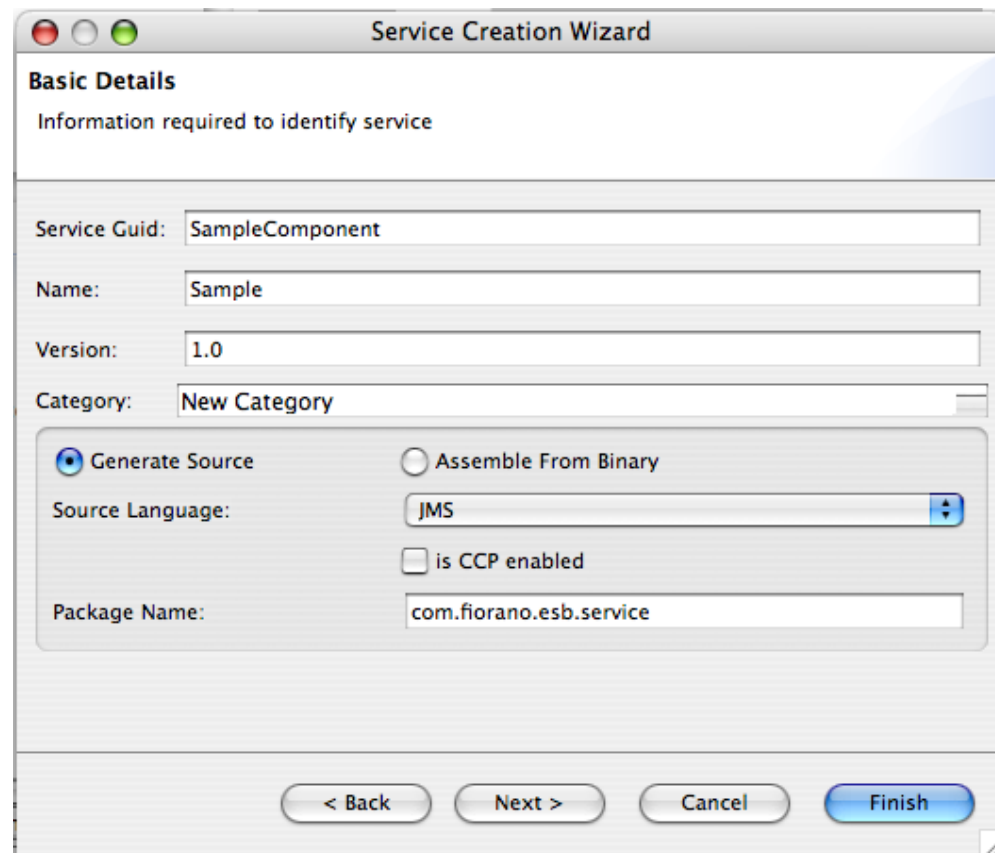


Figure 9.1.1: Specific Service Location

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.



The image shows a 'Service Creation Wizard' dialog box with a 'Basic Details' tab. The title bar says 'Service Creation Wizard'. Below the tab, it says 'Information required to identify service'. The form contains several fields: 'Service Guid' with the value 'SampleComponent', 'Name' with 'Sample', 'Version' with '1.0', and 'Category' with 'New Category'. There are two radio buttons: 'Generate Source' (selected) and 'Assemble From Binary'. Below these, there is a 'Source Language' dropdown menu set to 'JMS', a checkbox for 'is CCP enabled' which is unchecked, and a 'Package Name' field with the value 'com.fiorano.esb.service'. At the bottom, there are four buttons: '< Back', 'Next >', 'Cancel', and 'Finish'.

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.

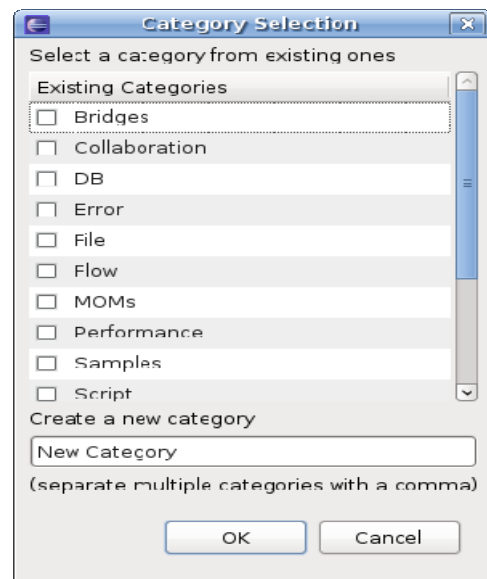


Figure 9.1.3: Category Selection dialog box

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.

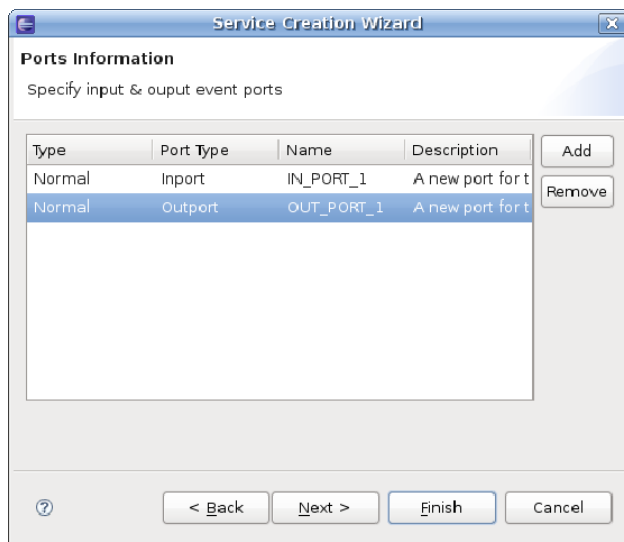


Figure 9.1.4: Ports Information

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.

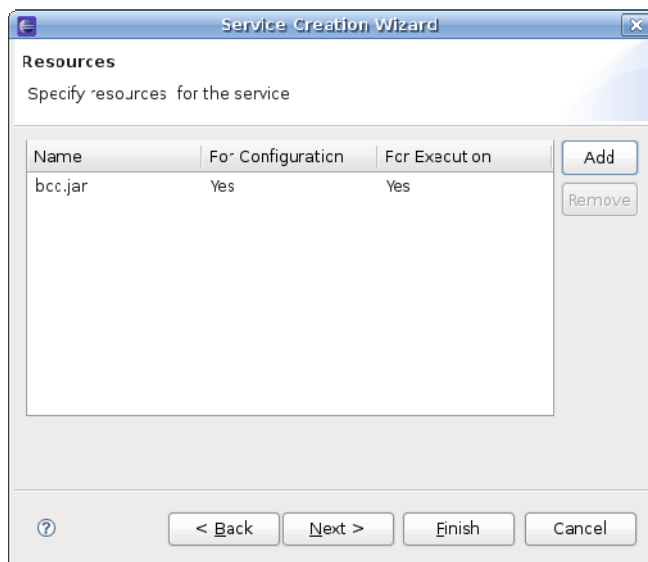


Figure 9.1.5: Resources section

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.

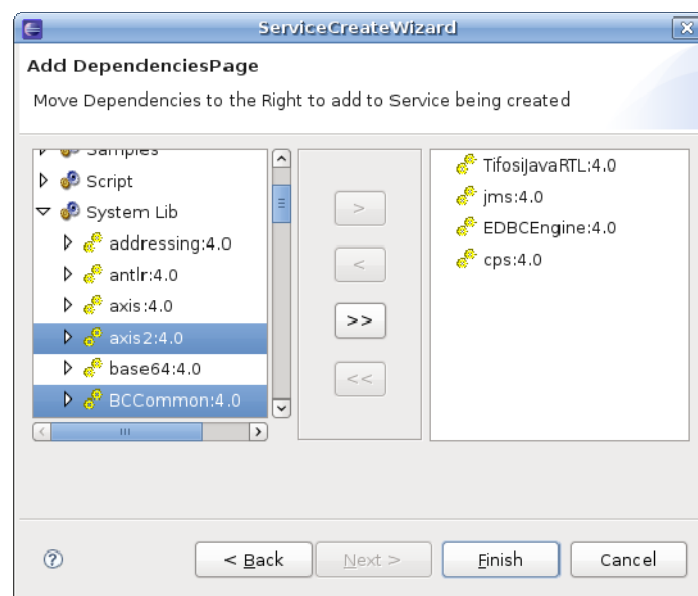


Figure 9.1.6: Dependencies

- 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.

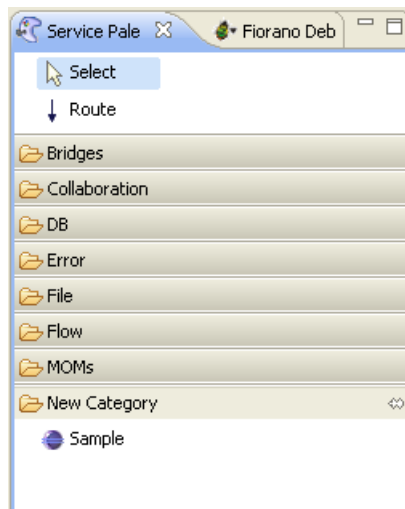


Figure 9.2.2: Service Palette

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.
- 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
 - o Map View
 - o 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.

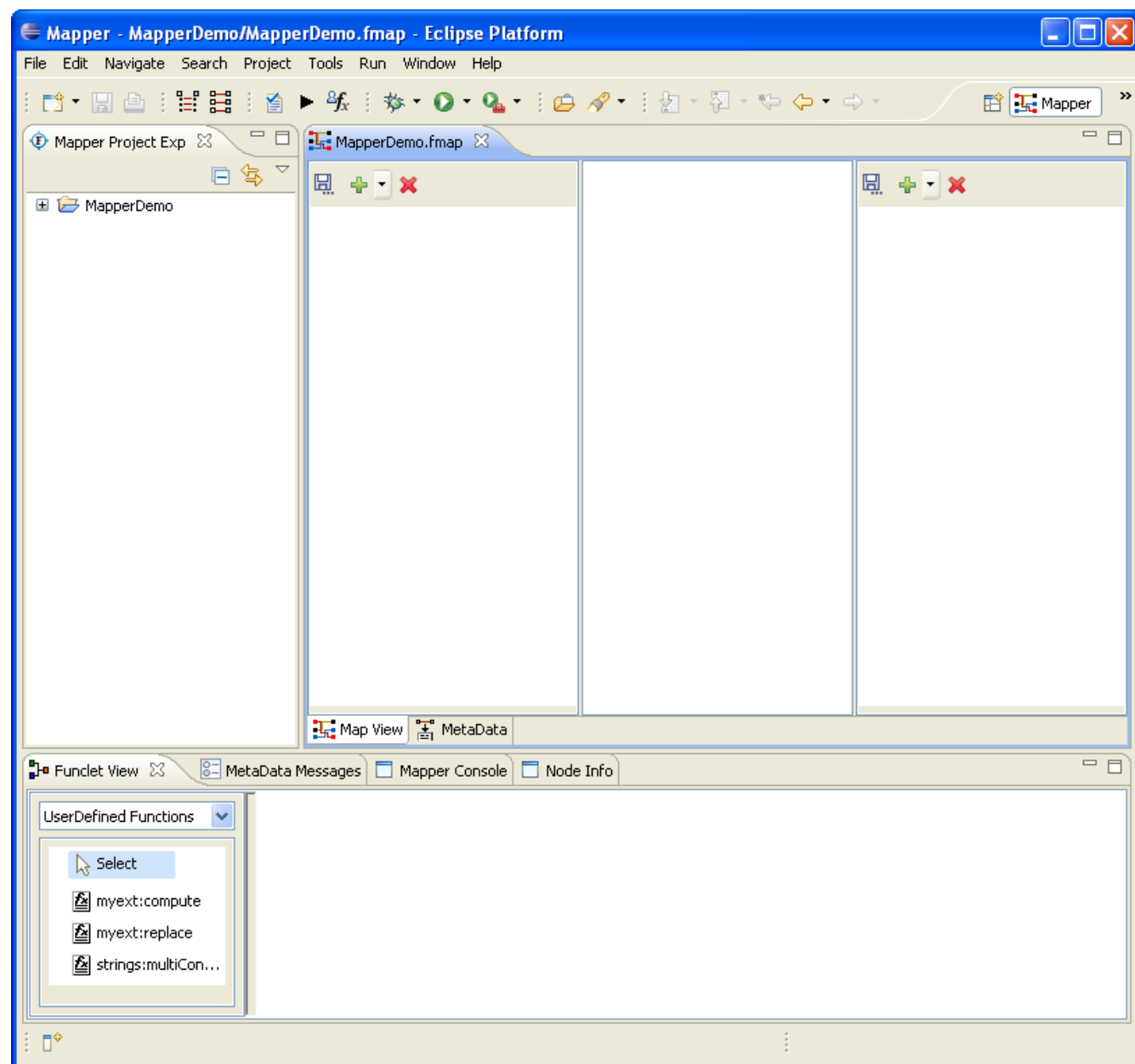


Figure 10.2.1: eMapper Perspective

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 eMapper Projects view and select **New > Fiorano Map**. 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 shows a sample eMapper project as shown in the eMapper Projects Explorer.

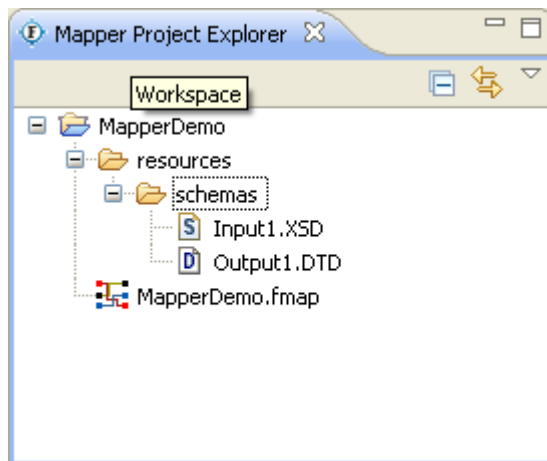


Figure 10.2.2: eMapper Projects

An eMapper Project contains a resources folder which holds the .fmp file. The .fmp stores the mappings defined between the input and output structures. The schemas provided for all the input and output structures are stored in the PROJECT_HOME/resources/schemas folder. The names of these schema files are of the form <Structure_Name>.<Mime_type>.

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.

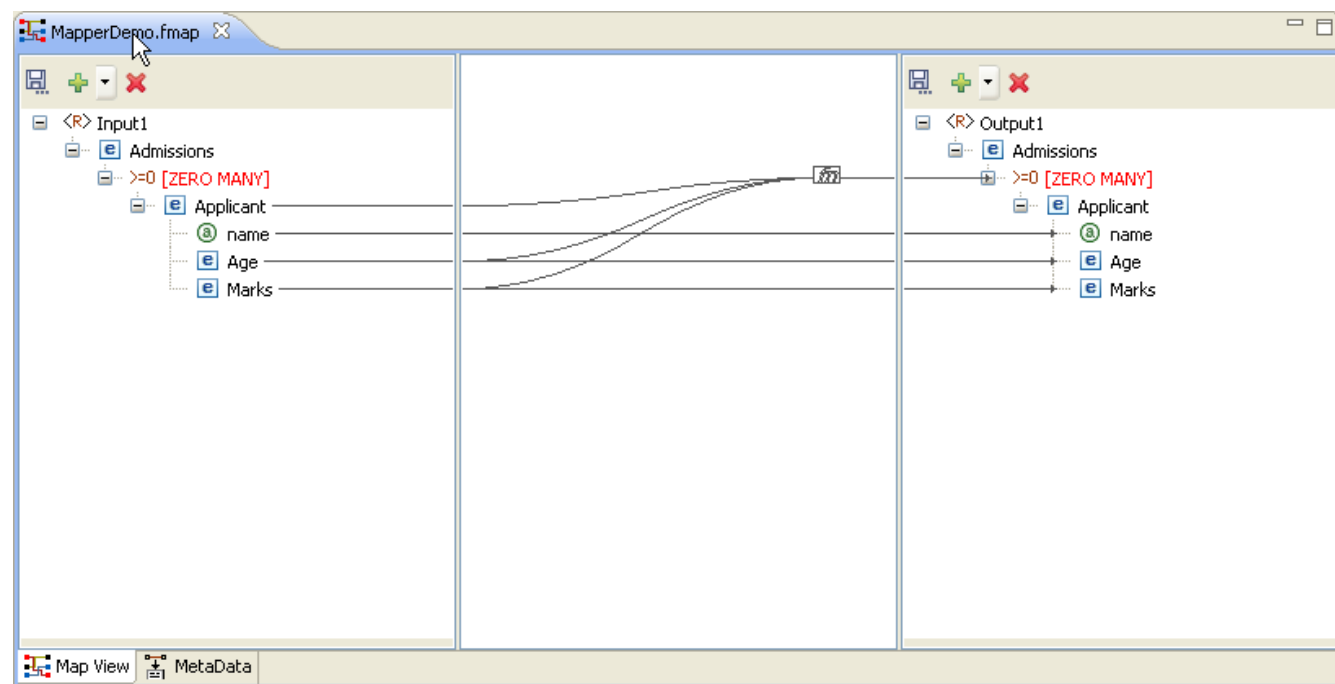


Figure 10.2.3: Map View

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.

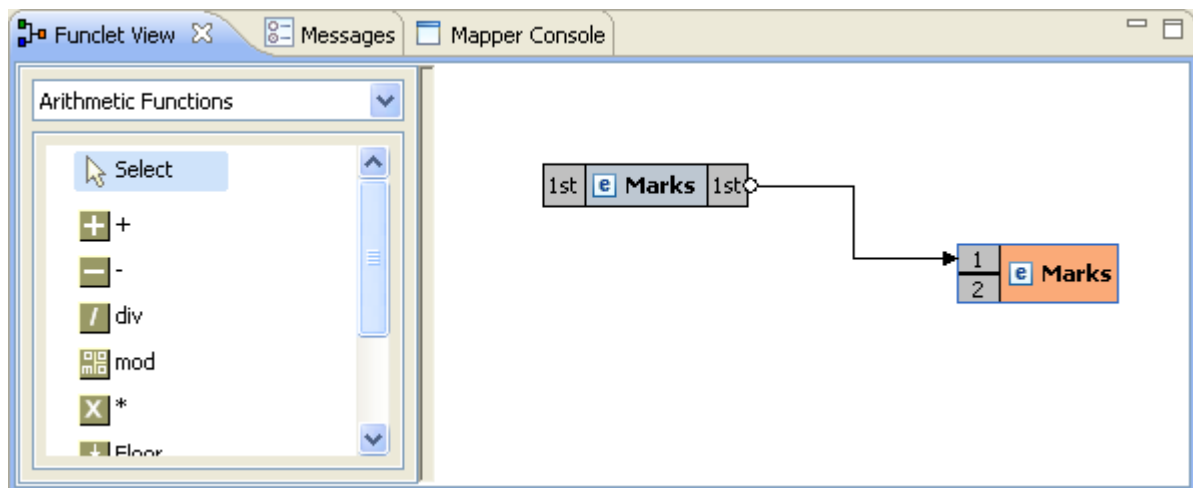


Figure 10.2.4: Funclet View

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.

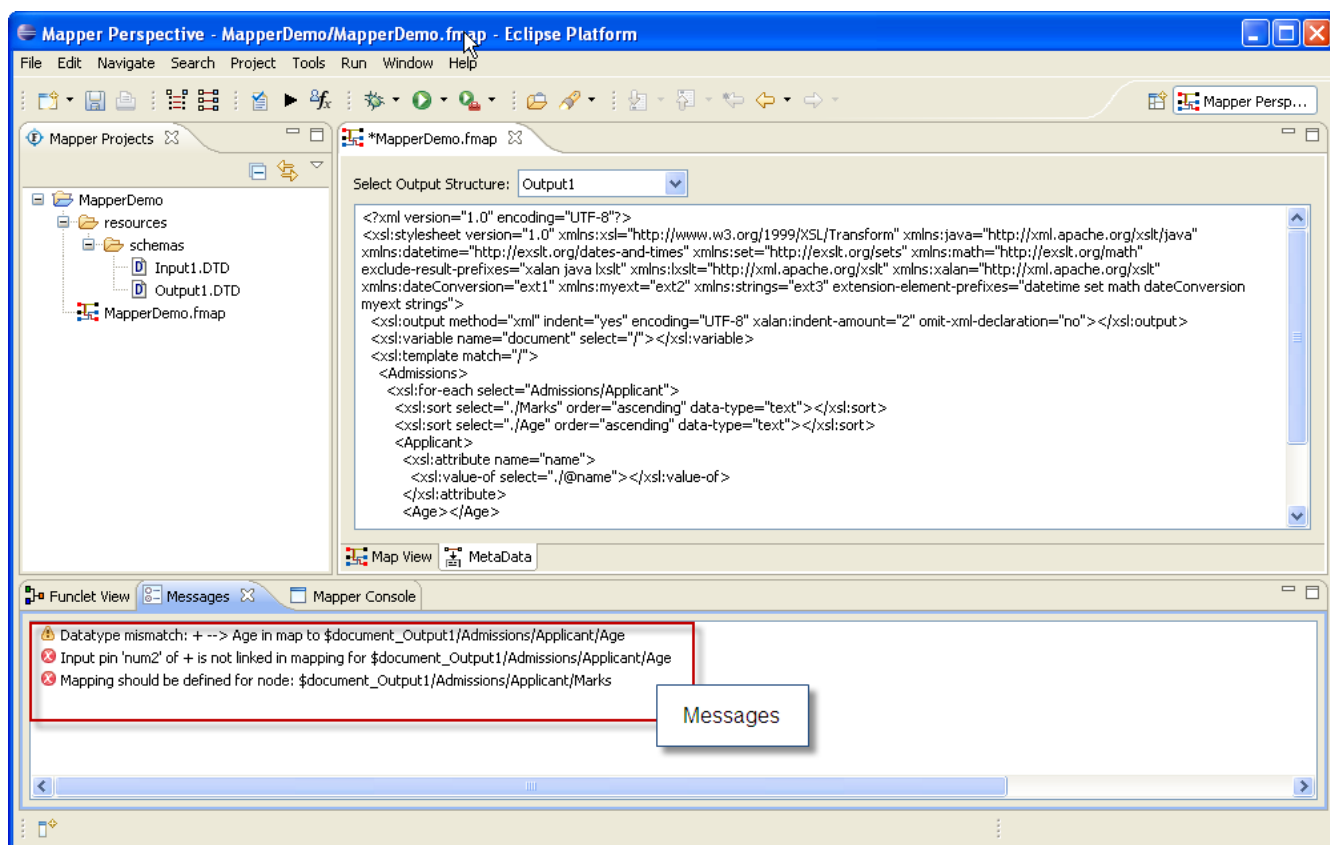
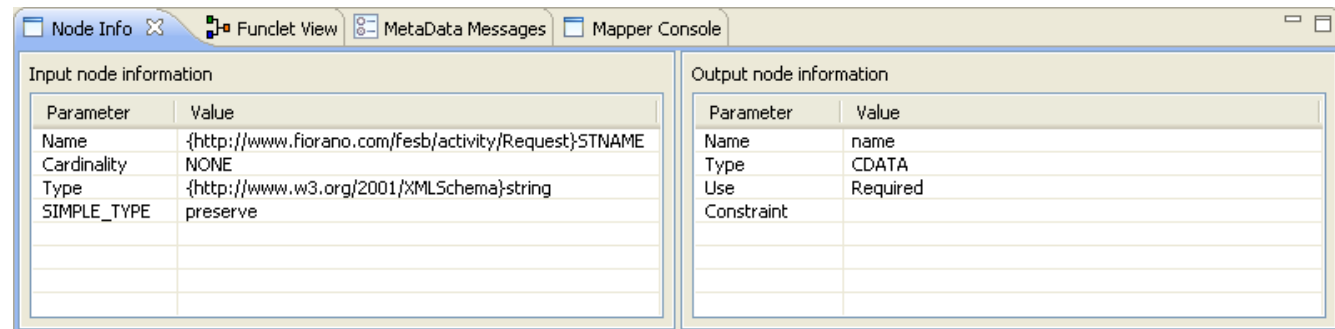


Figure 10.2.5: Meta Data and MetaData Messages view

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

1. 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.
2. 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

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.

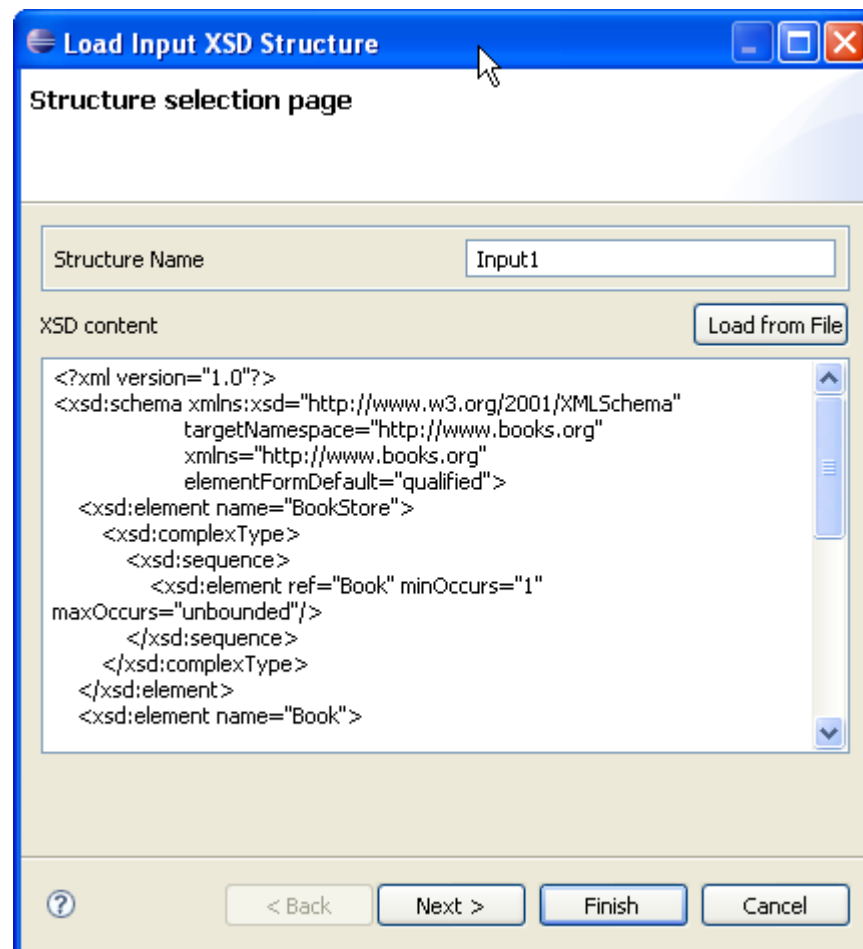


Figure 10.3.1: Structure Selection Page

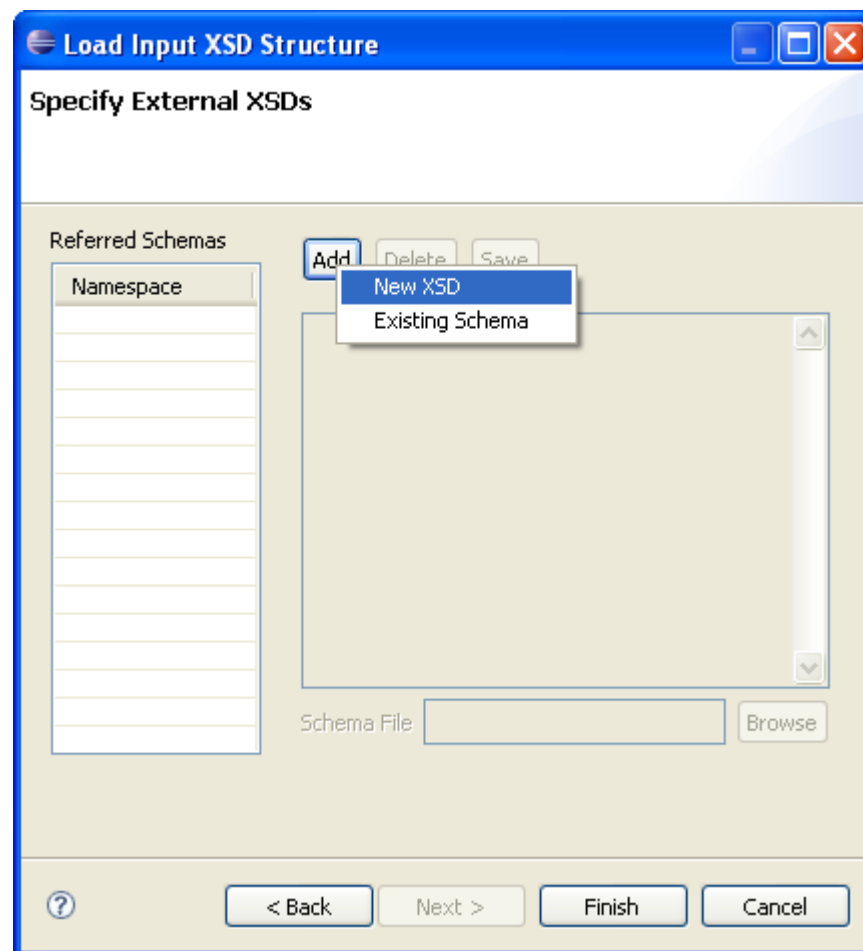
Structure Selection Page

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.

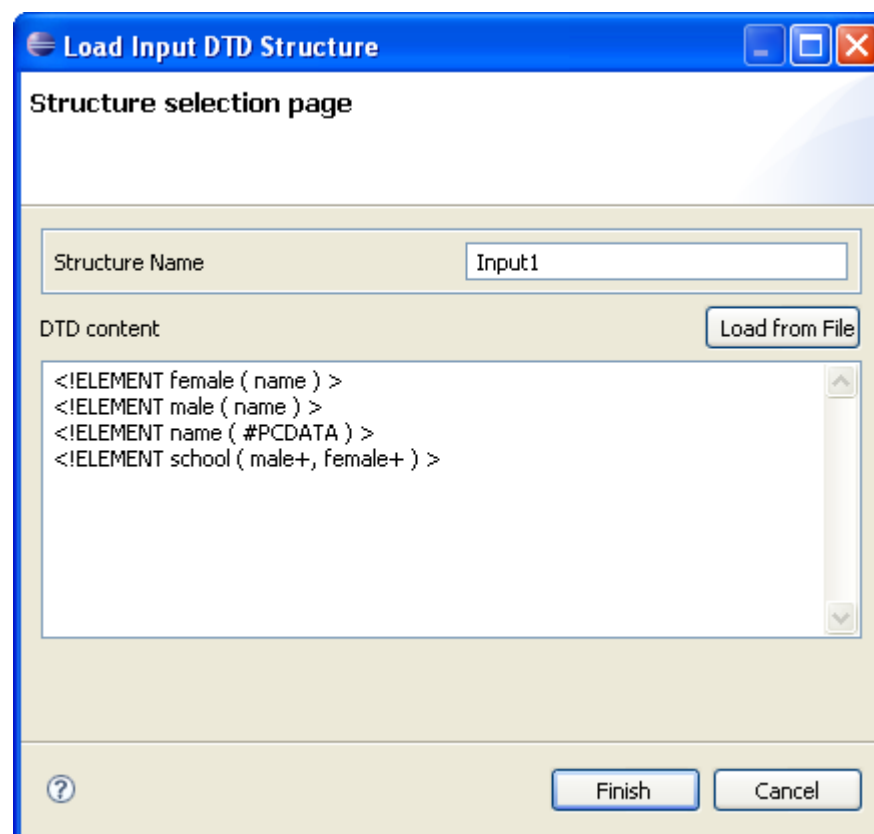


Figure 10.3.3: Load Input DTD Structure

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.

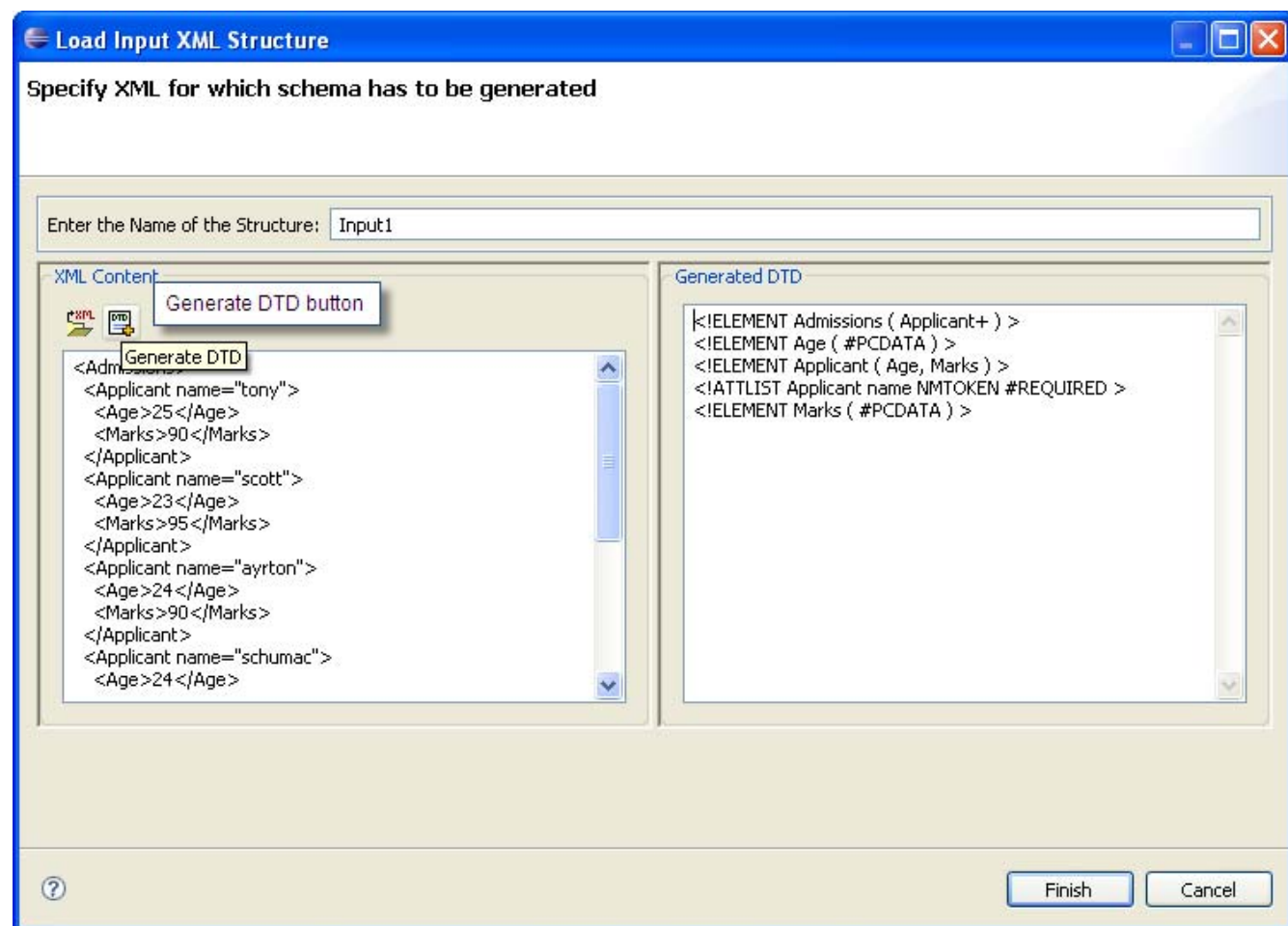


Figure 10.3.4: Load Input XML Structure

10.3.2 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.

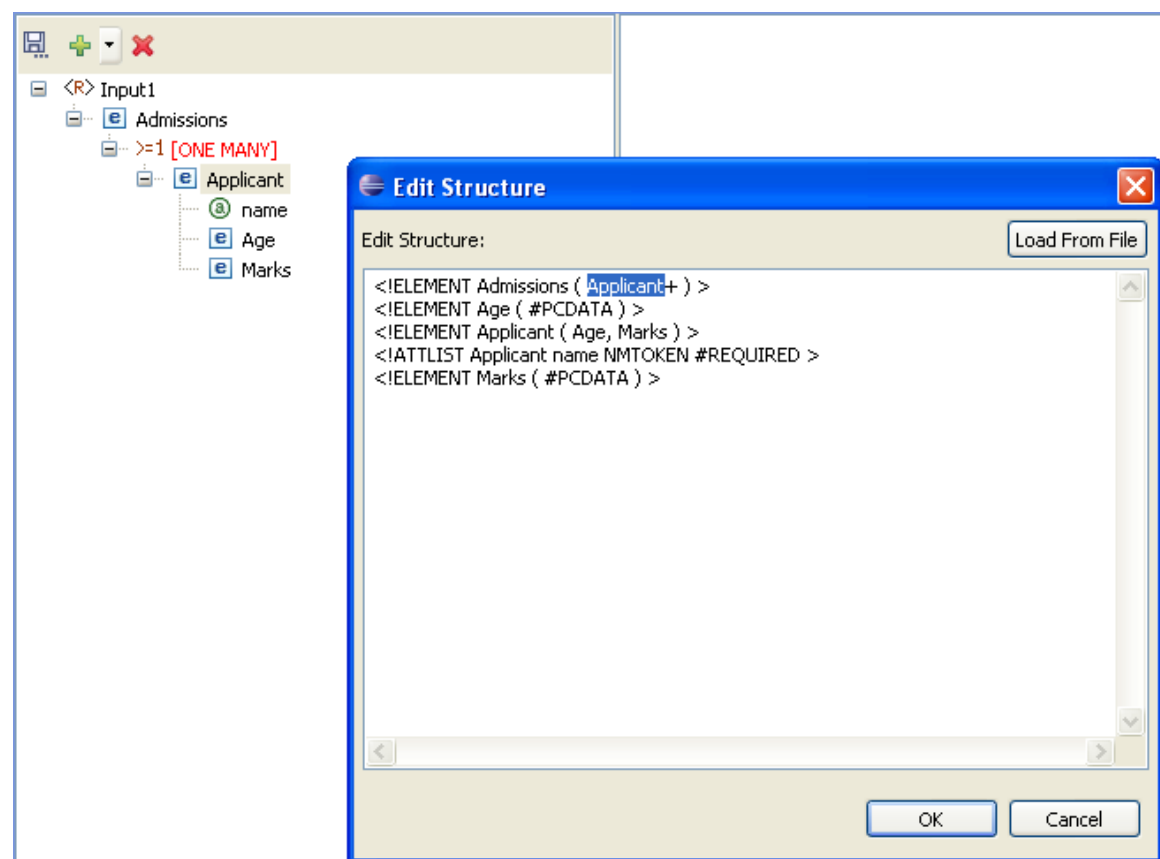


Figure 10.3.5: Edit Structure Dialog

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 Funclet 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

Funclet easel

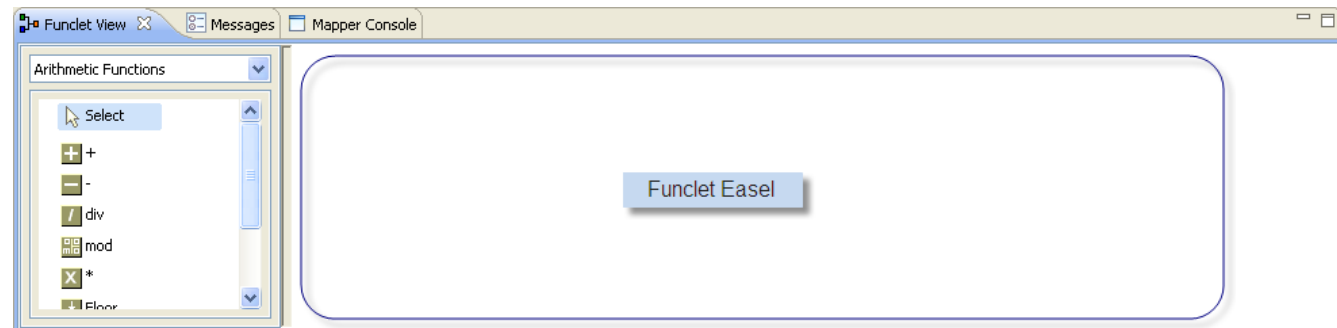


Figure 10.4 Funclet View

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	<code>_TIF_HEADER_<HEADERNAME></code>
Message (text)	<code>_TIF_BODY_TEXT_</code>
Message (byte)	<code>_TIF_BODY_BYTE_</code>
Attachment	<code>_TIF_ATTACH_<NAME></code>

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--

OutputThree-letter English month abbreviation: 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov' or 'Dec'. If the date/time string argument is not in valid, then an empty string ('') is returned.

MonthInYear**Visual representation**

Description: This function returns the month of a date as a number. The counting of the month starts from 0. If no argument is given, the current local date/time is used as the default argument.

Input: Date string in any of the following formats:

CCYY-MM-DDThh:mm:ss
CCYY-MM-DD
CCYY-MM
--MM--
--MM-DD

If the date/time string is not valid, then NaN is returned.

Output: A number representing the month in a year.

Example: 0 for January, 1 for February, 2 for March and so on.

MonthName**Visual representation**

Description: This function returns the full name of the month of a date. If no argument is given, then the current local date/time is used as the default argument.

Input: Optionally, a date string in any of the following formats:

CCYY-MM-DDThh:mm:ss
 CCYY-MM-DD
 CCYY-MM
 --MM--

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: – 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.

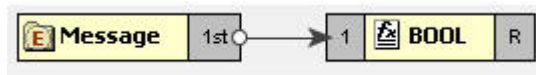


Figure 10.4.1: Linking Message and BOOL nodes

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.

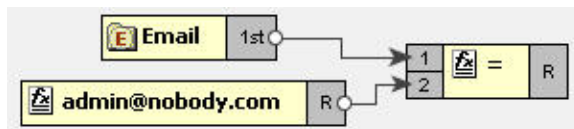


Figure 10.4.2: Linking the Email and Constant node outputs

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.

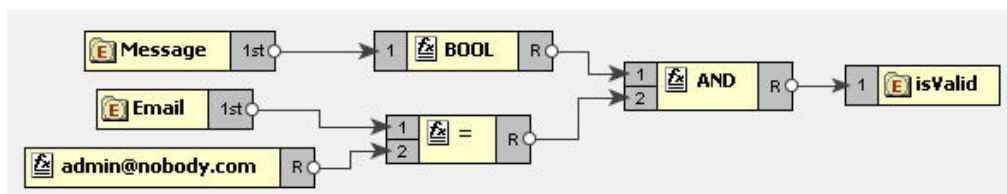


Figure 10.4.3: Linking the AND and = node outputs

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.

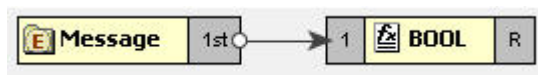


Figure 10.4.4: Linking Message and BOOL nodes

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.

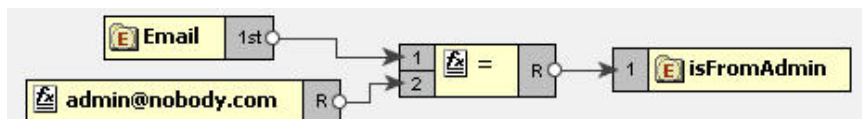


Figure 10.4.7: Linking = and isFromAdmin node

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.

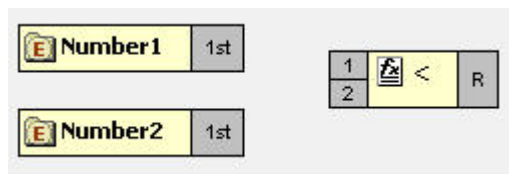


Figure 10.4.8: Placing a node on the Function Easel

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.

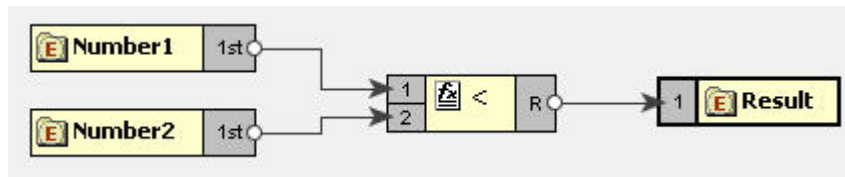


Figure 10.4.9: Linking < and Result node

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.

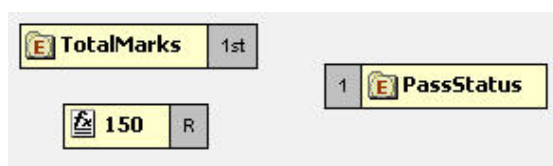


Figure 10.4.10: Setting the Constant building block to 150

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.

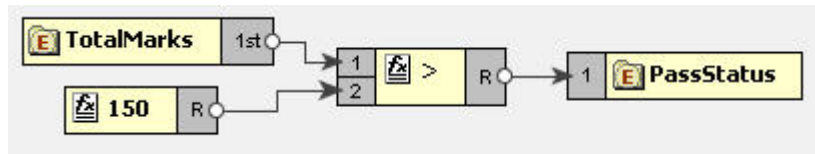


Figure 10.4.11: Linking the > and PassStatus node

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.

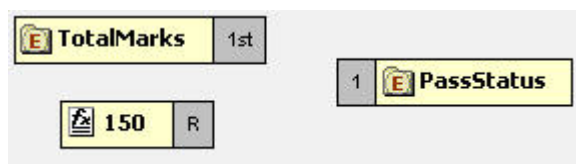


Figure 10.4.12: Setting the Constant building block to 150

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.

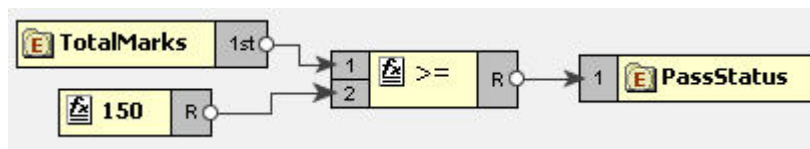


Figure 10.4.13: Linking the >= and PassStatus nodes

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.

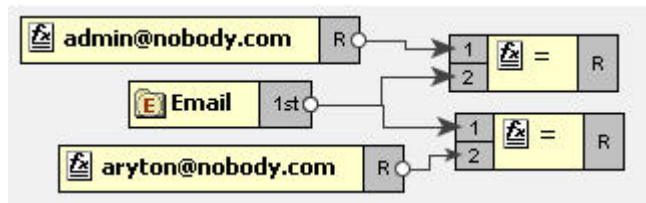


Figure 10.4.15: Placing two = nodes on the Function Easel

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.

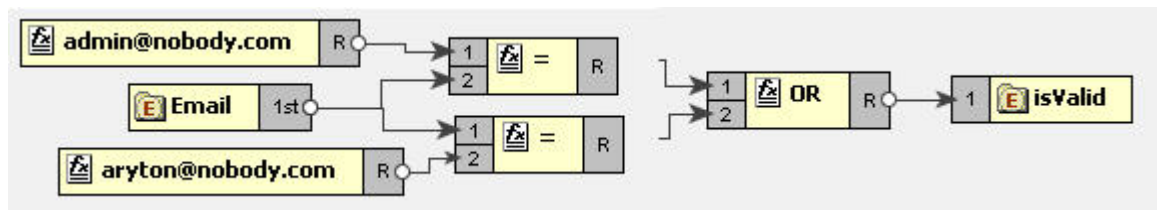


Figure 10.4.16: Linking the OR and isValid nodes

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:

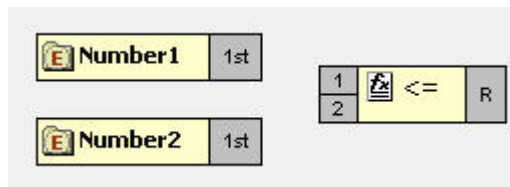


Figure 10.4.17: Placing <= node on the Function Easel

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:

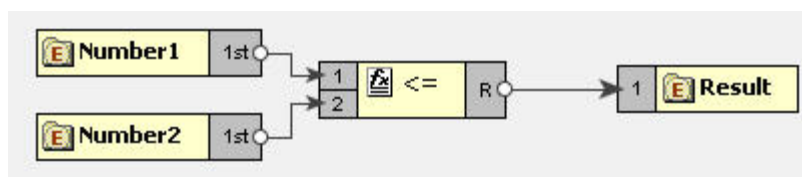


Figure 10.4.18: Linking outputs of the **Number1** and **Number2** nodes

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 10.4.18.



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.

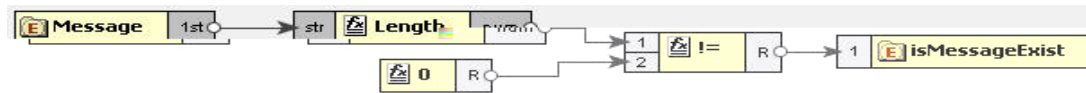


Figure 10.4.22: Linking the != and isMessageExist nodes

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. Lookup functions take key columns name value pairs as **<column1>=<value1>,<column2>=<value2>** etc.
For Example: dvSendDept=100, dvSendCode=BLK
2. Lookup functions can return the values 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 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. 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
2. Dates are expected in MM/dd/yyyy HH:mm:ss format
3. 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.

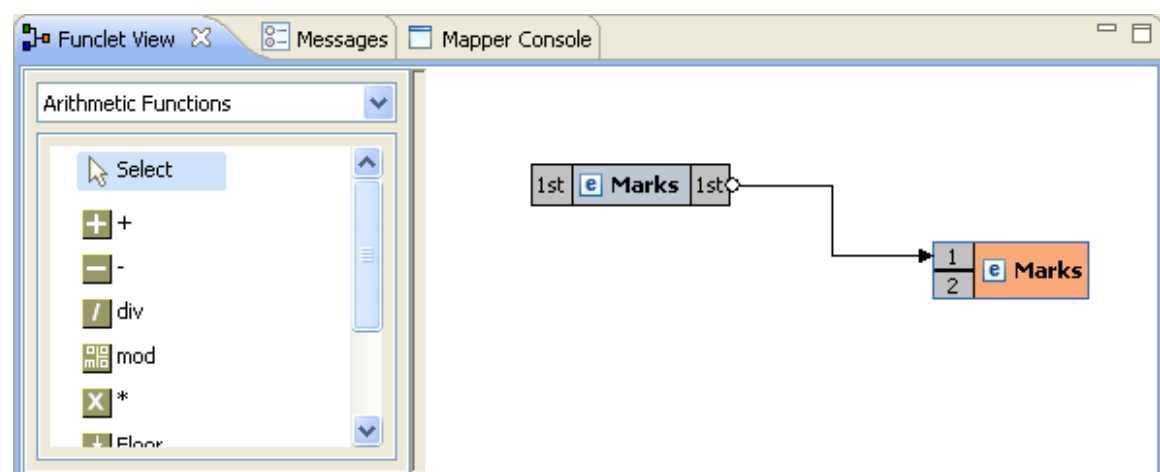


Figure 10.4.24: Funclet easel

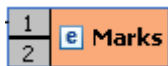
10.4.2.1 Source Node

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**

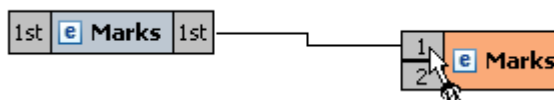
Add Link between two Nodes

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.

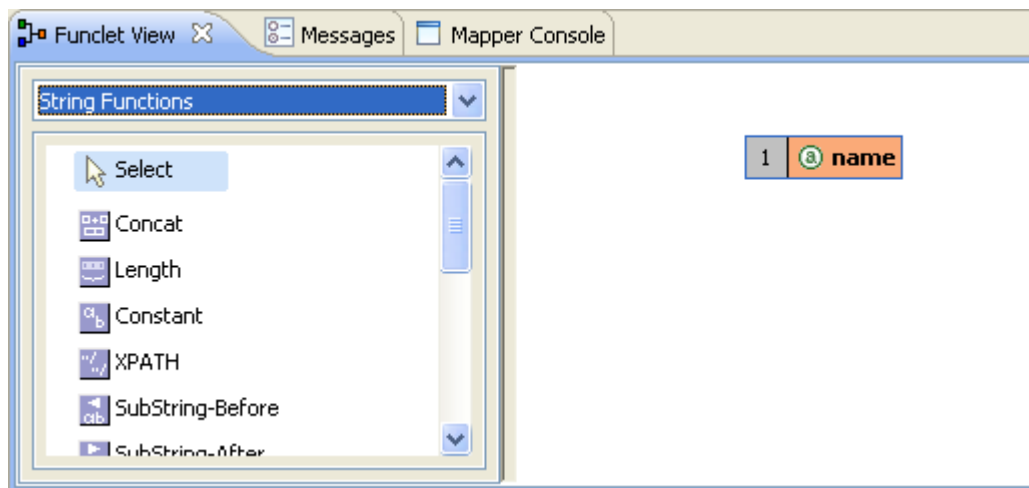


Figure 10.4.29: Adding Source node to Funclet easel

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.

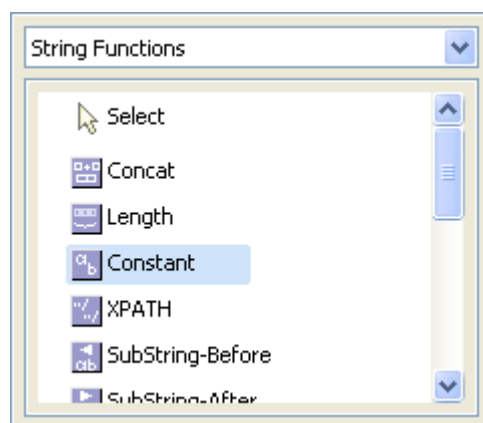


Figure 10.4.30: Selecting the Function node

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.

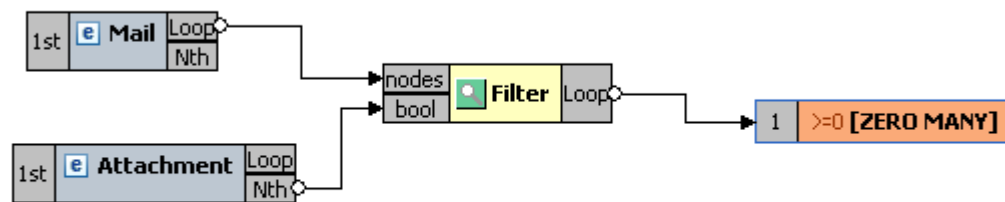


Figure 10.4.31: Funclet easel

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.

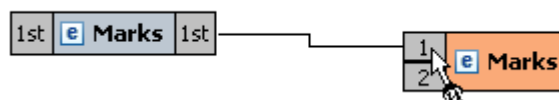


Figure 10.4.33: Linking the Source and the destination node

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

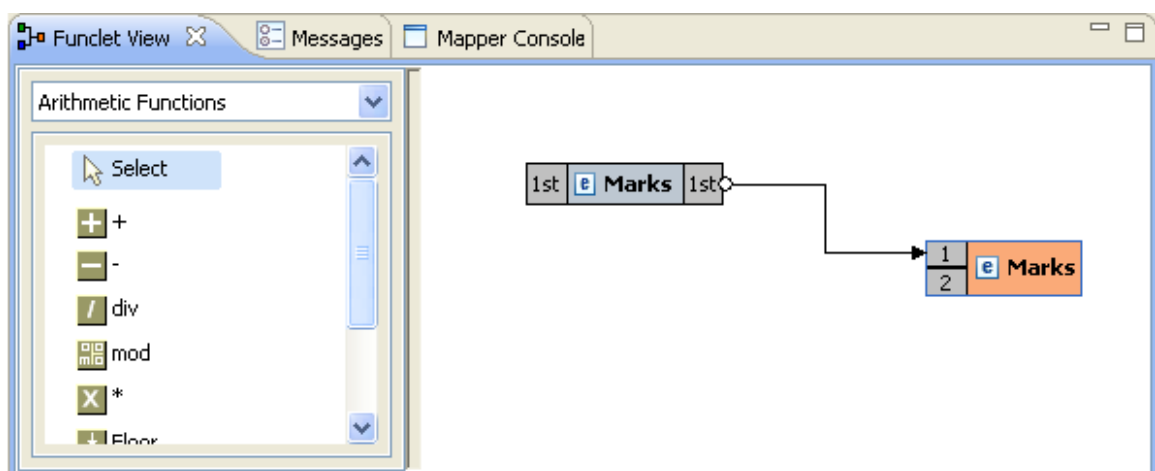


Figure 10.4.34: Linking Source and Destination nodes

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.

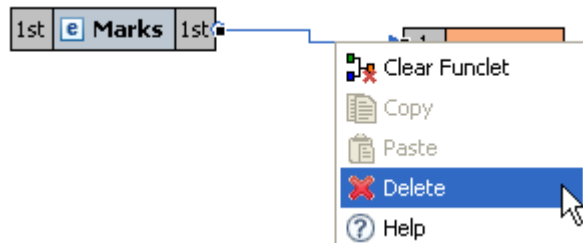


Figure 10.4.35: Deleting link

- 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.

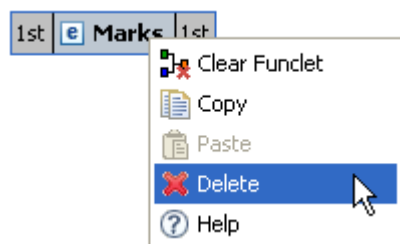


Figure 10.4.36: Pop-up menu

- 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.

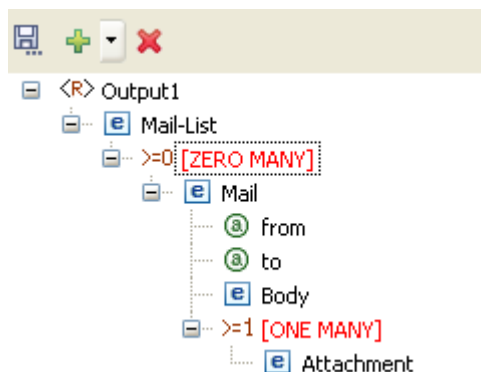


Figure 10.5.1: Example of Zero to Many control node

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.

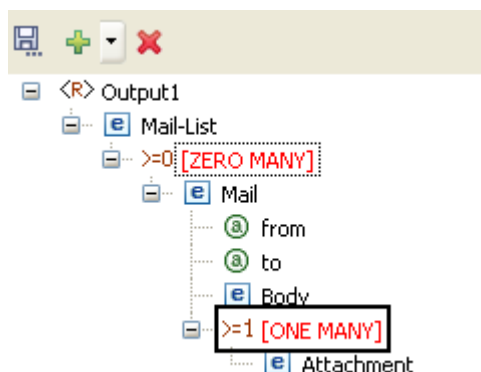


Figure 10.5.2: Example of One to Many control node

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.

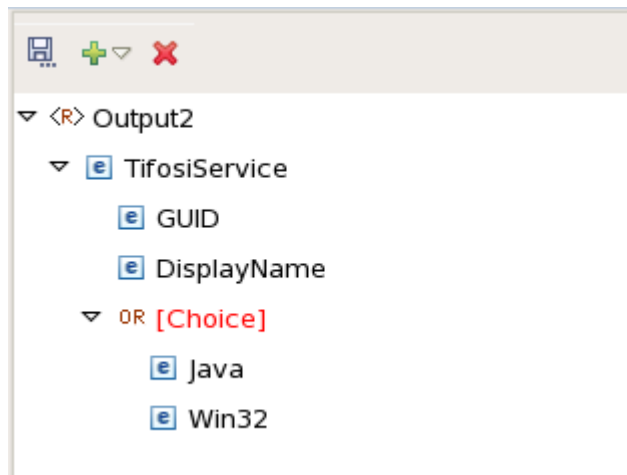


Figure 10.5.3: Example of Choice Control Node

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.

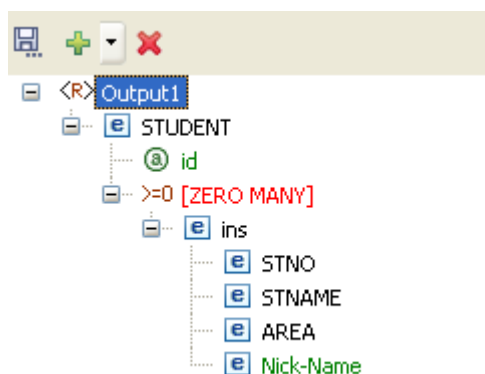


Figure 10.5.4: Example of Optional Node

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.

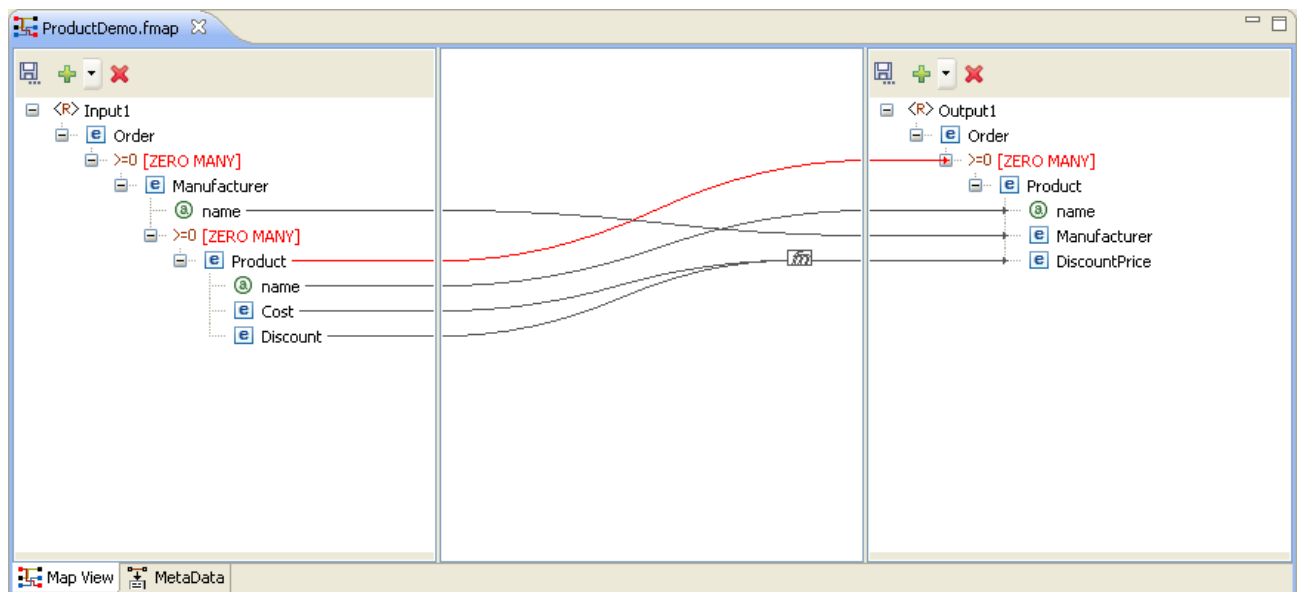


Figure 10.5.6: Name-to-name Mapping

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.

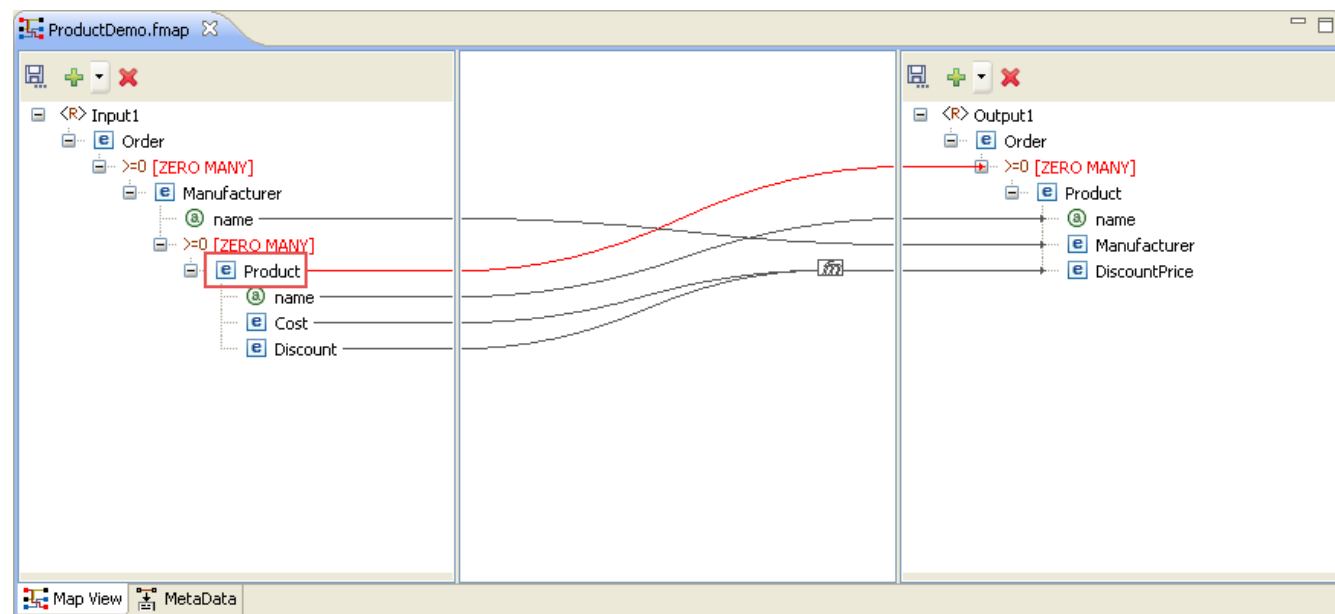


Figure 10.5.7: For-Each Mapping

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.

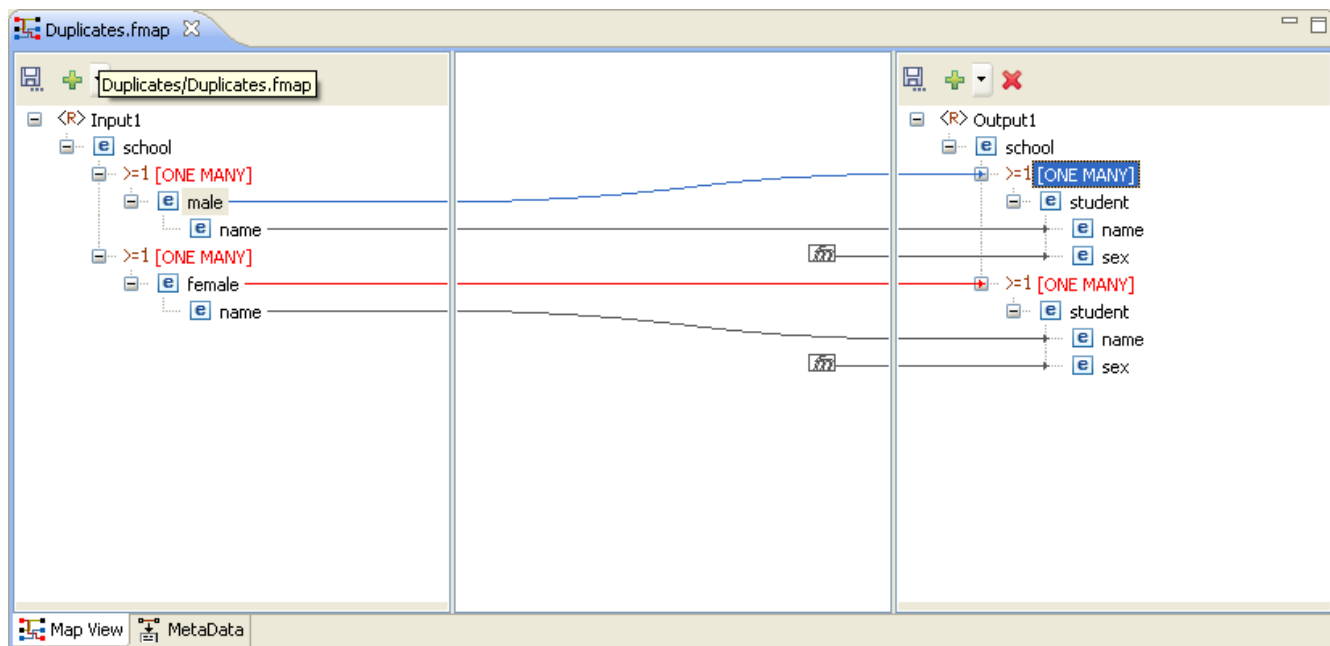


Figure 10.5.8: Mapping a node to One Many control node

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.

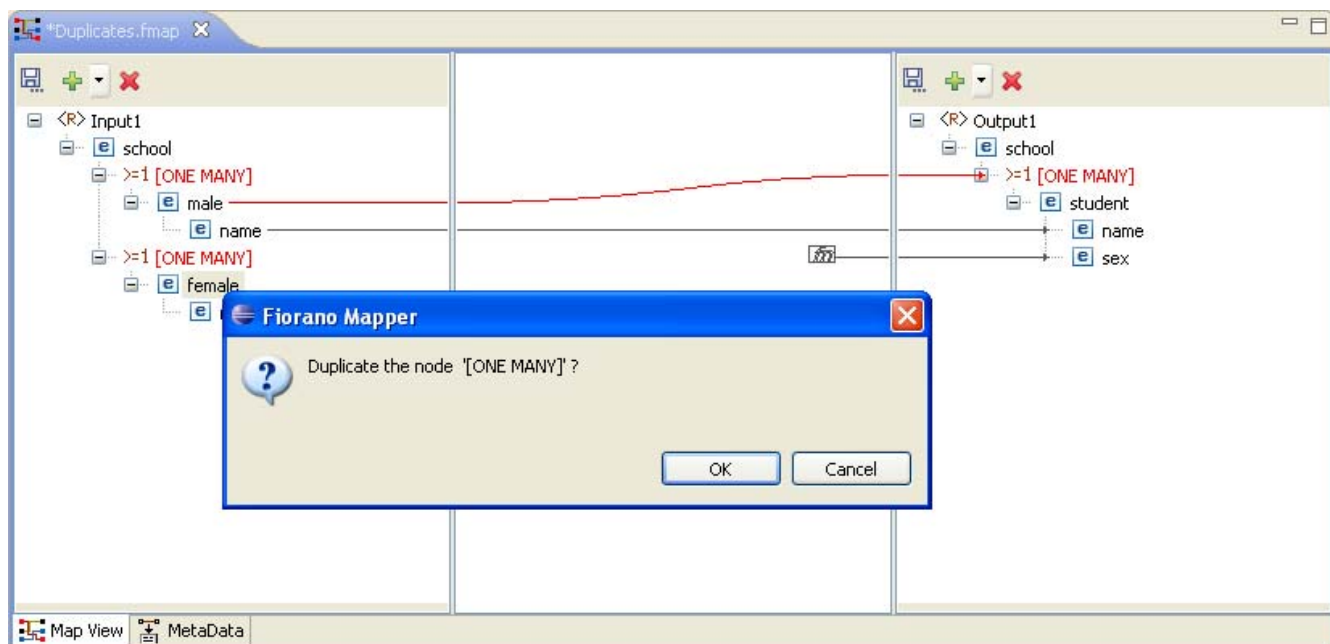


Figure 10.5.9: A shortcut menu prompts you to duplicate the node

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

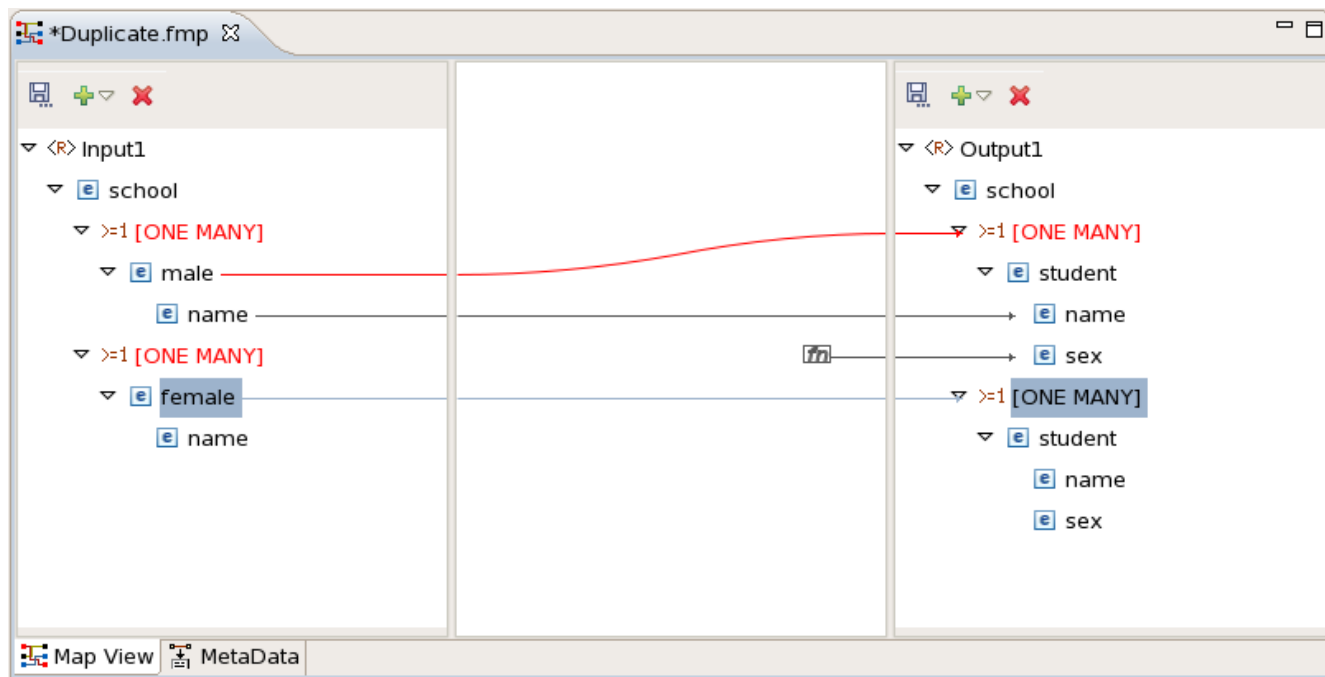


Figure 10.5.10: The One Many Node is Duplicated

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.

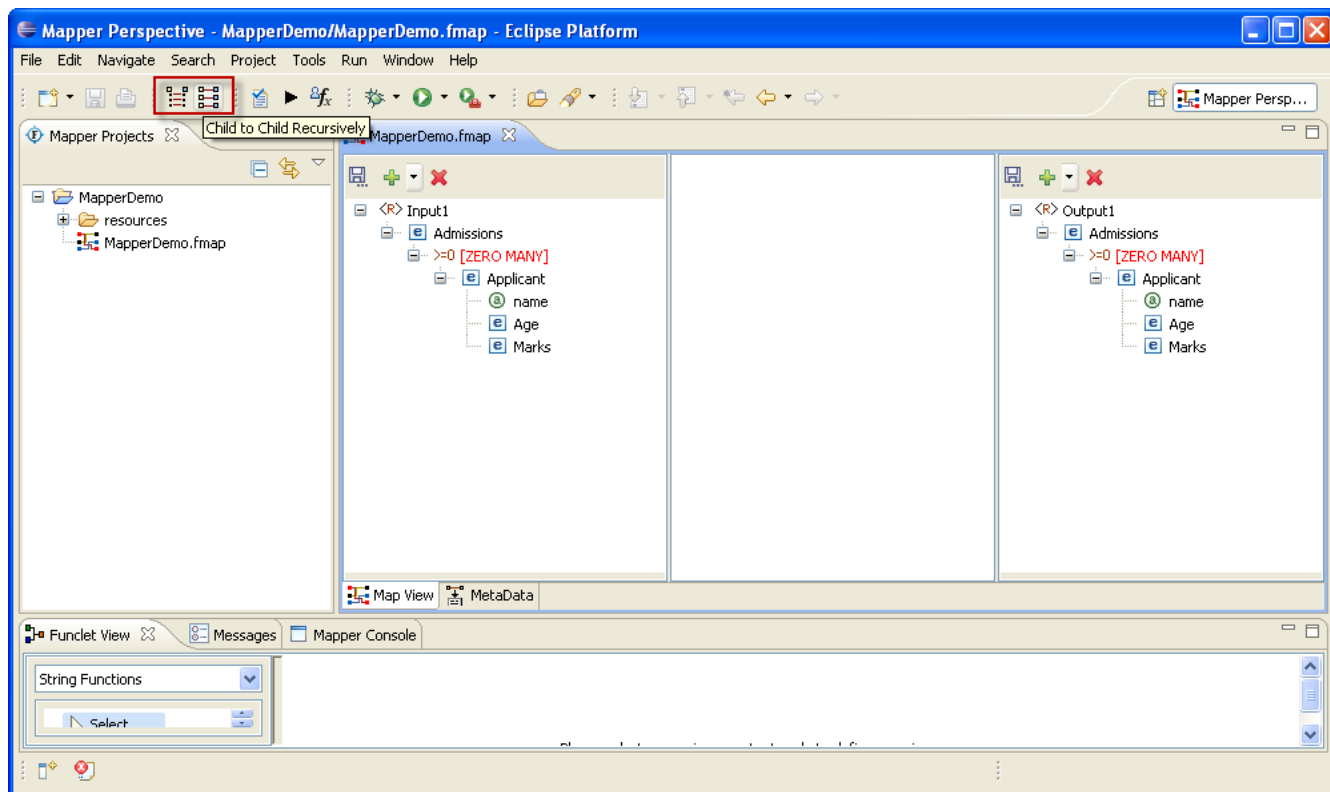


Figure 10.5.11: Creating Automatic Mapping between child nodes

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.

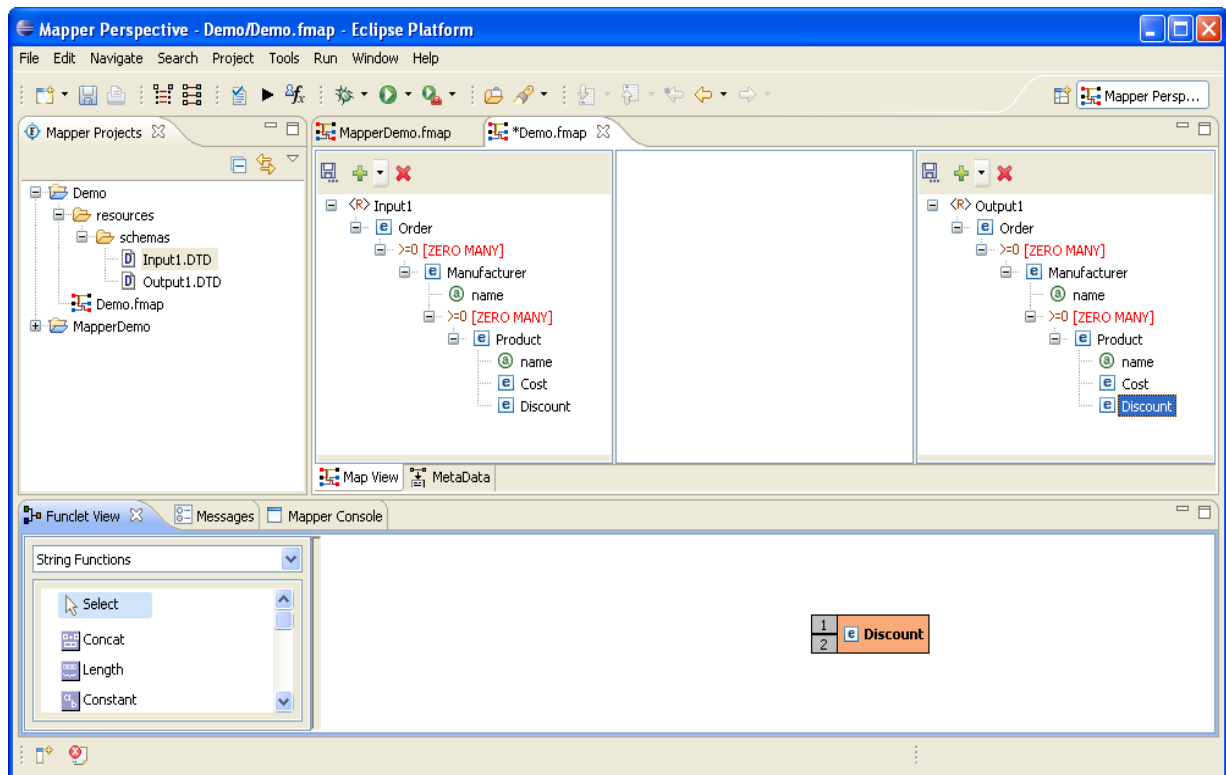


Figure 10.5.12: Selecting the Output Node for Mapping

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.

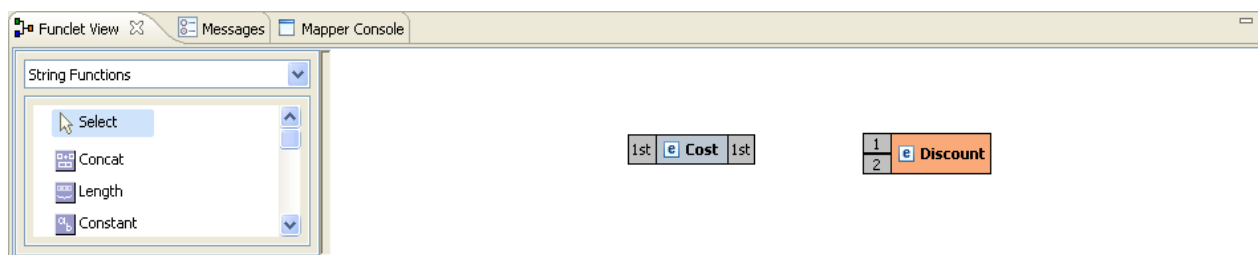


Figure 10.5.13: Dragging an Input node

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.

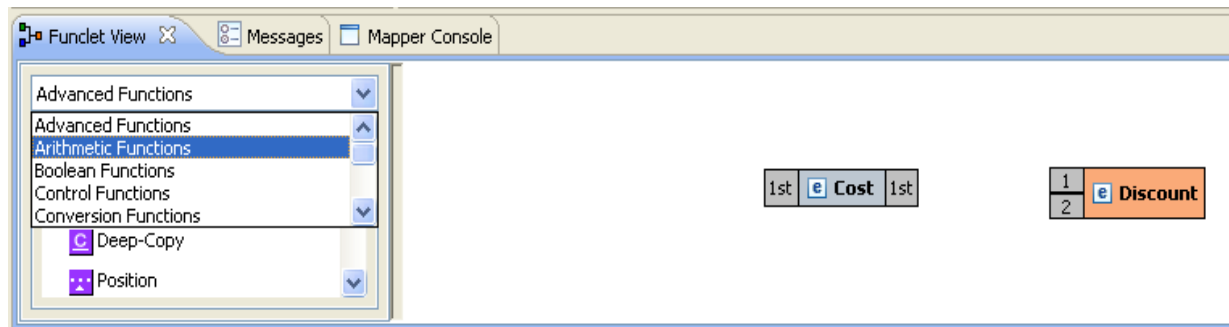


Figure 10.5.14: Selecting the Arithmetic Function Category in the Funclet palette

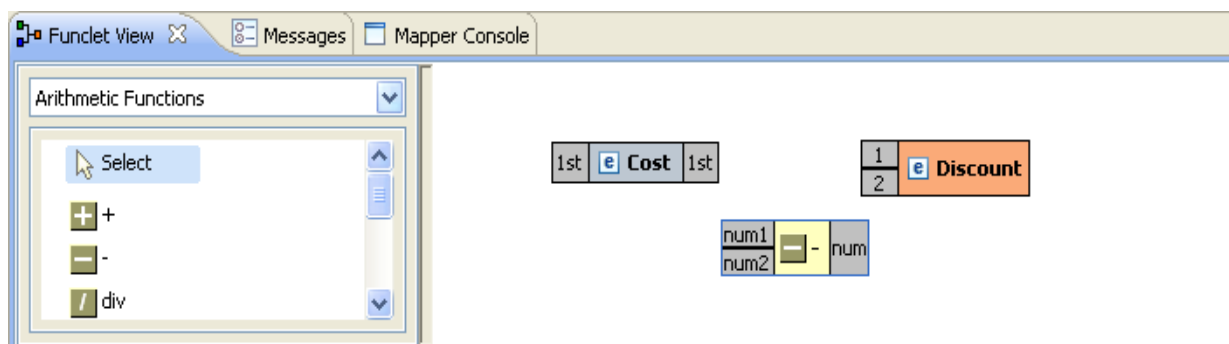


Figure 10.5.15: Adding the Subtract function

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

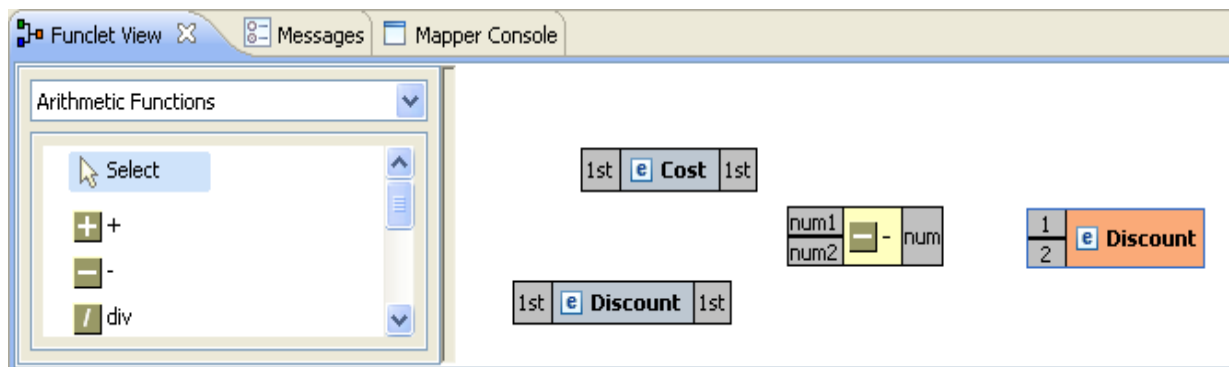


Figure 10.5.16: Adding another input node

- 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.

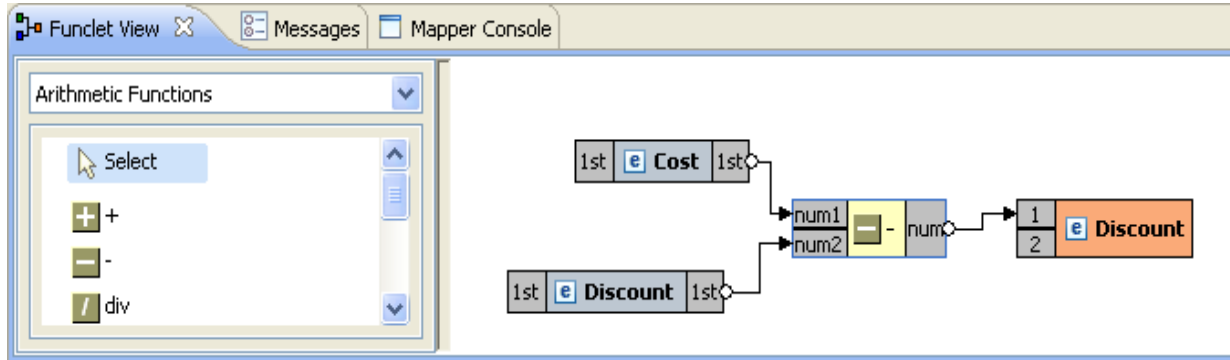


Figure 10.5.17: The final mapping is defined

- 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.

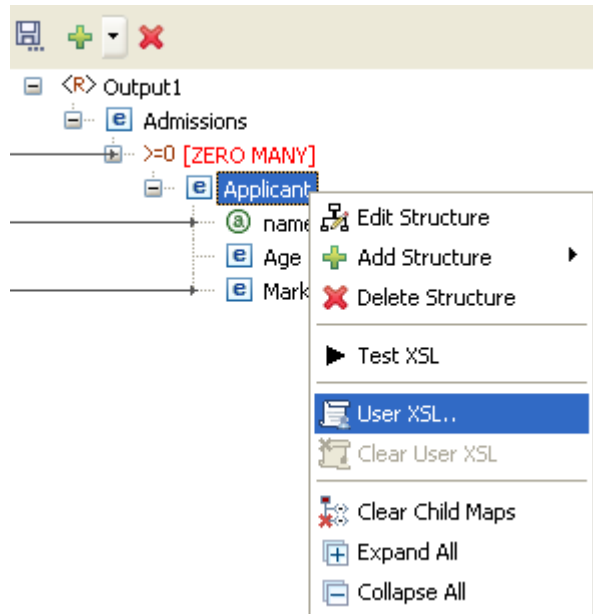


Figure 10.6.1: Selecting the User XSL option from the context menu

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.

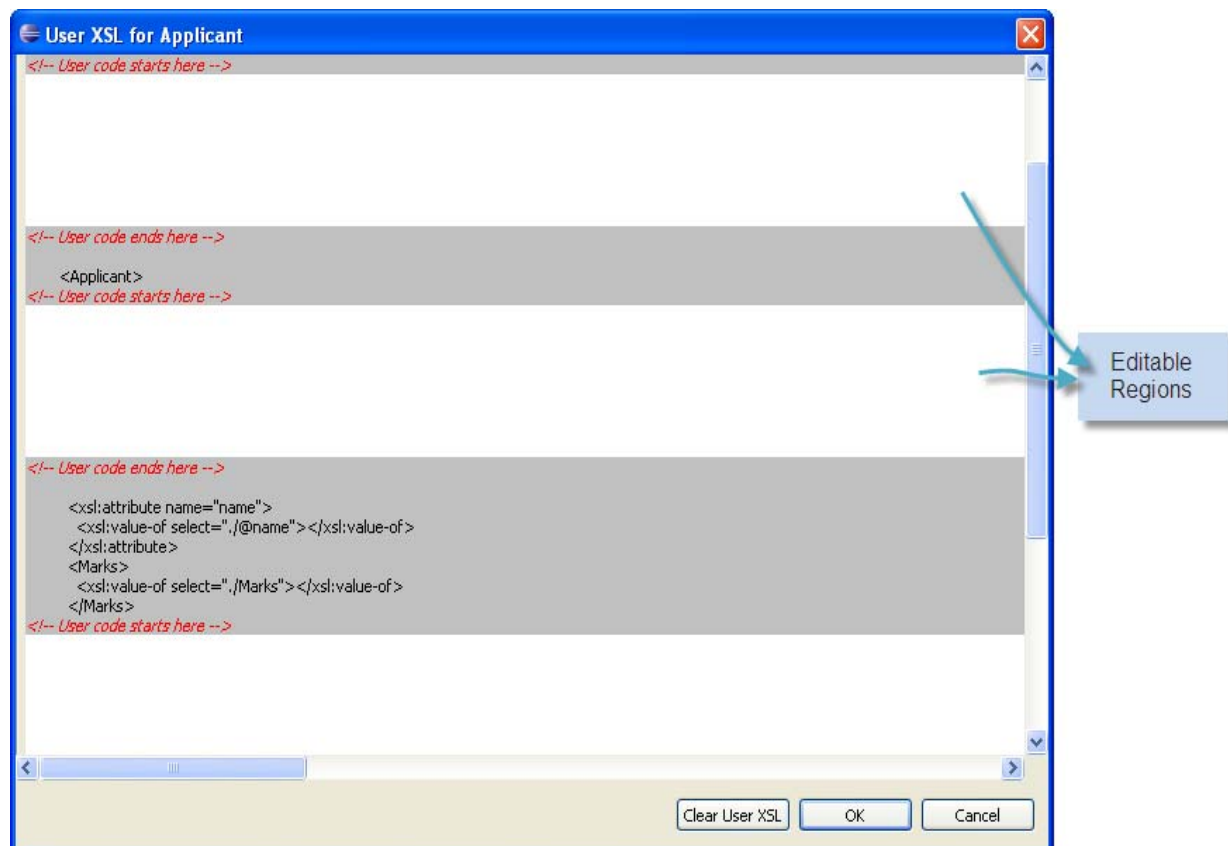


Figure 10.6.2: Editing the user xsl

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.

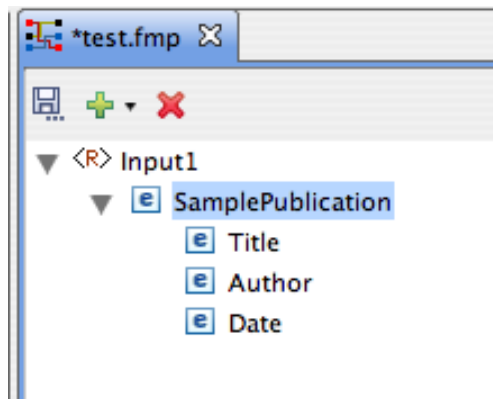


Figure 10.7.2: Sample Publication with default type

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.

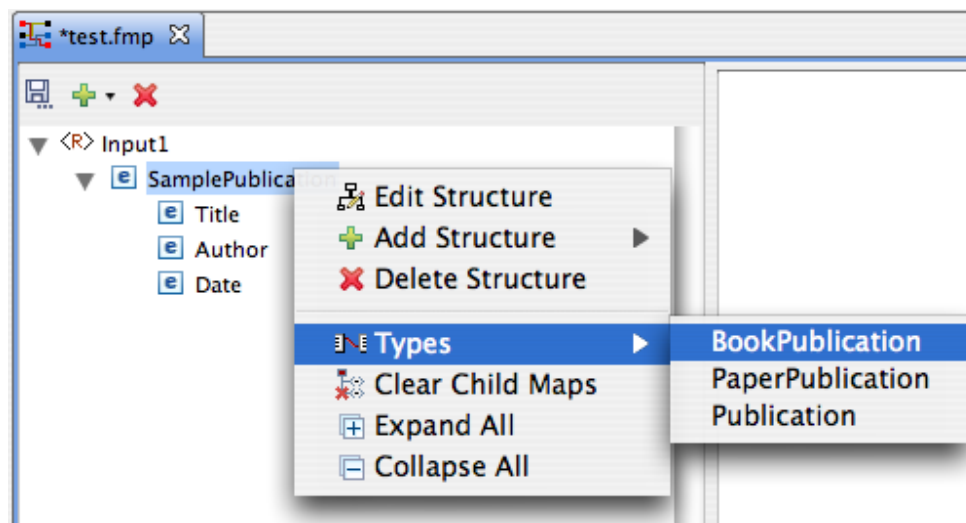


Figure 10.7.3: Available derived types

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

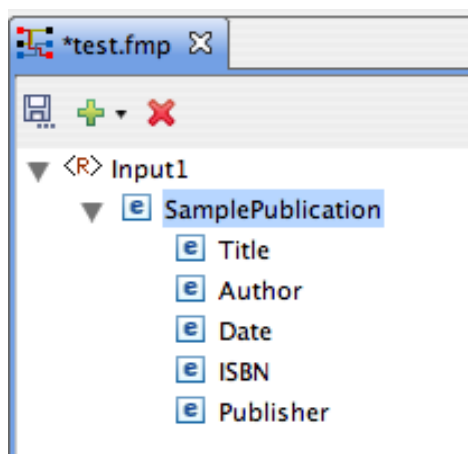


Figure 10.7.1: SamplePublication element when BookPublication type is chosen

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.

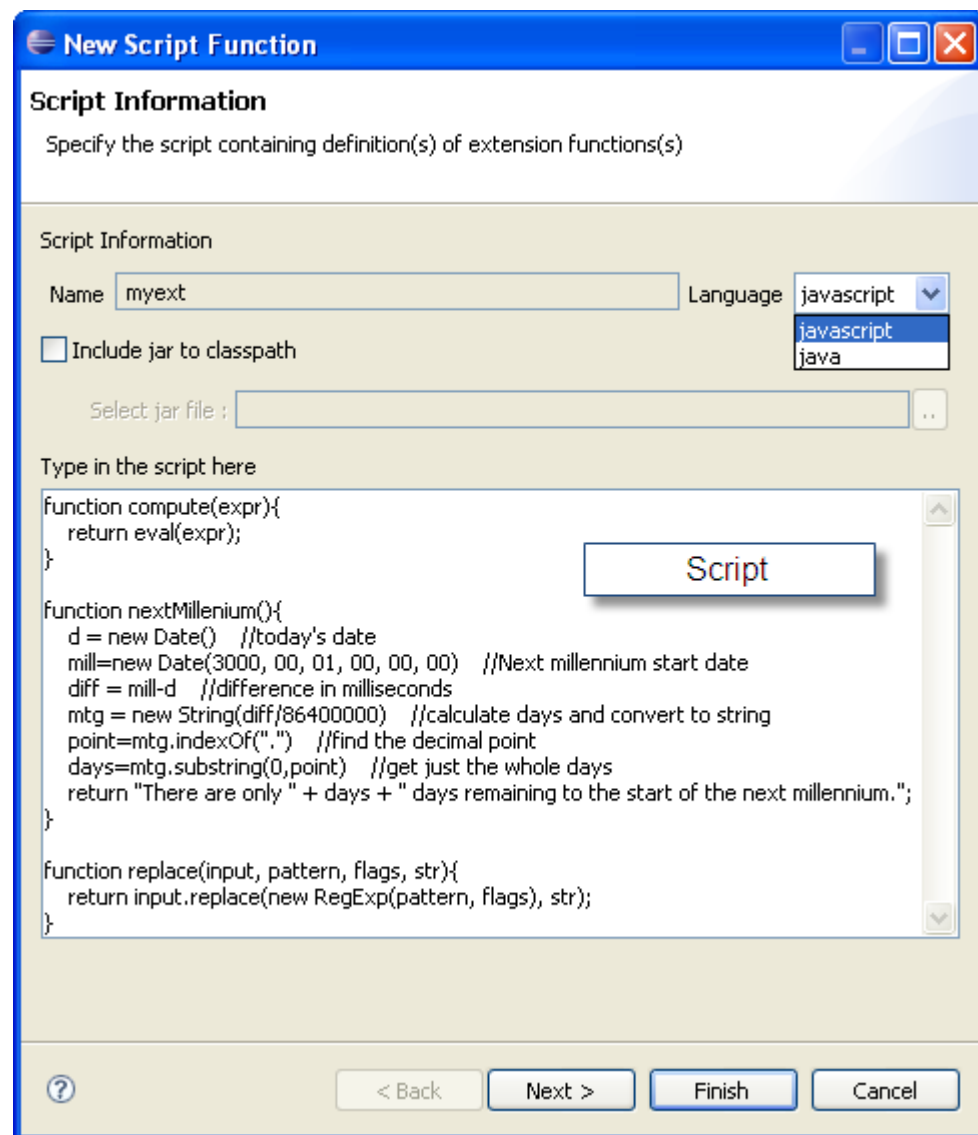
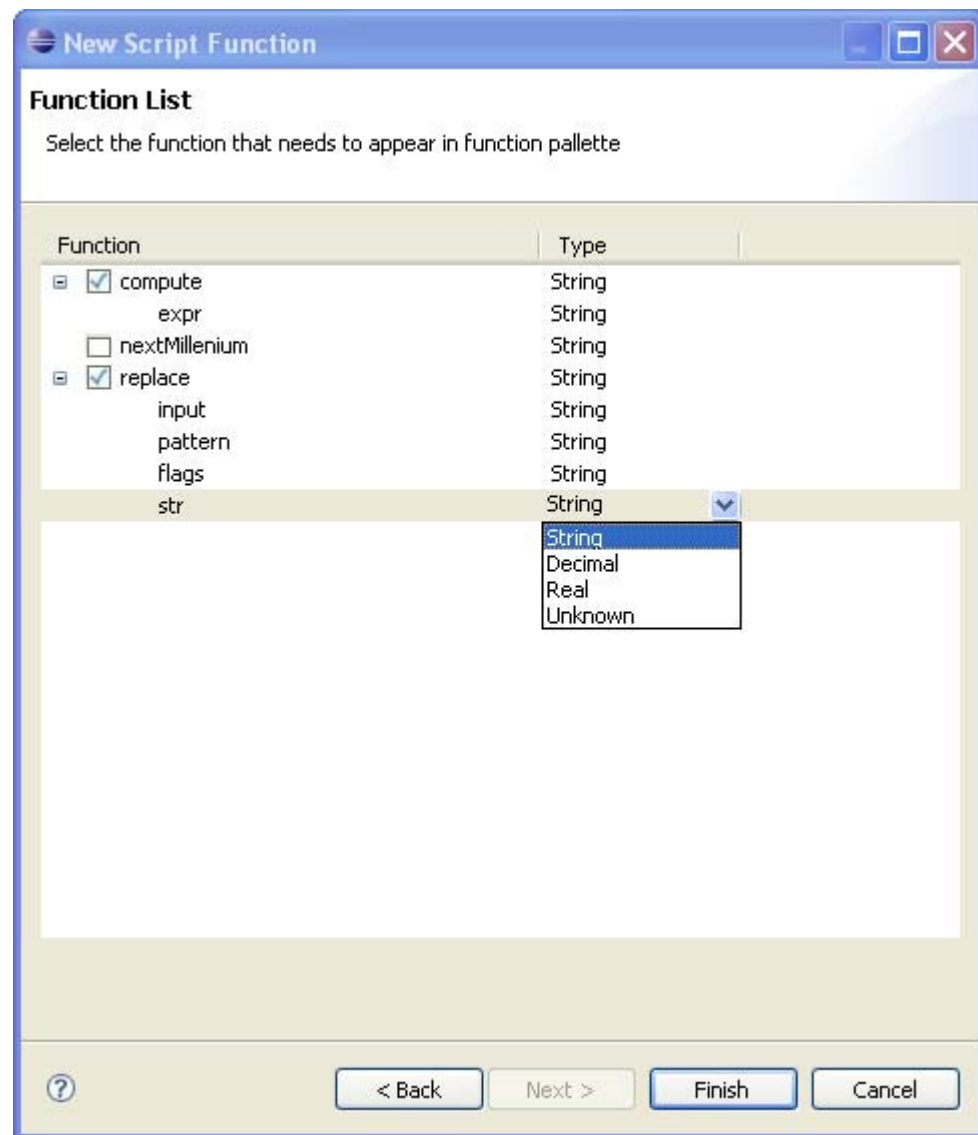


Figure 10.8.2 Script Information Page

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.

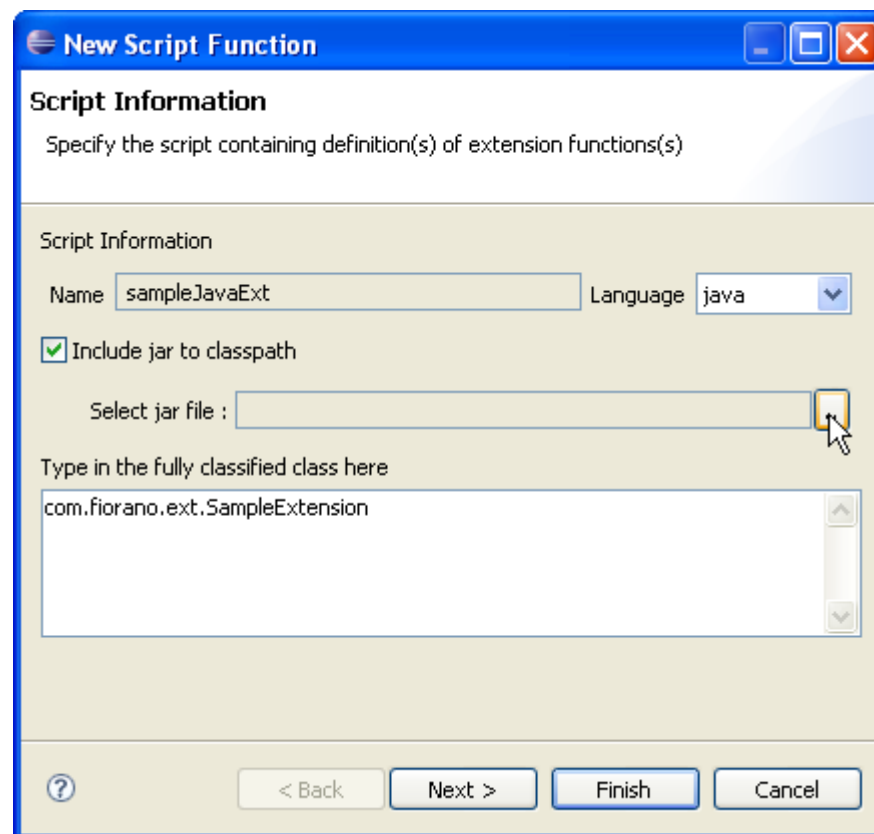


Figure 10.8.4 Adding Jar to classpath

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

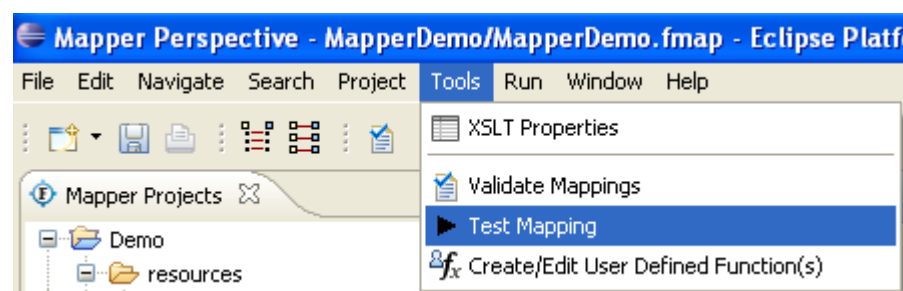


Figure 10.9.1: Invoking the Test option

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.

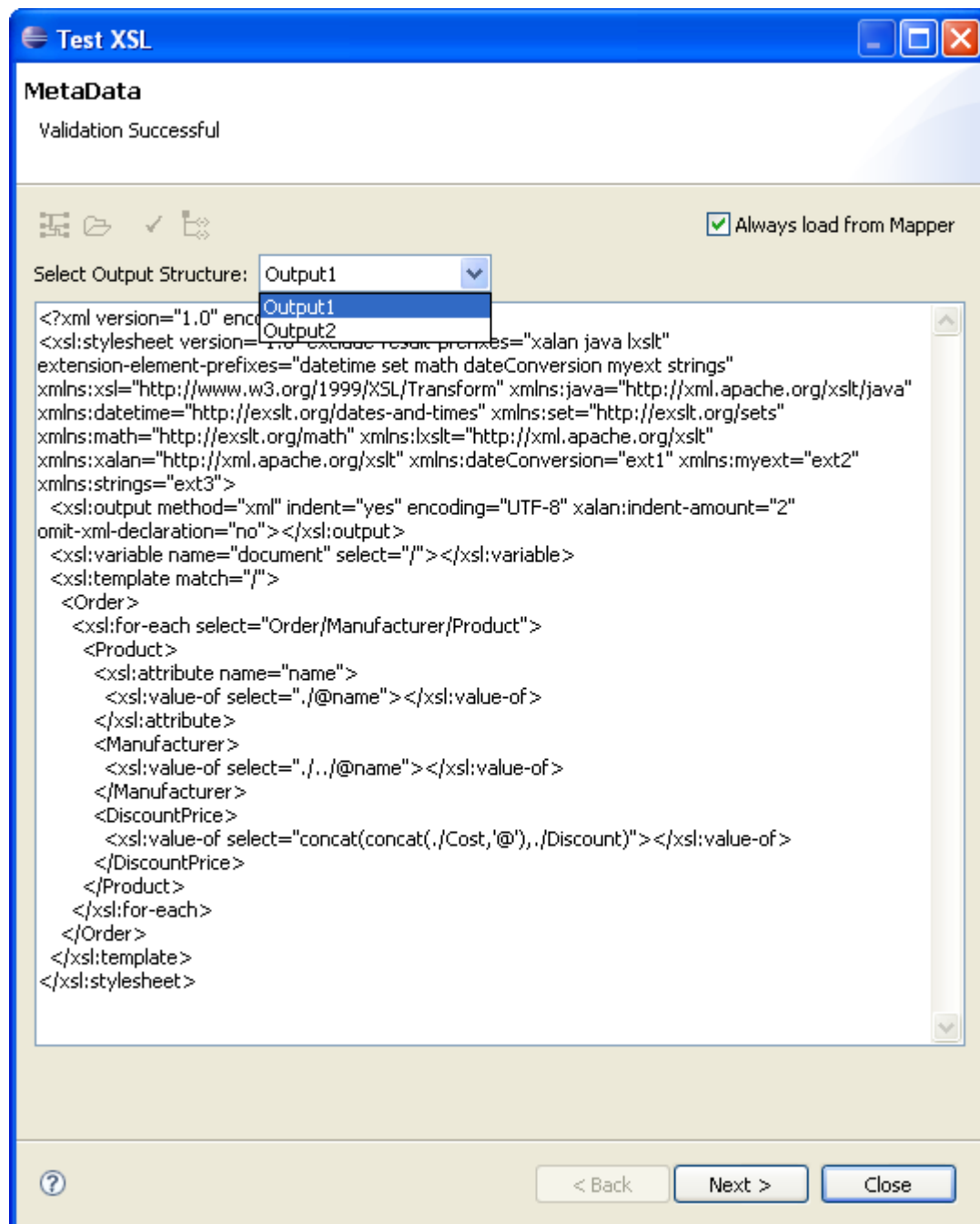


Figure 10.9.2: Metadata Page

- 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.

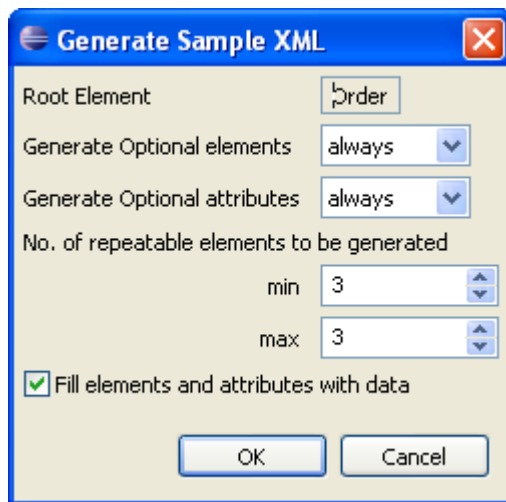


Figure 10.9.3: Selecting the sample Input XML generation options

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

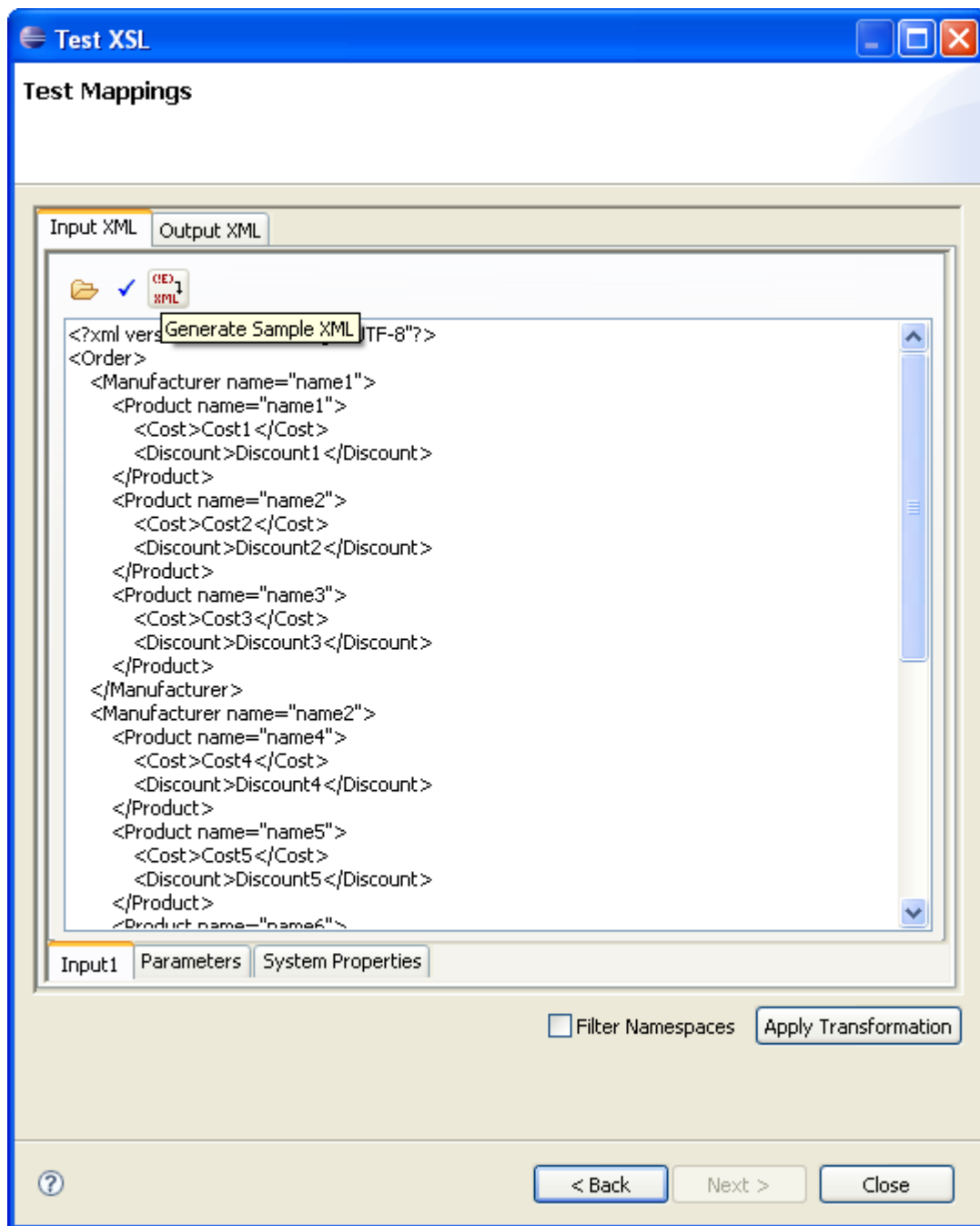


Figure 10.9.4: A Sample Input XML

- 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.

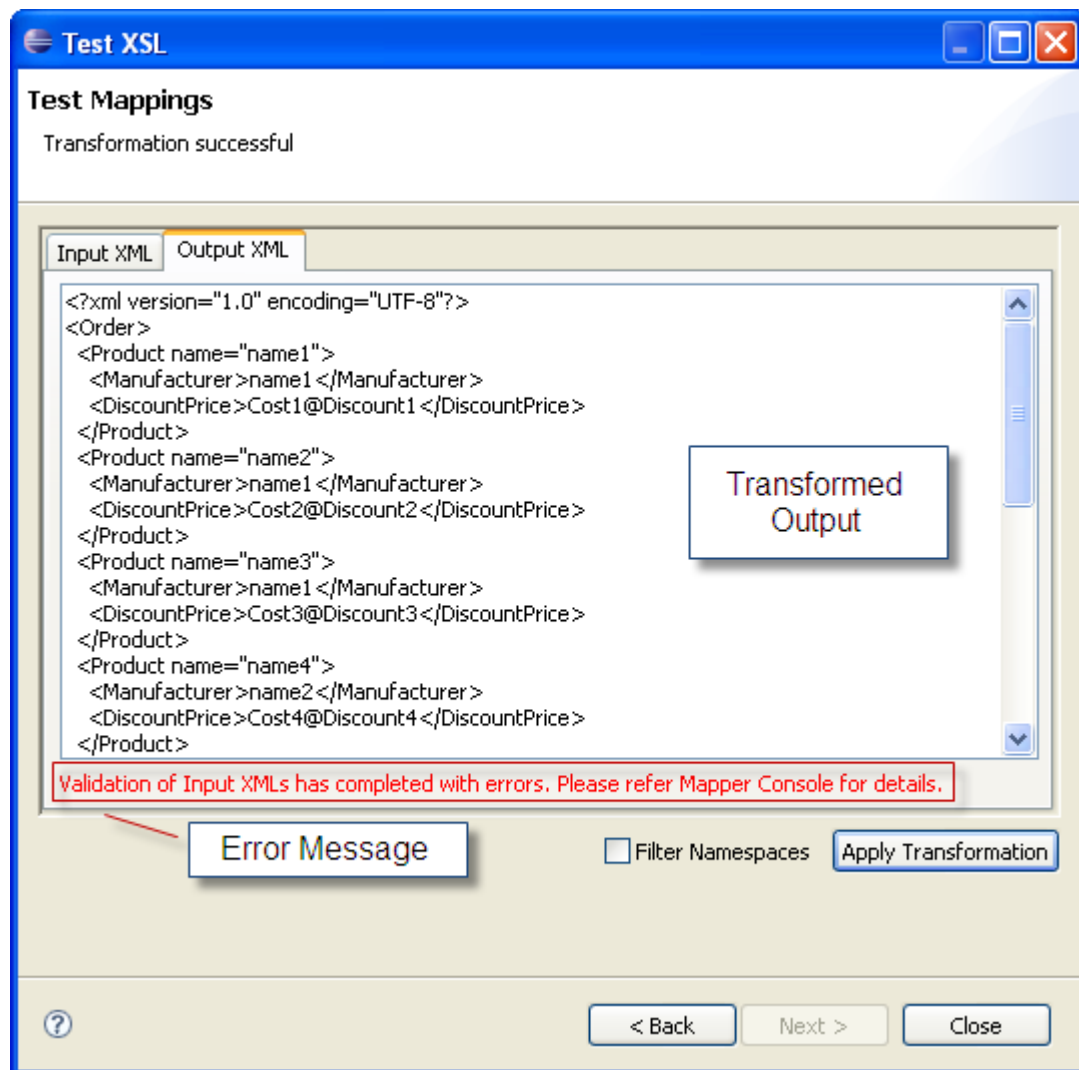


Figure 10.8.5: The Output XML resulting from the Transformation

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, perform the following steps:

1. Click **File > Export** or Right-click on the project to be exported in the eMapper Projects explorer and choose **Export**.
2. To export the project as an archive file select **General > Archive File** as the export destination.
3. Enter the file name in which you want to export the project and click on **OK** button.
4. The project gets as an archive file.
5. The project folders can also be exported as it is to the local file system by selecting **General > File System** in the Export wizard

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. Click **File > Import**. The Import wizard is shown.
2. Choose the appropriate import source. (Archive File or Existing Projects into Workspace depending on the source of import).
3. Provide the location of the source and the project is imported to the workspace.

To import mappings from a .tmf file:

1. Click **File > New > Fiorano Map**
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 funclet view and click **Copy** from the right-click menu as shown below in Figure 10.10.1.

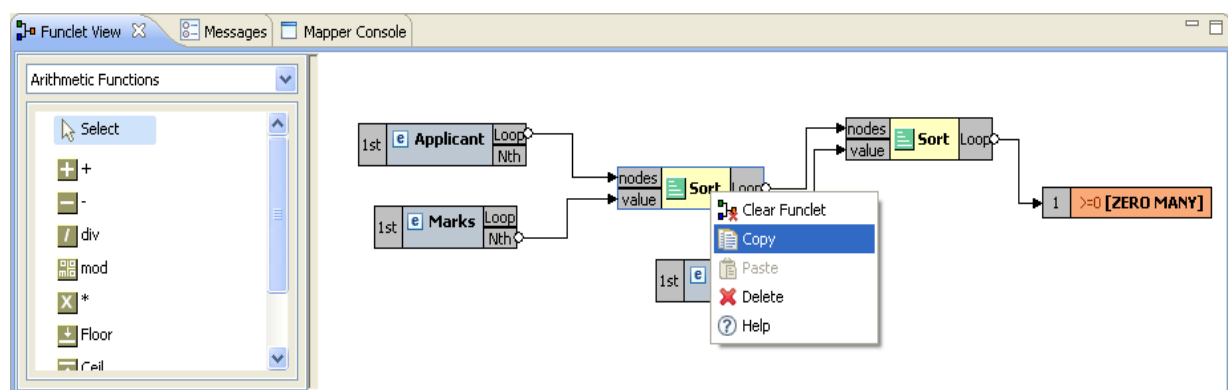


Figure 10.10.1: Copying a function

Click Paste and the function is pasted in the funclet 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.

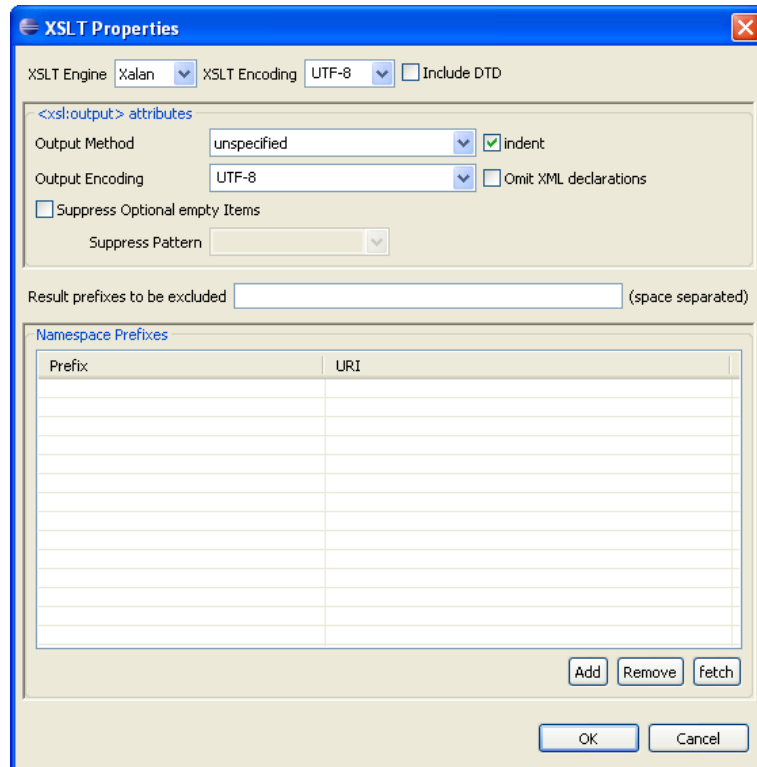


Figure 10.10.2: Viewing XSLT Properties

This dialog box contains the following components:

1. **XSLT Encoding:** Specifies the encoding of the generated XSL.
 2. **Include DTD:** Select this option to include the internal specified DTD in the transformation output. This option is disabled by default.
- **<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.
 - **Omit-xml-declaration:** Specifies whether the output XML generated should contain XML declarations or not.
 - **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.

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.

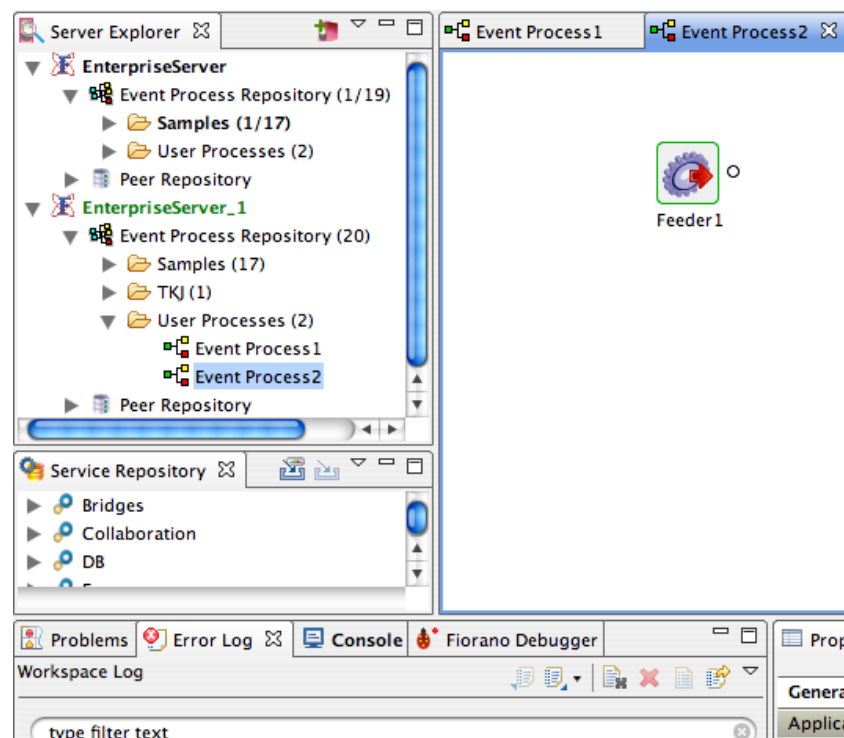


Figure 11.1.1: Multiple Enterprise Server login

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.

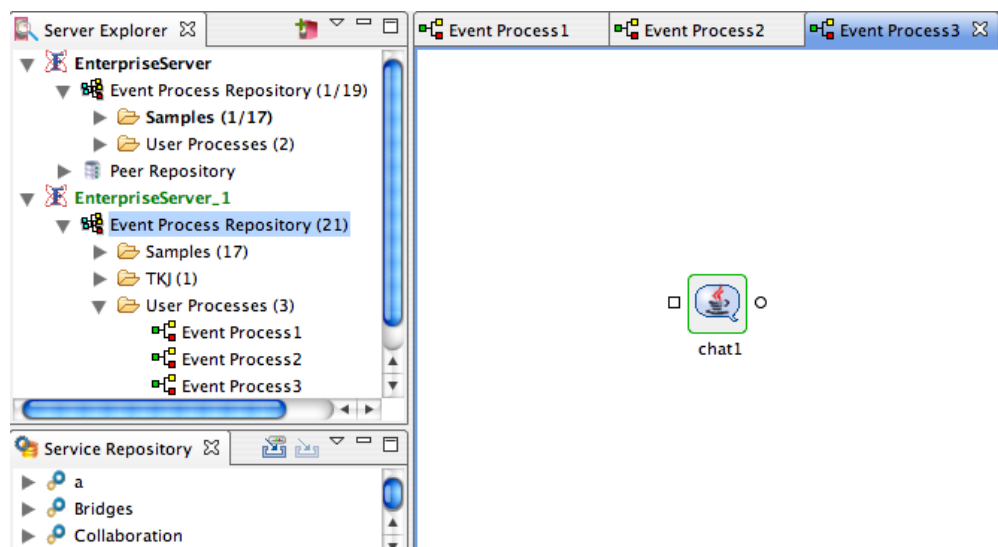


Figure 11.2.1: Active Enterprise Server

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.

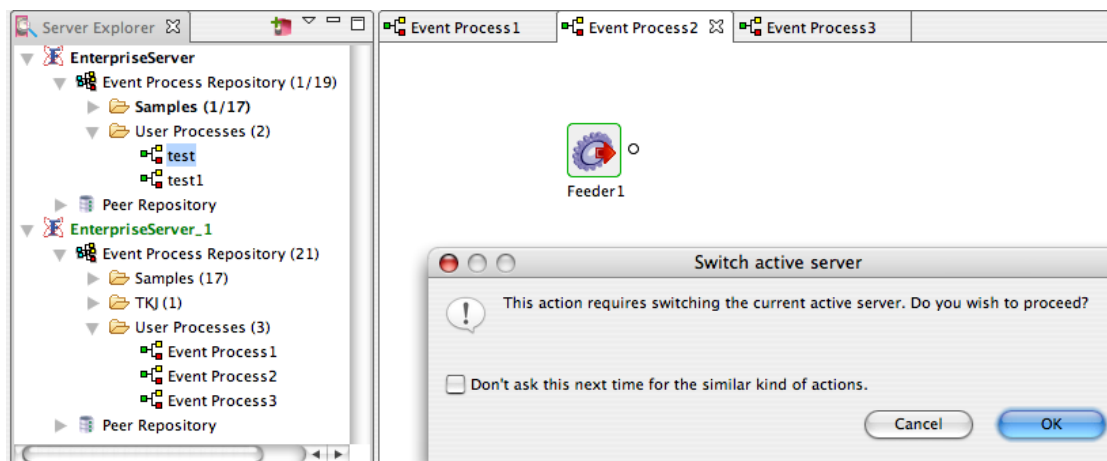


Figure 11.2.2: Switch Active Server

- 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.

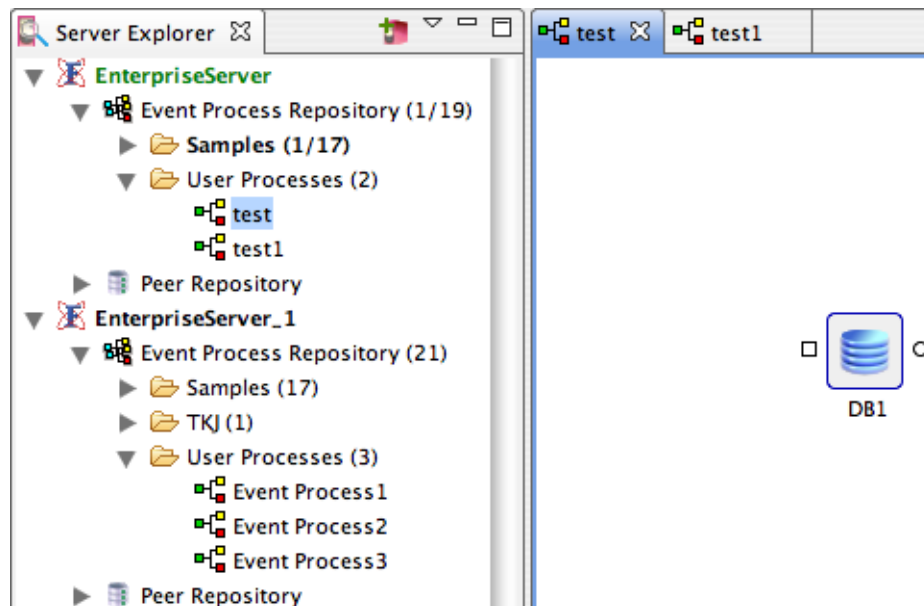


Figure 11.2.3: Event Processes present in Enterprise Server

- 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.

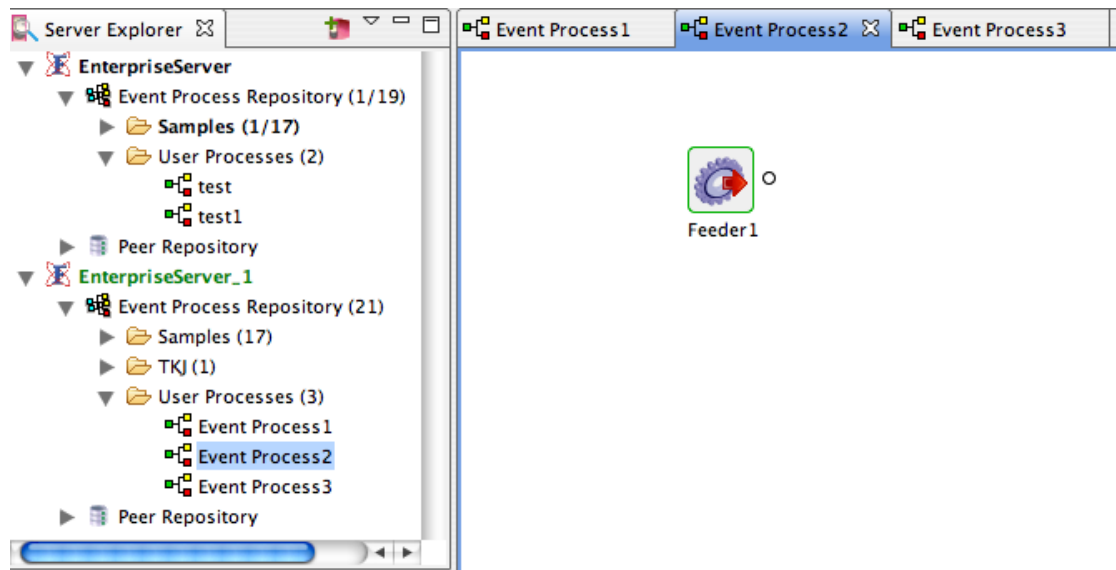


Figure 11.2.4: Services restored in Enterprise Server_1

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.

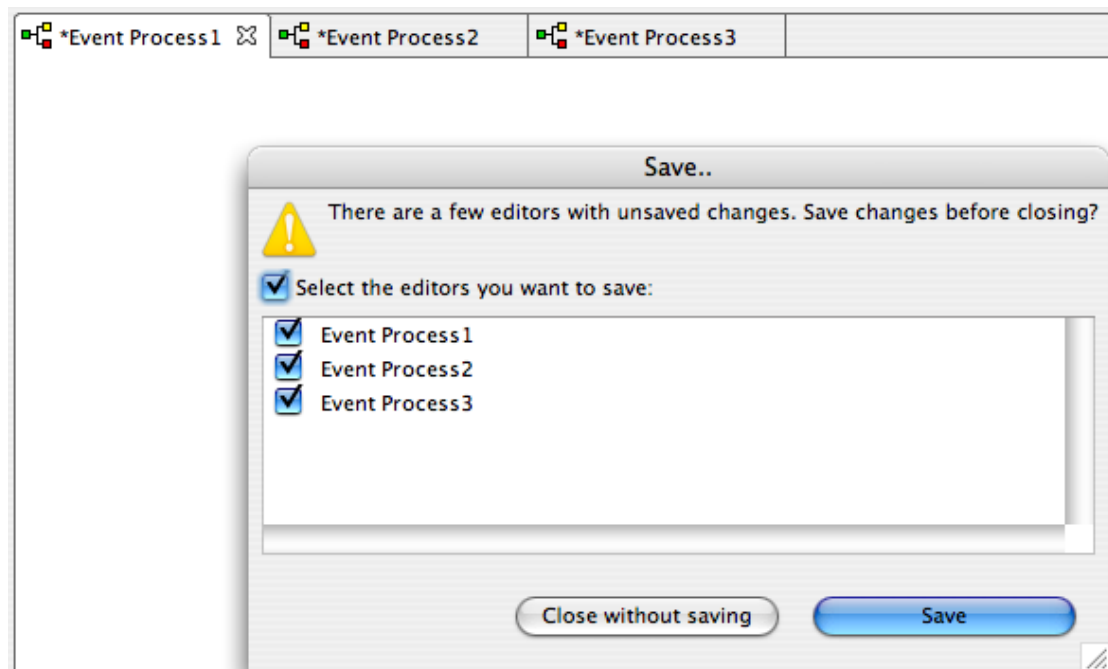


Figure 11.2.5: Unsaved editors dialog

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.

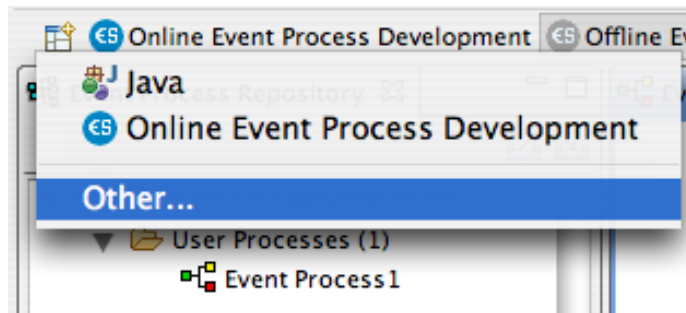


Figure 11.3.1: Selecting the Other.. option from perspective button

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

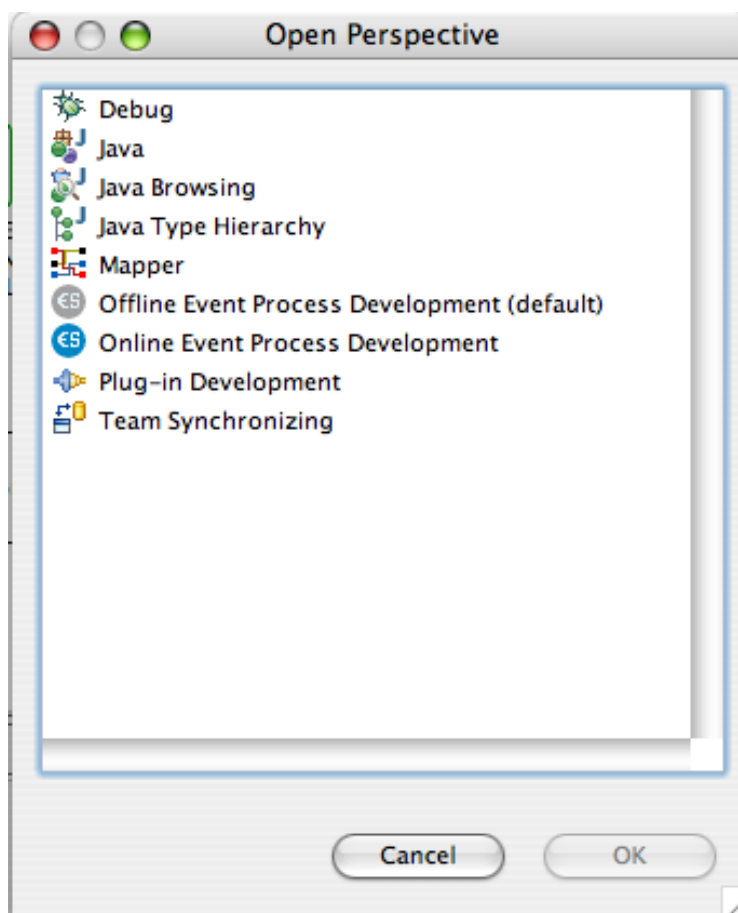


Figure 11.3.2: Selecting Online Application Development perspective

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.

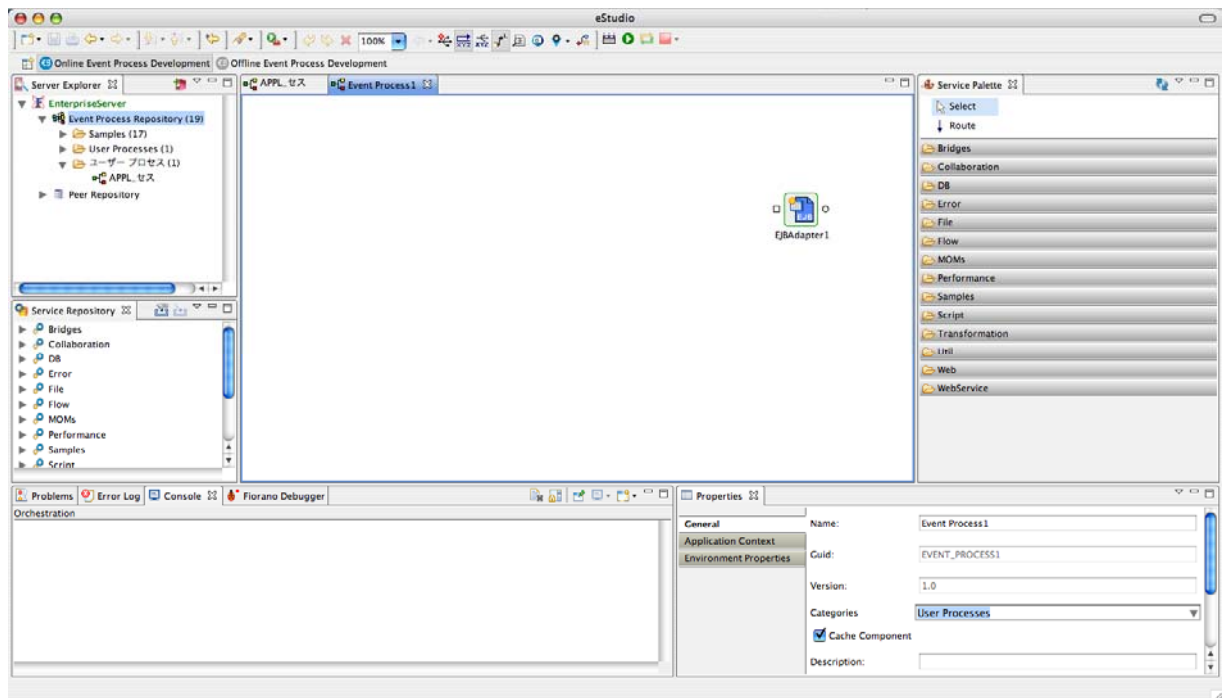


Figure 11.3.3: Online Application Development Perspective

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.

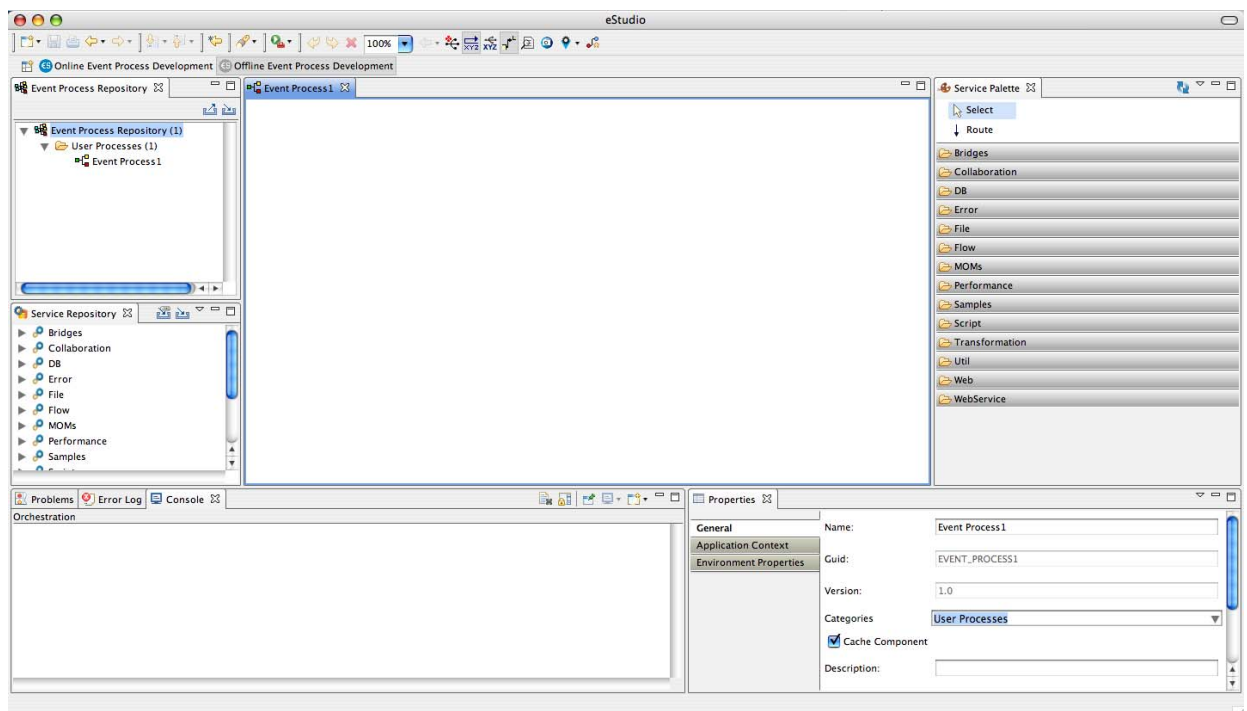


Figure 11.3.4: Offline Application Development Perspective

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.

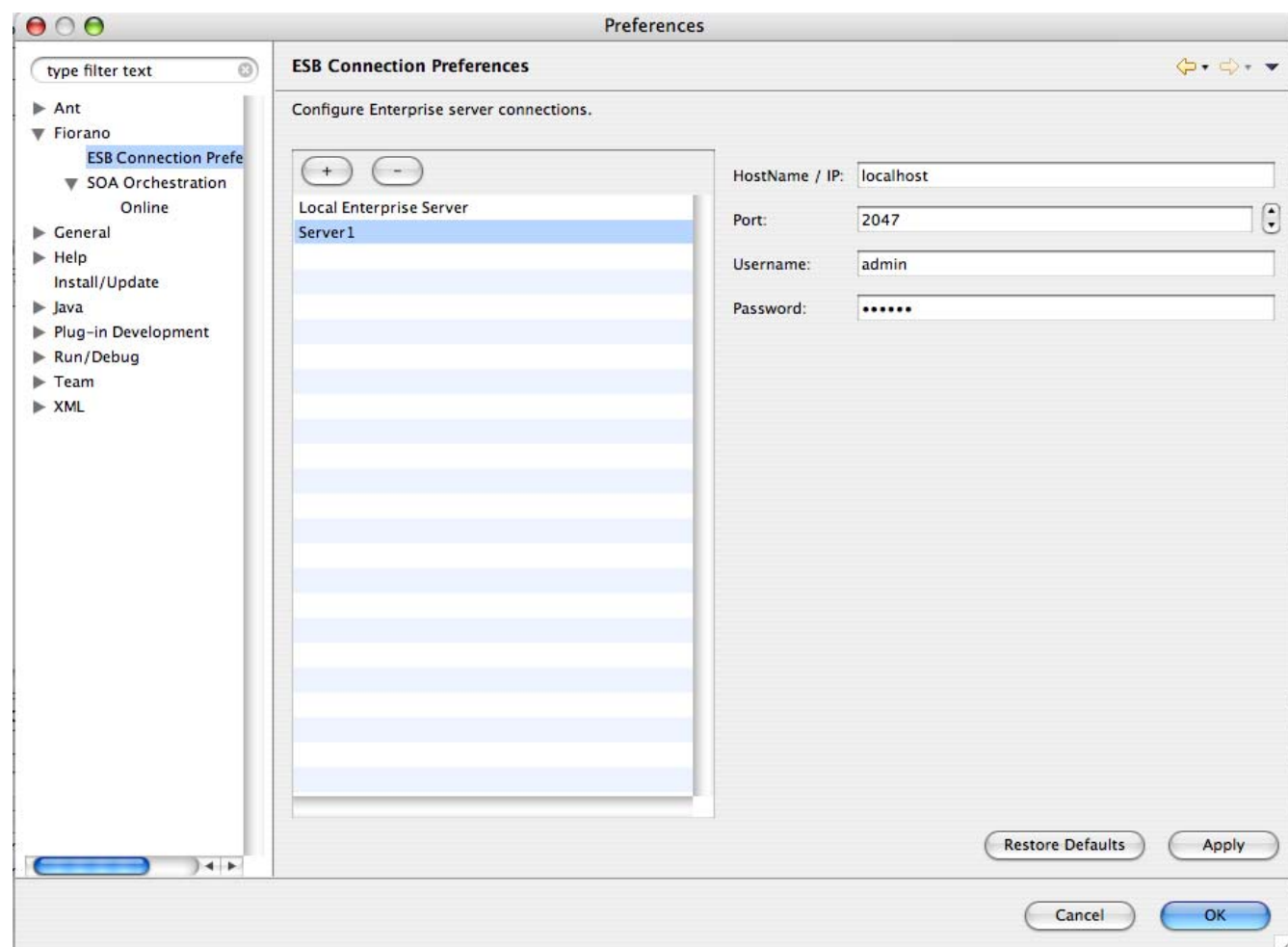


Figure 12.1.1: Enterprise Server Configurations

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.

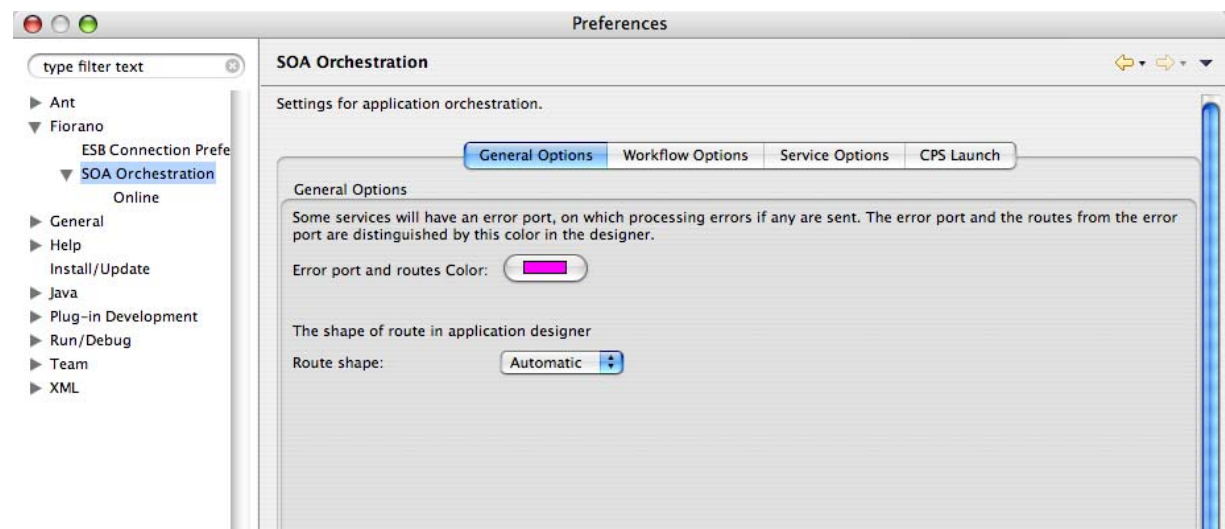


Figure 12.2.1: SOA Orchestration Preferences

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.

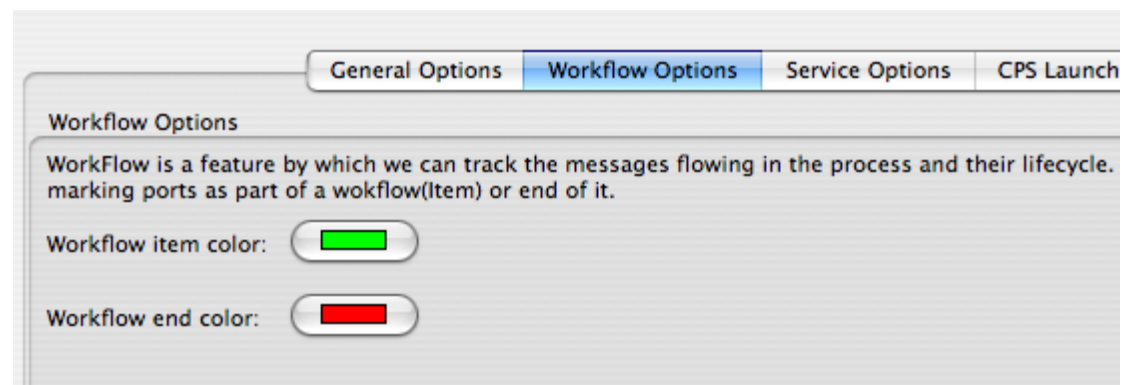


Figure 12.2.2: Workflow options

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.

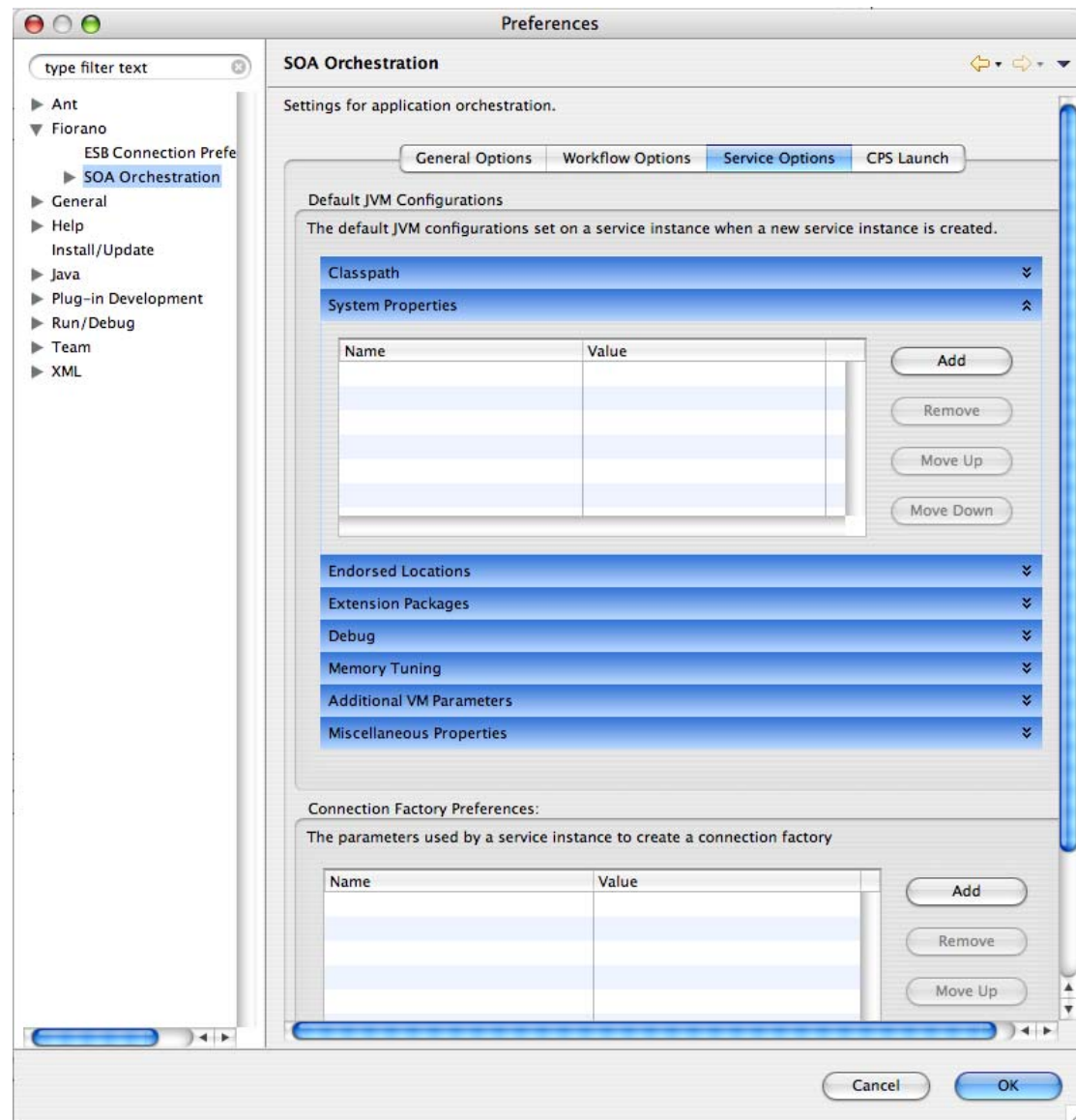
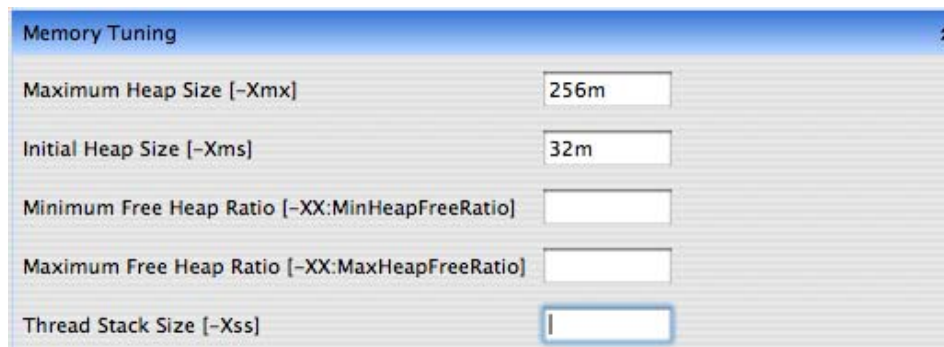


Figure 12.2.3: Service options

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

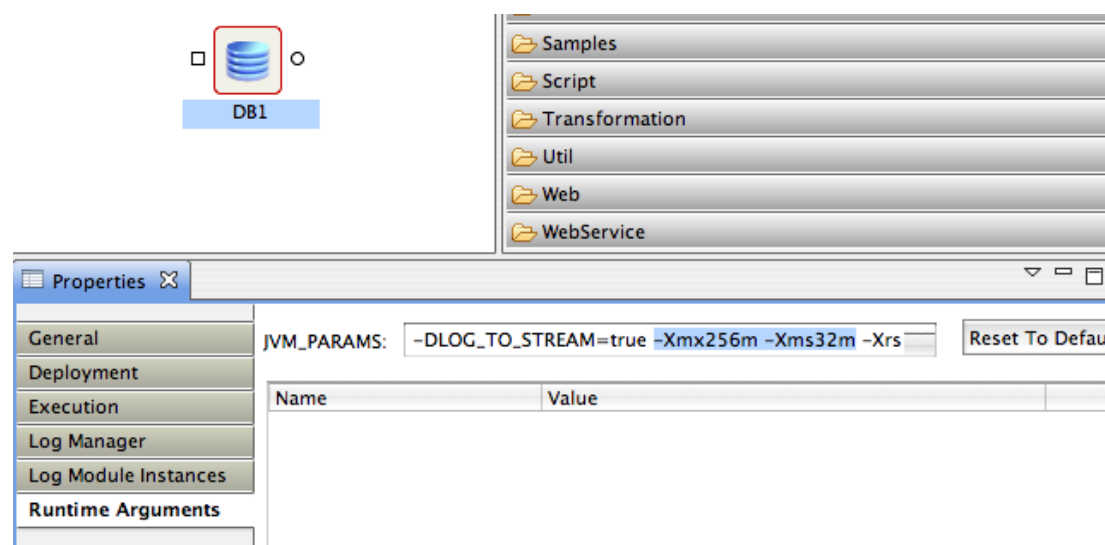


Memory Tuning

Maximum Heap Size [-Xmx]	256m
Initial Heap Size [-Xms]	32m
Minimum Free Heap Ratio [-XX:MinHeapFreeRatio]	
Maximum Free Heap Ratio [-XX:MaxHeapFreeRatio]	
Thread Stack Size [-Xss]	

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.



DB1

- Samples
- Script
- Transformation
- Util
- Web
- WebService

Properties

General

Deployment

Execution

Log Manager

Log Module Instances

Runtime Arguments

JVM_PARAMS: -DLOG_TO_STREAM=true -Xmx256m -Xms32m -Xrs

Reset To Defau

Name	Value
------	-------

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.

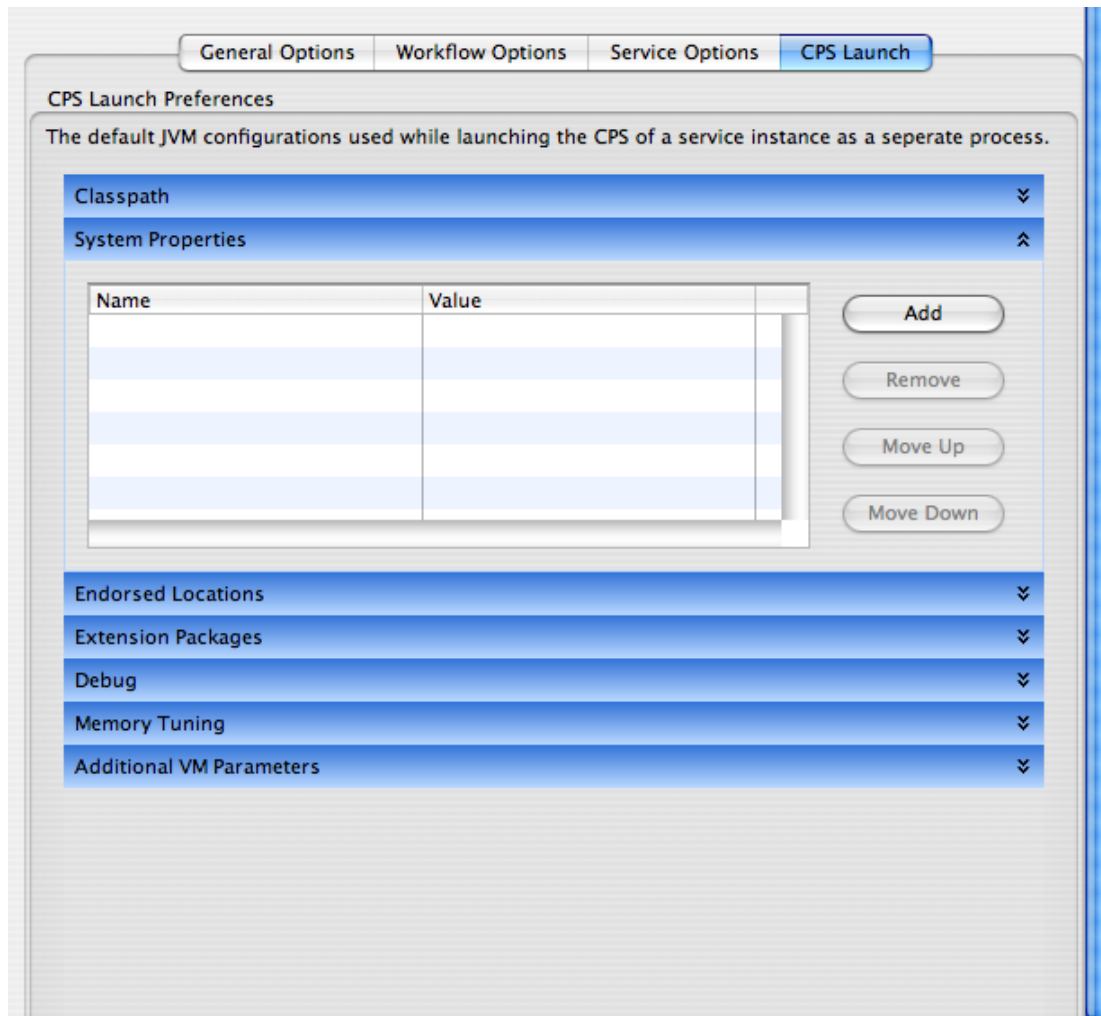


Figure 12.2.6: CPS launch options

12.3 SOA Orchestration Online

This section contains configurations for online Event Process orchestration.

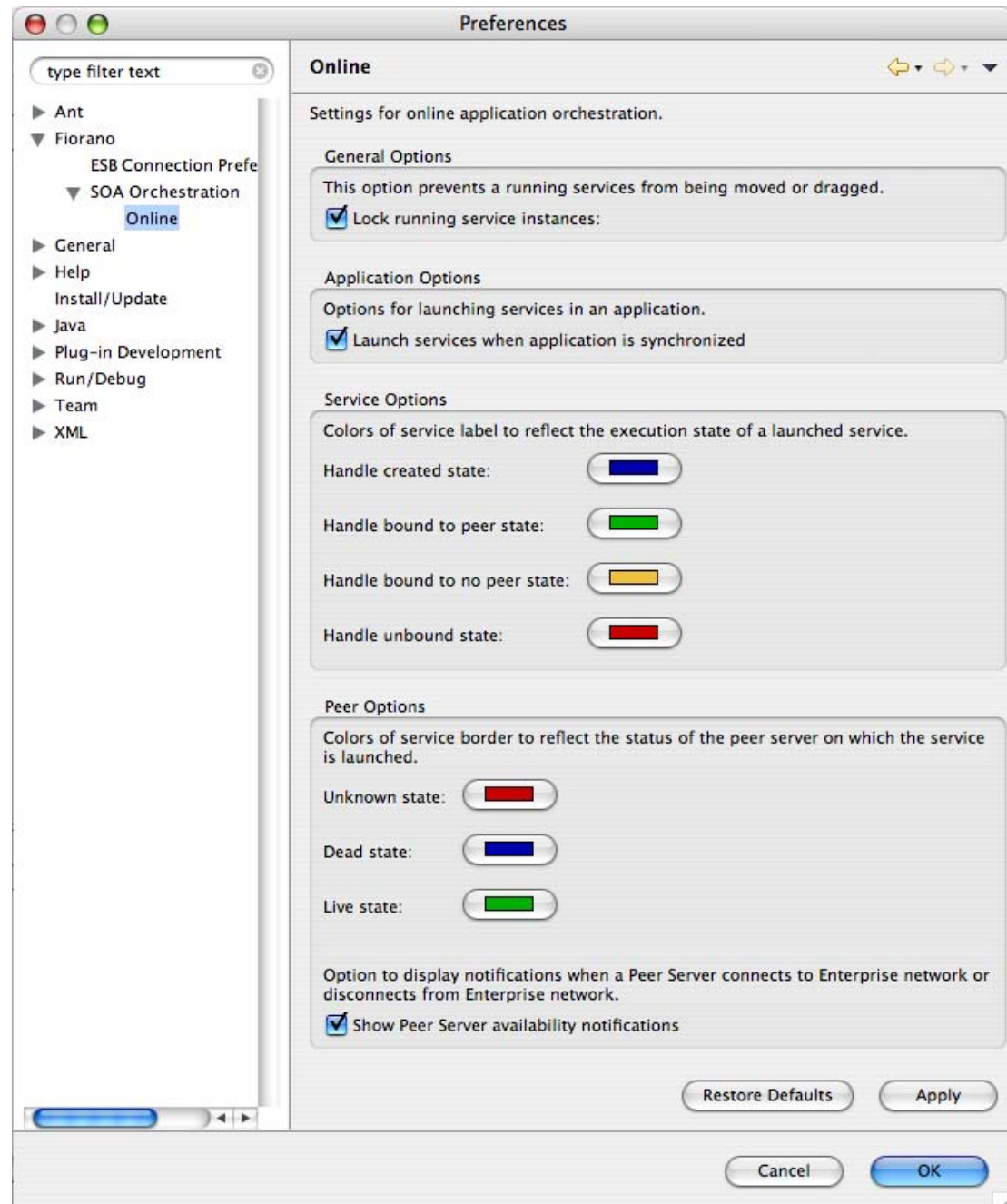


Figure 12.3.1: Online Orchestration preferences

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.

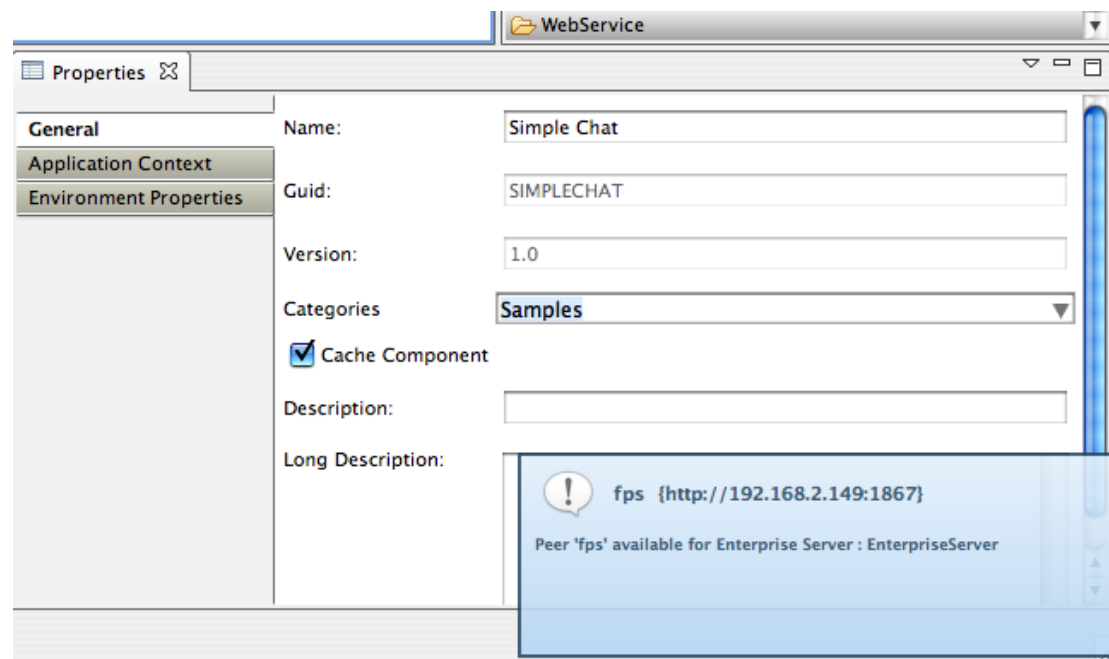


Figure 12.3.2: Peer server notification

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).

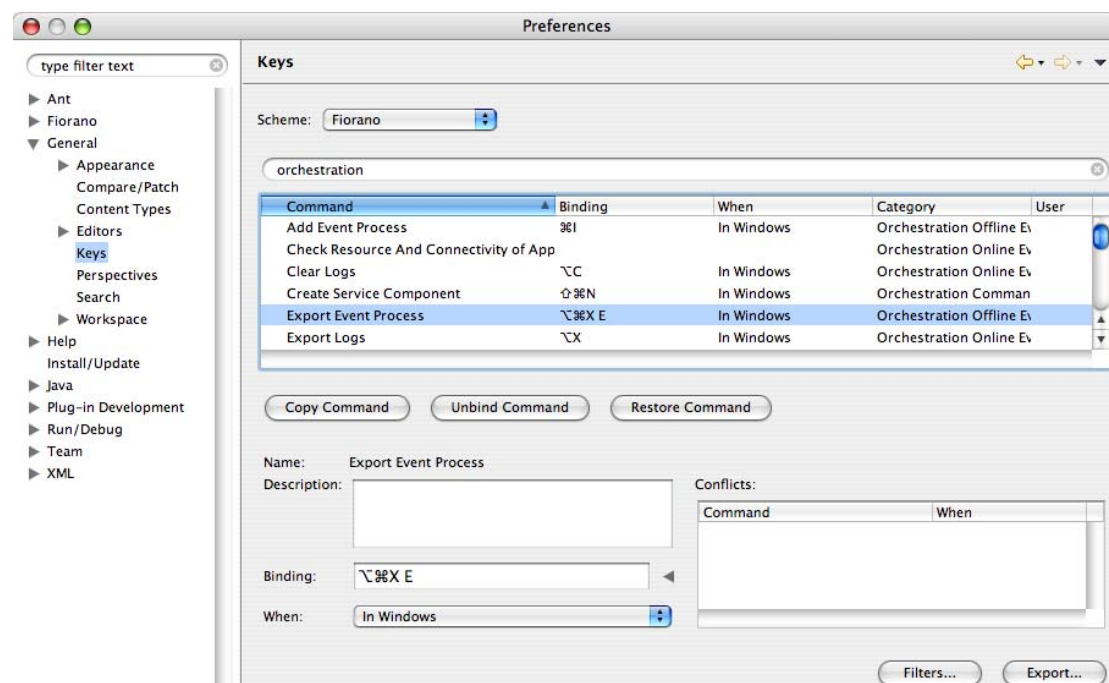


Figure 12.4.1: Key-binding preferences

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.

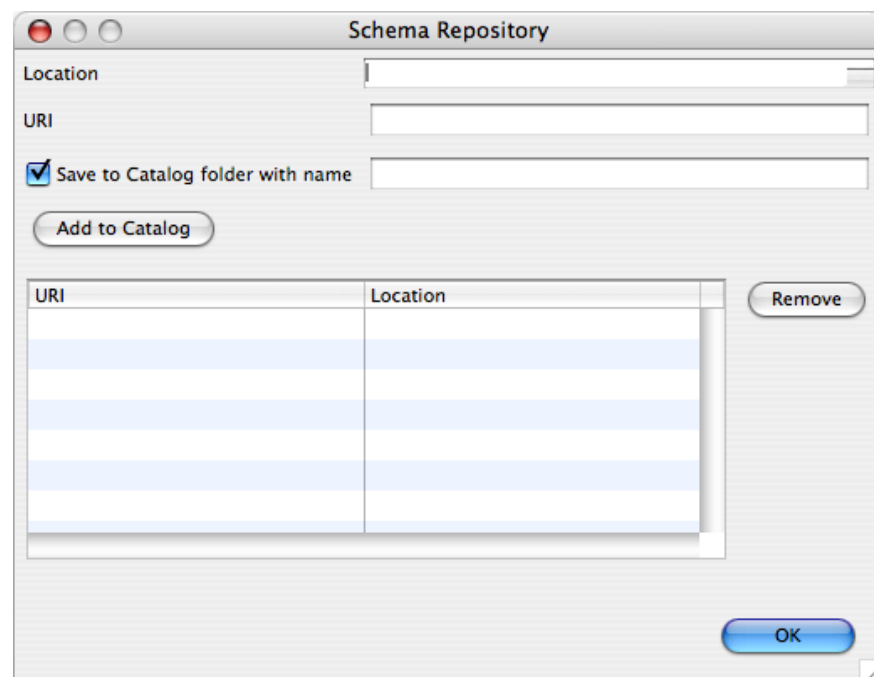


Figure 13.1.1: Schema Repository

2. Click on ellipsis button against the **Location** property. A file chooser dialog is opened where the location of schema file can be selected. Select the file and click Open.
3. **URI** and **Save to Catalog folder with name** fields are populated automatically.

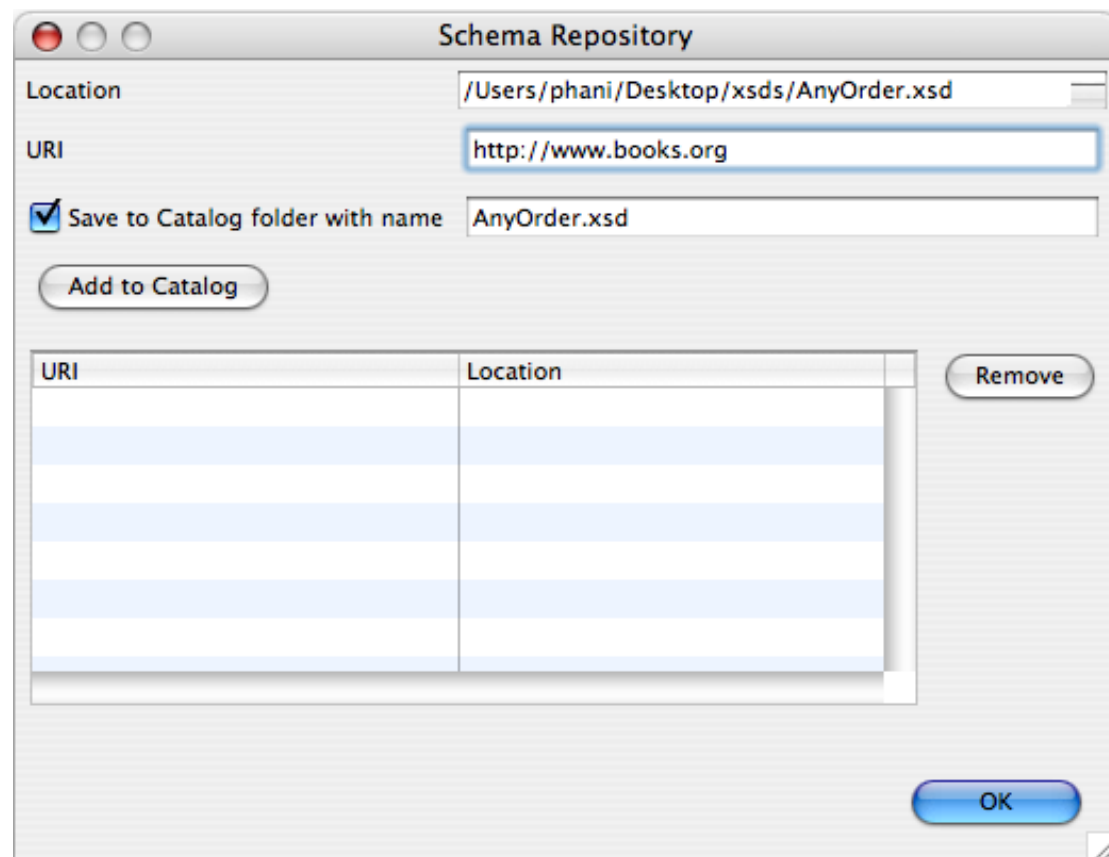


Figure 13.1.2: Adding schema to Schema Repository

The URI value should not be an empty field. In case, if the schema has a target namespace, URI should be same as the target namespace of the schema.

4. Click **Add to Catalog** button to add the schema to Schema Repository. The file is added to <FIORANO_HOME>/xml-catalog/user.

The Location field displays the absolute path of the schema file. If this dialog is closed without clicking Add to Catalog button, the file is not copied to the location <FIORANO_HOME>/xml-catalog/user and will be referred from its original location.

5. A new row specifying the URI and Location of the XSD will be added in the table.
6. To remove the schema from the schema repository, select a row from the table and click Remove.



Figure 13.1.3: Deleting schema from Schema Repository

7. 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

SCM is commonly known as version control, and is achieved using tools such as SVN, CVS etc. SCM integration support is present in eStudio. Event Processes and Services can be stored and retrieved from a version control system. The following steps explain the procedure to add required dependencies and work with SVN version control system. Similar steps can be followed for other tools.

14.1 Downloading and integrating SCM plugins in Fiorano eStudio

1. Download latest version of Subclipse for Eclipse 3.x version. This is available for download at <http://subclipse.tigris.org>.
2. Extract the **downloaded zip** file. In the extracted directory, two folders plugins and features can be found.
3. Copy the contents inside the plugins directory and paste them at \$FIORANO_HOME/eStudio/plugins.
4. Copy the contents inside the features directory and paste them at \$FIORANO_HOME/eStudio/features.
5. Restart eStudio. When eStudio comes back up, Subclipse is installed and ready to go.

14.2 Specifying SCM repository

To use version control, the URL of an existing svn repository has to be added. The following steps explain this procedure.

1. Open **SVN Repositories** view (Window -> Show View -> Other -> SVN -> SVN Repositories).
2. Right click and select **New -> Repository Location** option, specify the repository URL and Finish the wizard.

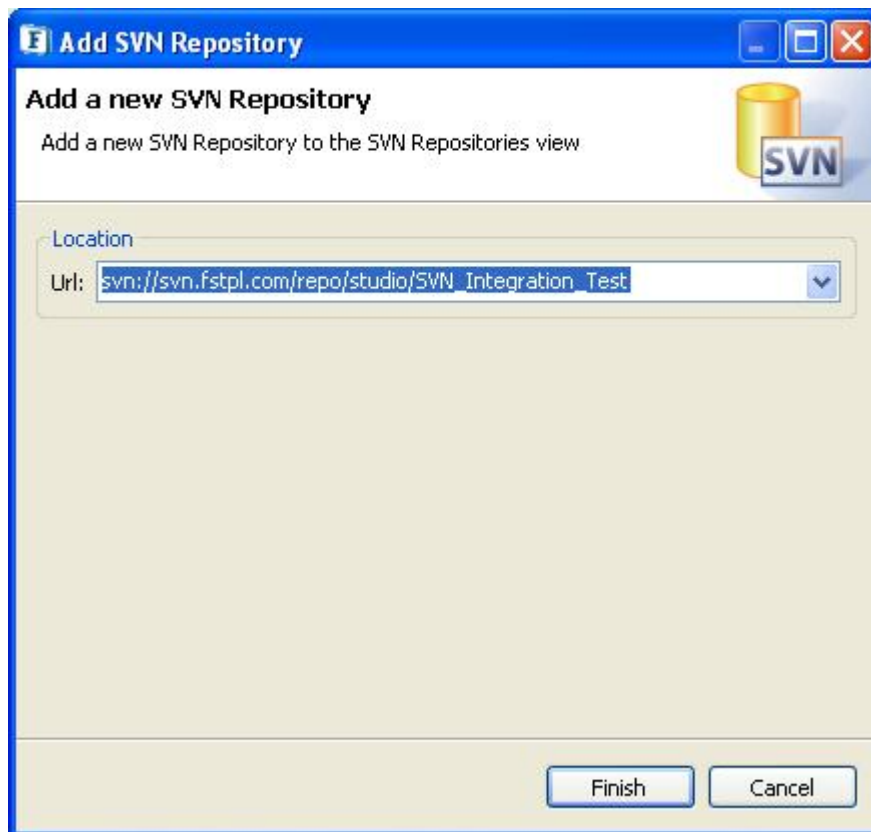


Figure 14.2.1: Add SVN Repository

3. The repository is added and is displayed in the SVN Repositories view.

14.3 Creating a project for version control

Currently version control in eStudio is achieved using an intermediate project. Event Processes or Services that have to be under version control have to be first exported to this project and this project is added to the svn repository. The following steps explain the procedure to create a project and export the Event Processes into the project.

1. Create a new project from the File menu (File -> New -> Other -> General -> Project).
2. Specify a project name (say EStudioSvnProjects) and click Finish.

Note: By default the project is created in eStudio work space (i.e. FIORANO_HOME/runtimedata/eStudio/workspace). If required the user can specify a new Location.

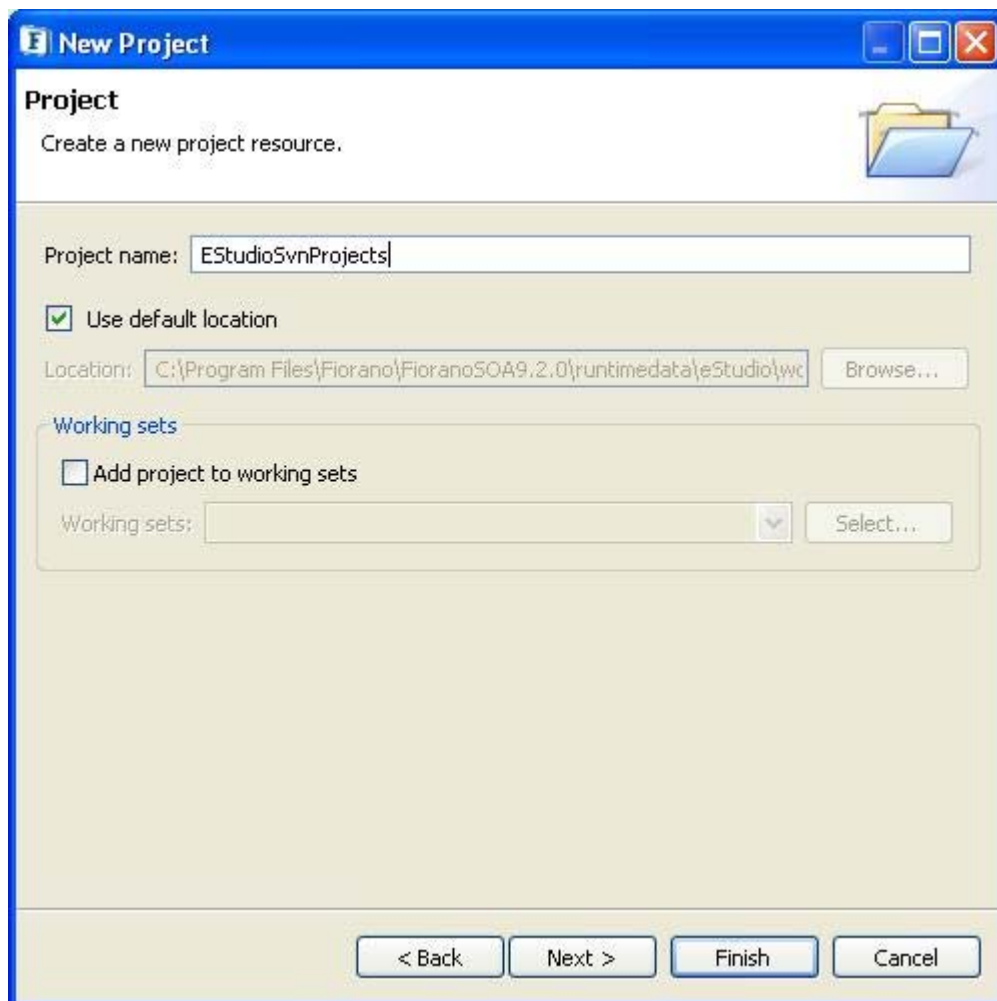


Figure 14.3.1: Creating a temporary project for SVN integration

3. The created project can be seen in **Project Explorer** view as shown in Figure 14.3.2.

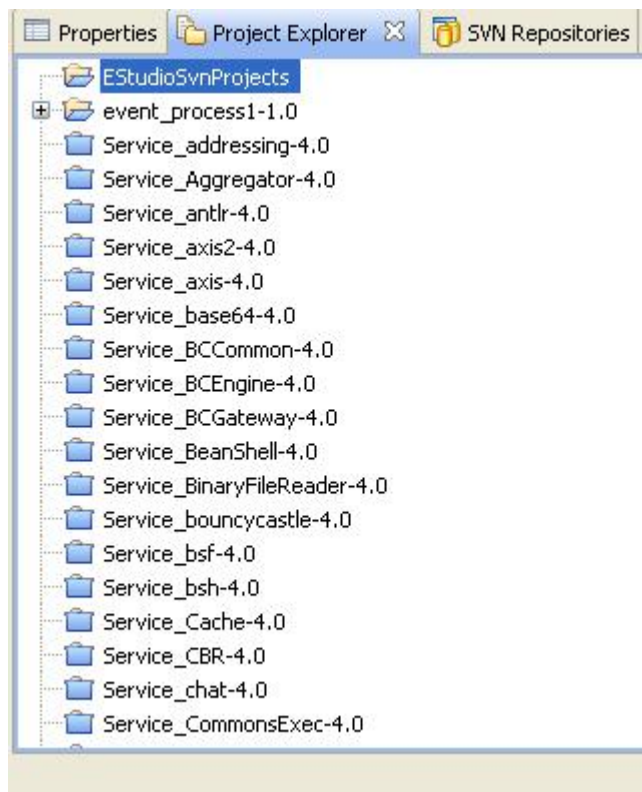


Figure 14.3.2: Project Explorer view

4. Select the Event Processes to be exported and choose **Export** option from the context menu. Specify the location of the **EStudioSvnProjects** created in step 2 and click Ok. The selected Event Processes are exported to **EStudioSvnProjects**.

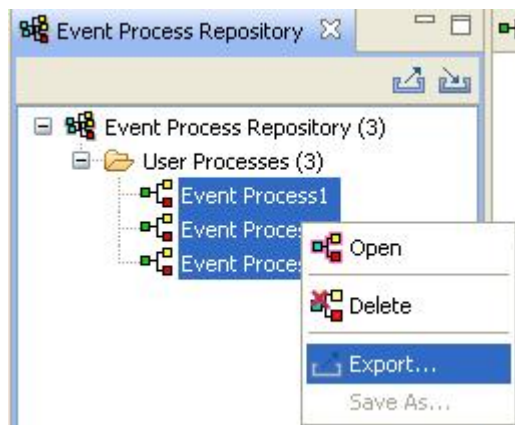


Figure 14.3.3: Exporting Event Processes from Event Process Repository

5. In Project Explorer view, right click on **EStudioSvnProjects** and select Refresh option. The Event Processes exported are now visible.

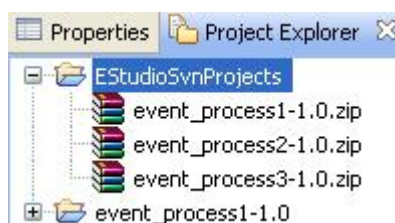


Figure 14.3.4: Project Explorer view displaying exported Event Processes

14.4 Adding the Project to Repository

1. Version Control options are available in the context menu of the EStudioSvnProjects. Select **Team -> Share Project** option to add this project into the repository.

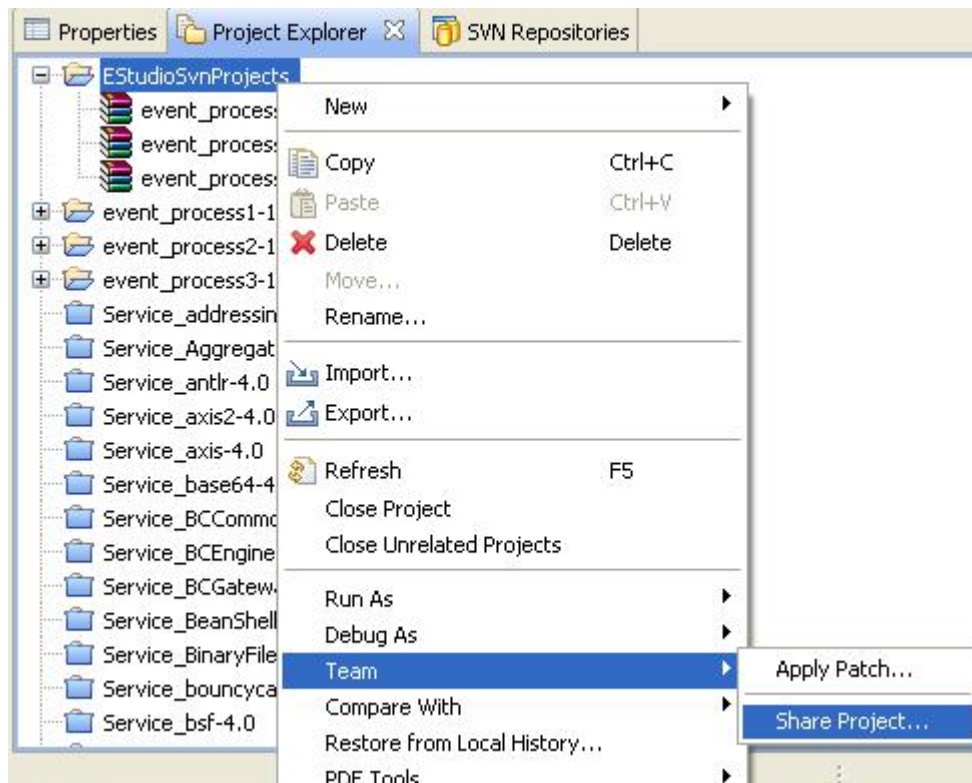


Figure 14.4.1: Adding the project

2. A dialog lets you choose an existing repository location, or a new repository can be created.

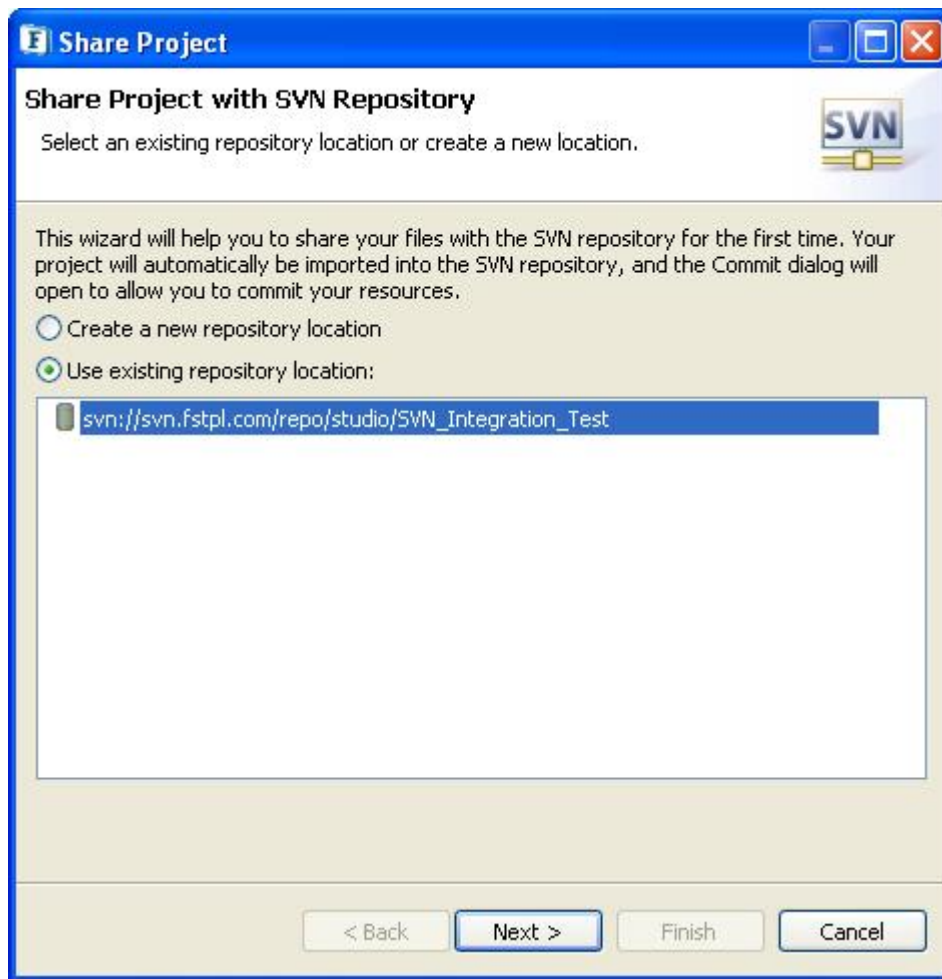


Figure 14.4.2: Selecting the repository

3. Finish the wizard. A dialog will be prompted to switch to **Team Synchronizing** perspective. Select Yes.
4. A view named **Synchronize** is shown in the perspective. Right click on the **EStudioSvnProjects** and select Commit option.

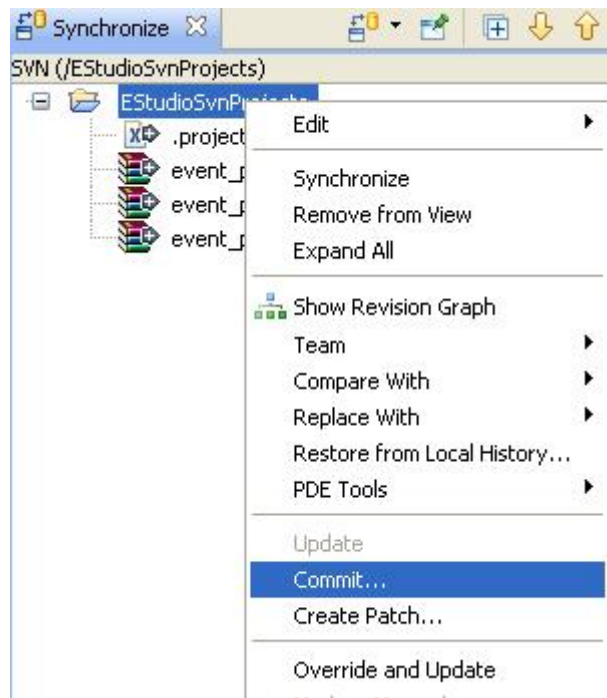


Figure 14.4.3: Adding Event Processes to repository

5. Commit dialog will be shown. Enter a comment and select Ok.
6. A new directory with the name **EStudioSvnProjects** will be created in the repository.

14.5 Updating the project into the Repository.

1. After making changes in the Event Processes, export them to **EStudioSvnProjects** as mentioned in section 14.3.
2. Overwrite the already existing Event Processes with the latest ones.
3. Refresh **EStudioSvnProjects** from the Project Explorer view and commit into the repository.

14.6 Updating an Event Process with older version from Repository

The following steps explain the process to update an Event Process with an older version from the repository.

1. Right click on the Event Process project to be updated and select **Replace With -> Revision** option as shown in Fig 14.6.1.

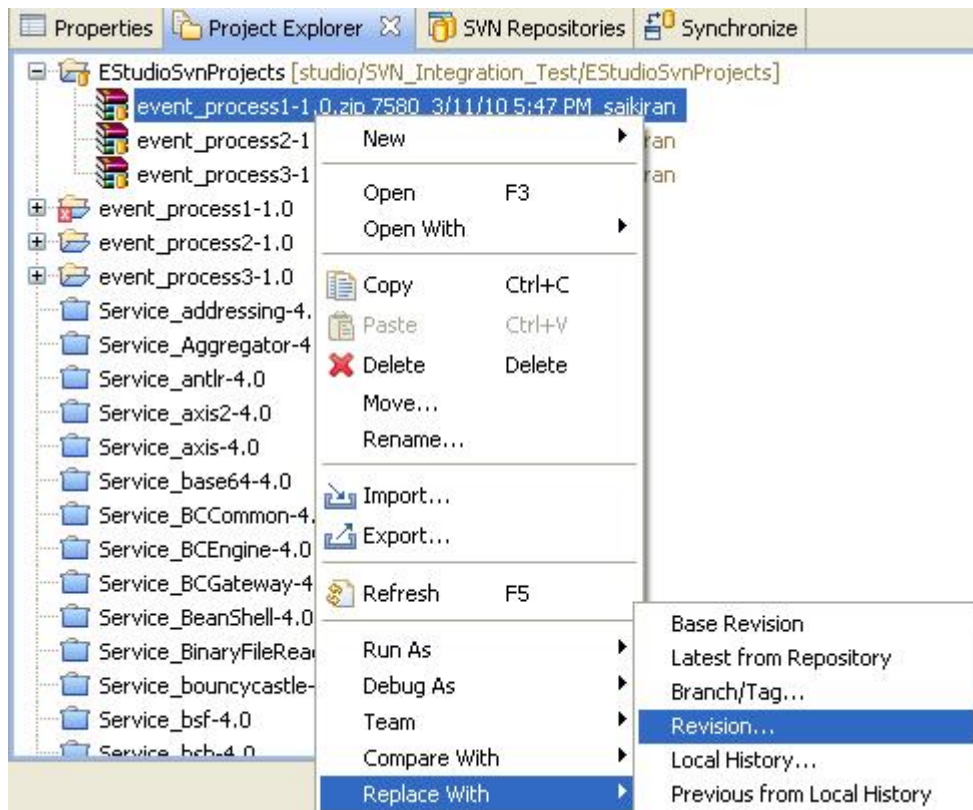


Figure 14.6.1: Updating the project

2. All the existing revisions are shown. Select the required revision and from the context menu choose **Get Revision** option.

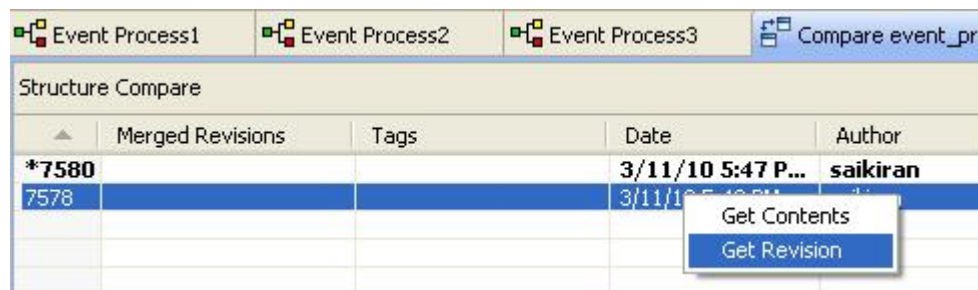


Figure 14.6.2: Getting the required revision.

3. The Event Process will be updated with the revision selected. This Event Process has to be imported into eStudio from **EStudioSvnProjects** for the changes to take effect.
4. To import, right click on **Event Process Repository** node and select **Import Event Process** option. Specify the location of Event Process in **EStudioSvnProjects** and select Ok.
5. Overwrite the already existing Event Process.
6. The Event Process will be updated in eStudio with the selected revision.

Similarly new Event Processes added externally can be updated and added into eStudio. For this, **EStudioSvnProjects** has to be updated with the latest contents from the repository and the new Event Processes have to be imported into eStudio.