Fiorano

Enabling change at the speed of thought

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

Email: info_asiapac@fiorano.com

Contents

Chapter 1: Introduction to eStudio

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

1. Offline Event Process Development

With eStudio, you can develop Event Processes without connecting to any server. A server connection is required only when you are deploying an Event Process.

2. EPLCM (Event Process Life Cycle Management)

EPLCM allows an user to move an Event Process in different labeled environments that is, Testing, Staging, QA, and Production at the click of a button. Pre-created profiles for each environment are automatically picked up by the Server at the deployment time.

3. Sub Flows

A powerful new sub-flow concept has been added.

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, which makes them more readable and reduces the memory footprint of eStudio.

6. Service Descriptor Editor

An editor to edit the `ServiceDescriptor.xml` file. It makes the edit easy as compared a Text/Xml editor.

7. Quicker Custom Property Sheet (CPS) launch

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

8. Dynamic Validations while Editing and Creating Services and Applications

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

9. Different Perspective for eeMapper and eStudio within Same Workbench

The eStudio incorporates different perspectives for the creation of mappings and for event-process development/editing; user no longer needs to open the eMapper in a separate window and process.

10. UI crafted for Rich User Experience

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

11. Support for Version Control Systems

Users can now store applications into any version control system (SVN, CVS, or VSS) using Fiorano eStudio.

12. New Mapping Tool: eeMapper

The eStudio incorporates a brand new mapping tool developed ground-up in Eclipse. This new version fixes many bugs over past versions and has several other enhancements.

13. 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 and use existing ones and can also customize the eStudio the way they want. For instance, a user can add a version control plug-in.

1.1 Getting Started with eStudio

To start eStudio:

1. Navigate to **$FIORANO_HOME/eStudio/eclipse**, open the eclipse.exe file. This opens the **Workspace Launcher** dialog box. Here you to select your workspace directory. A workspace is the directory where your work will be stored.
2. After you chose the workspace location, a **Welcome** page is displayed.
3. In Case, workbench is not loading properly and no Welcome page is displayed. Install XULRunner on your machine. Follow the guide lines from here https://developer.mozilla.org/en/Getting_started_with_XULRunner to install and add following

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

Click on **Workbench** icon as shown in the figure 1.1. A Workbench window offers one or more perspectives. A perspective defines the initial set and layout of views in the Workbench window. Within the window, each perspective shares the same set of 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 combines views that you would commonly use while editing Java source files, while the Debug perspective contains the views that you would use while debugging Java programs. As you work in the Workbench, you will probably switch perspectives frequently.

Figure 1.1: Welcome screen showing Workbench icon

1.2 Fiorano Perspective

Fiorano eStudio has two perspectives:

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

These perspectives are for online and offline application developments, which are more explained in following sections.

1.2.1 Offline Event Process Development Perspective

To open Offline 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 as shown in the figure 1.2.1. The Open Perspective dialog box appears.

Figure 1.2.1: Open Perspective Short cut menu

2. Select the **Offline Event Process Development** to open the offline perspective. Click the **OK** button. The **Offline Event Process Development** panel appears.

An icon is added to the shortcut bar, allowing you to quickly switch back to that perspective from other perspectives in the same window as shown in figure 1.2.2.

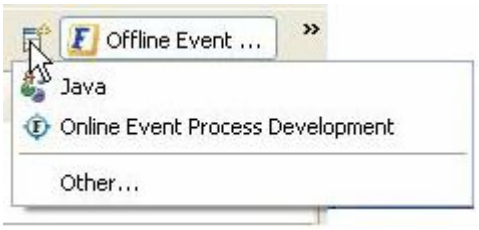


Figure 1.2.2: Shortcut icon to switch perspectives

By default, a perspective will open in the same window. We recommend you to open it in a new window by changing the settings in the Window>General > Perspectives preference page.

Note: Offline perspective contains all the views and editors required for the offline application development. During offline application development, there will not be any interaction with the server. Created Event Process can be exported to Server and any Event Process can be imported from the server.

1.2.2 Online Event Process Development Perspective

To open 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 Application Development to open online perspective. Click the **OK** button. The **Online Event Process Development** panel appears.

Note: Online perspective contains all the views and editors required for the online application development. During online application development, application development takes place after logging in to the server.

1.3 Event Process Repository

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

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

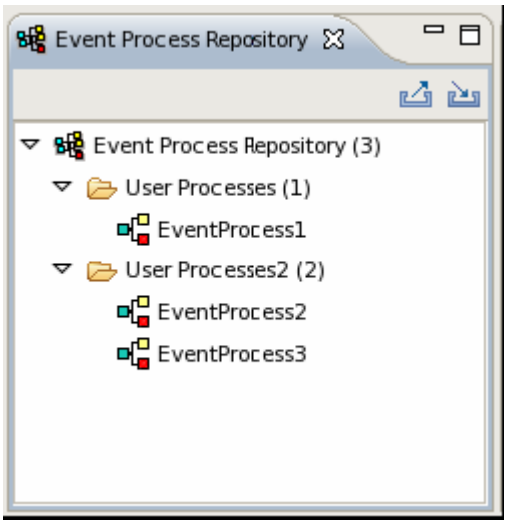


Figure 1.2: Event Process Repository

1.4 Fiorano Orchestration

Any Fiorano perspective in the Workbench is comprised of an editor area Fiorano orchestrator.

For example, when you open an Event Process in the Server Explorer view or Event Process Repository view, it will show the design of the application in Fiorano Orchestrator. You can also open event process from EventProcess.xml file in other editors such as, Text Editor, XML Editor, and so on from Project Explorer view.

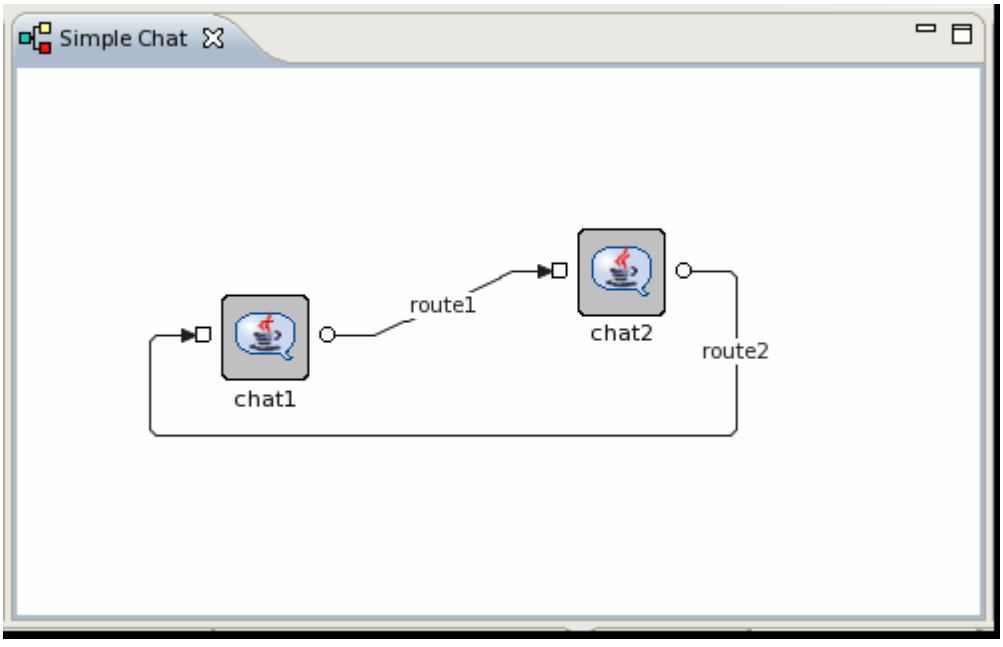


Figure 1.3: Simple Chat dialog box

1.5 Service Palette

The Service Palette shows the services that are present in the eStudio repository. The Service Palette contains all the Fiorano services such as Bridges, Collaboration, DB, Error, File, and so on as shown in figure 1.4.

Figure 1.4: Fiorano Service Palette

1.6 Properties

The Properties view displays all the property names and values for a selected item such as, a service instance. To add the Properties view to the current perspective, click Window > Show View > Other > General > Properties.

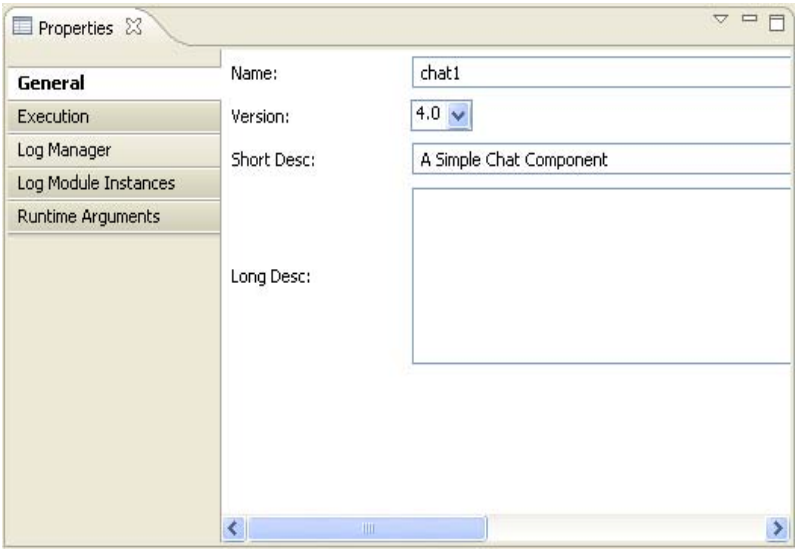


Figure 1.5: Properties pane

1.7 Problems

As you work with Fiorano Processes in the workbench, various builders may automatically log errors or warnings in the Problems pane. For example, when you save an Event Process file that contains errors, those errors will be logged in the Problems pane.

To add the Problems pane to the current perspective, click Window > Show View > Other > General > Problems.

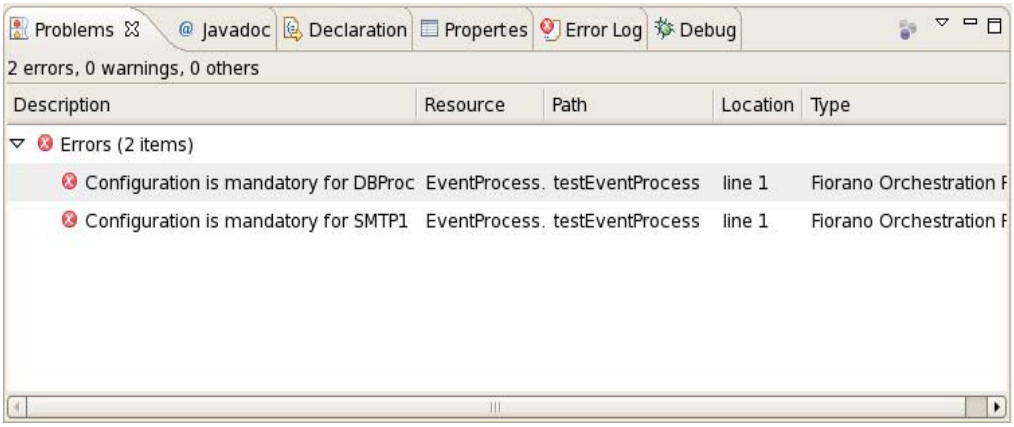


Figure 1.6: Problems pane

By default the problems are grouped by severity level. You can also group them by type or grouping can be removed at anytime. Certain services will add their own grouping. For instance, the Java Development Tools (JDT) support adds a Java Problem Type group. The grouping can be selected using the Group By menu.

You can configure the contents of the Problems to view only 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. You can also 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). The two most popular filters (All Errors and Warnings on Selection) are provided by default.

1.8 Error Log

The Error Log view captures all the warnings and errors logged by plug-ins. The underlying log file is a `.log` file stored in the `.metadata` subdirectory of the workspace. The Error Log view is available under Window > Show View > Error Log.

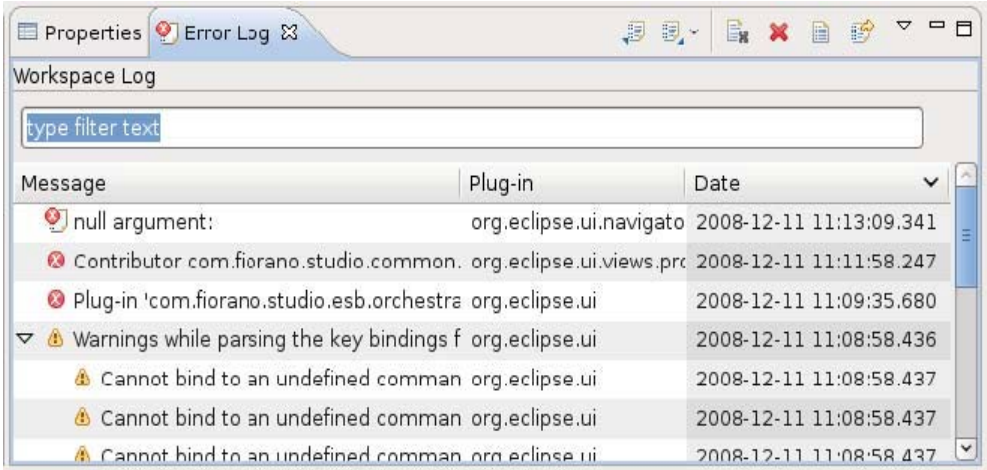


Figure 1.7: Error Log pane

1.9 Fiorano Debugger

The Fiorano Debugger view is one of the Online Application Development Perspective views. This view shows the list of routes on which debugger is enabled and messages trapped in each route. This gives user to take action on debug message.

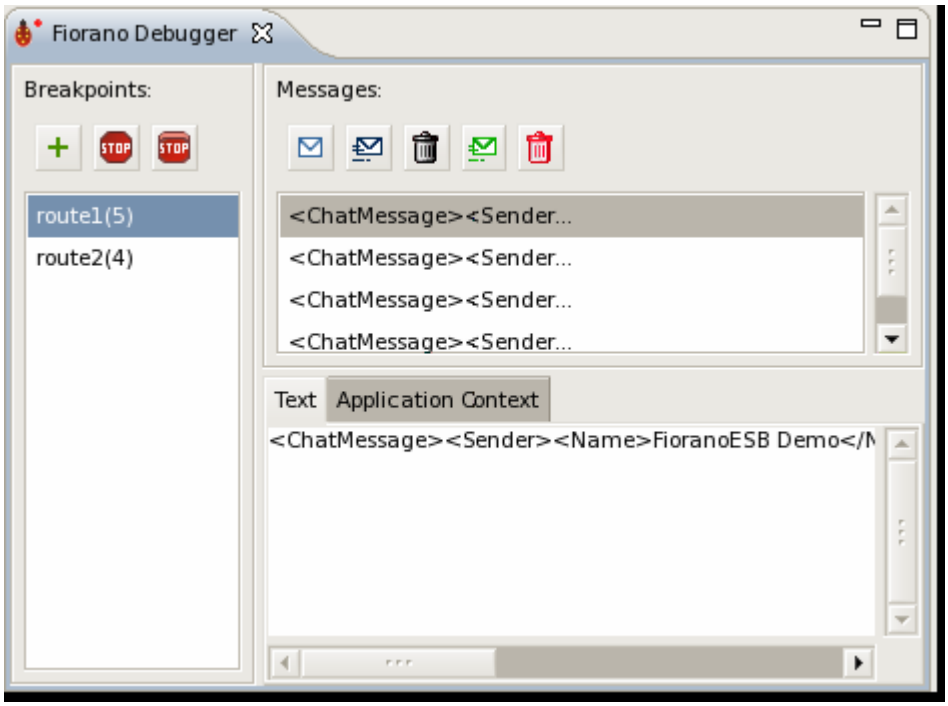


Figure 1.8: Fiorano Debugger pane

1.10 Service Descriptor Editor

You can edit a service through the Service Descriptor editor, to open a service in the service descriptor editor, right-click on the desired service in service palette and click the **edit** option from the pop-up menu.

The properties of service are divided into three categories:

- Overview
- Execution
- Deployment

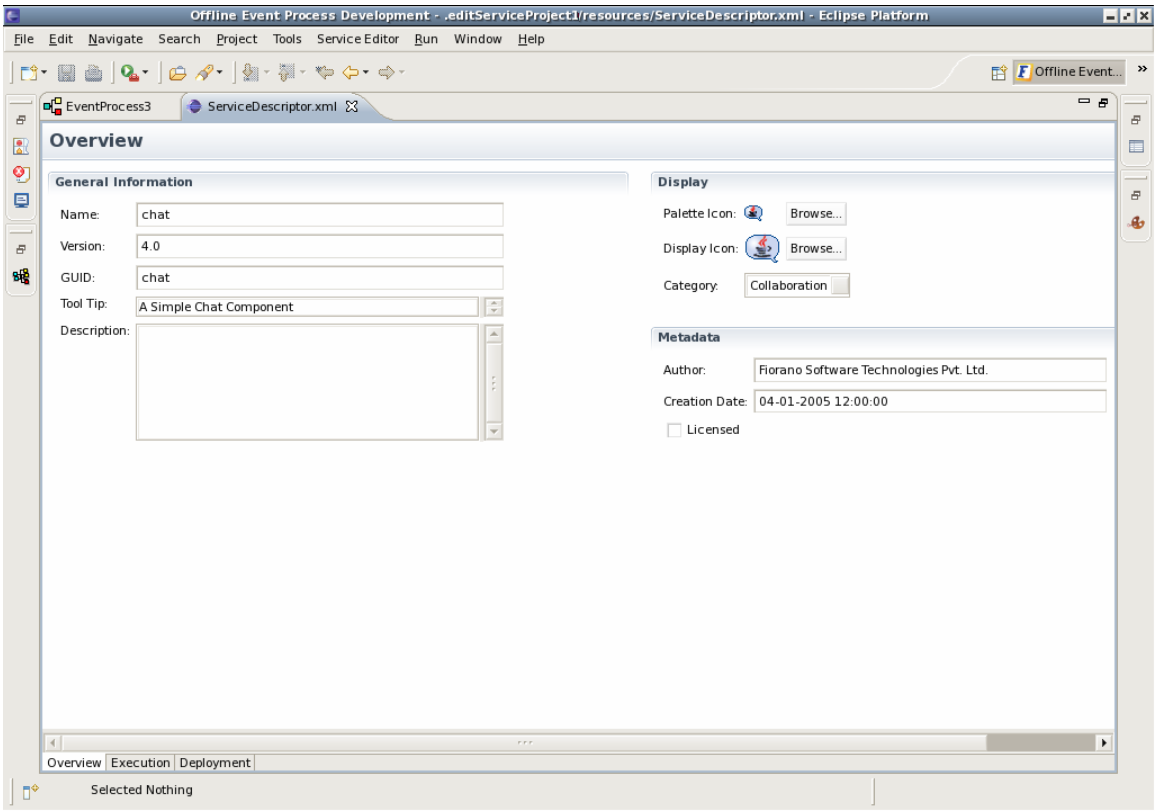


Figure 1.9: Fiorano Service Descriptor Editor

1.11 Service Repository

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. The Fiorano eStudio contains its own service repository to enable offline composition of event processes and are not tied to any particular Fiorano ESB Server.

Services which are available only in 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 Service Repository.

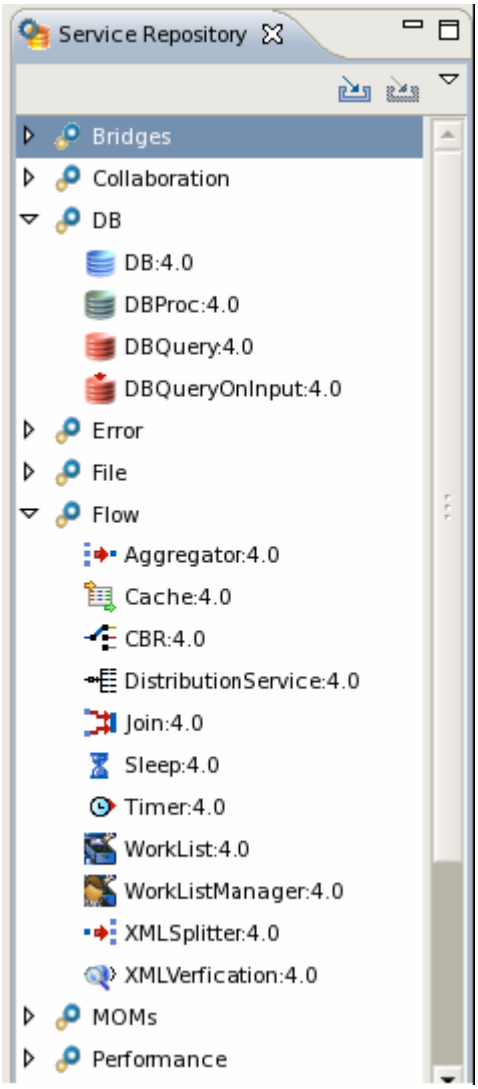


Figure 1.10: Fiorano Service Repository pane

1.12 eMapper

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

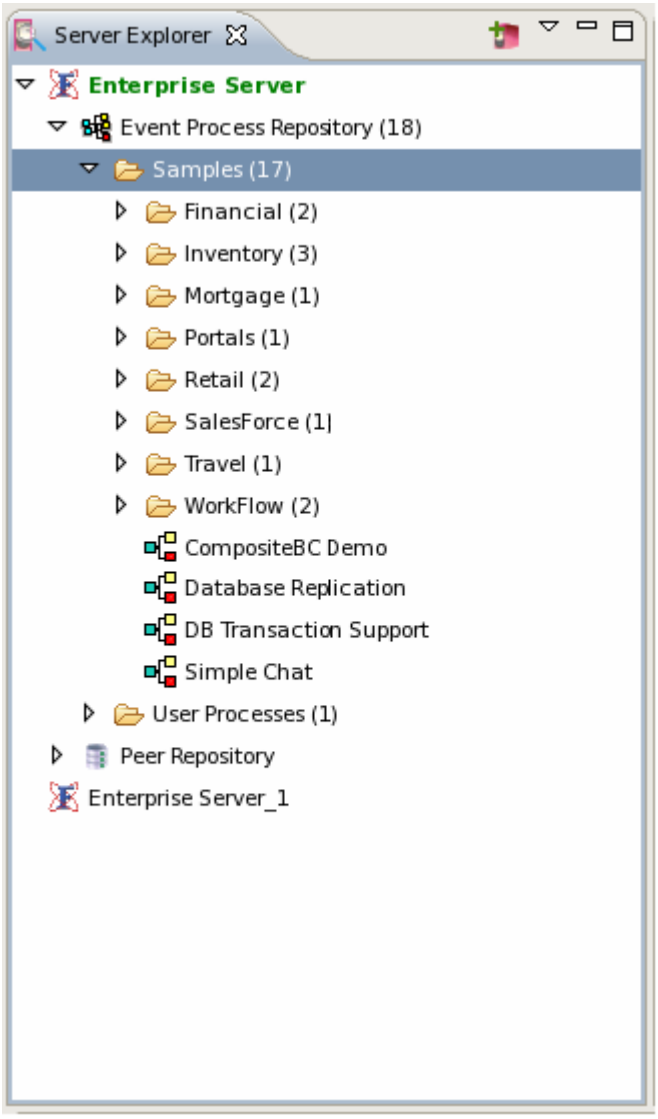
1. Click the **Open Perspective** button from the shortcut bar on the left-hand side of the Workbench window.
2. Select **Other...** from the drop-down menu.
3. Select the **eMapper Perspective** to open Fiorano perspective. Click **OK** button. The eMapper perspective containing Project Explorer and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

1.13 Server Explorer

The Server Explorer view is one of the views of Online Application Development Perspective. Which is available under Window > Show View > Fiorano > Server Explorer.

The Server Explorer view shows the Enterprise servers, which contains event process repository and peer repository.

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



Chapter 2: Event Processes

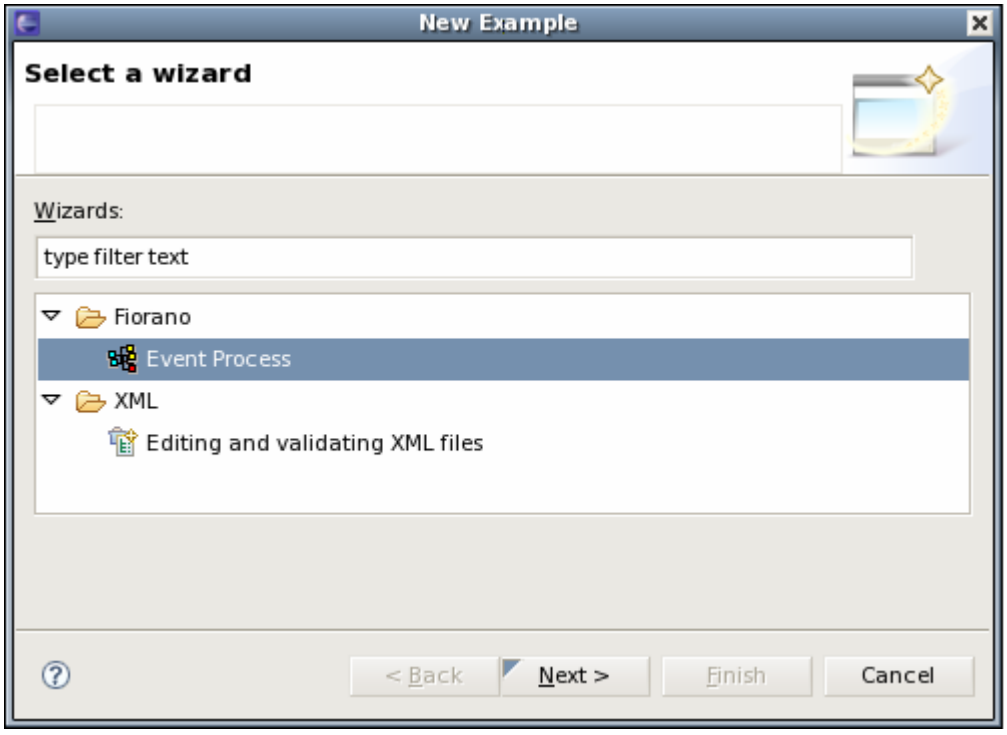
Event Processes are composite applications created as event-driven assemblies of service components. They represent the orchestration of data flow across customized service-components distributed across the ESB network. Event processes in Fiorano are designed to connect disparate applications in a heterogeneous 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 and profile information and the layout. The event-process meta-data contains all required information in XML format and is stored in the service repository managed by the Fiorano Enterprise Server.

2.1 Opening Sample Event Process

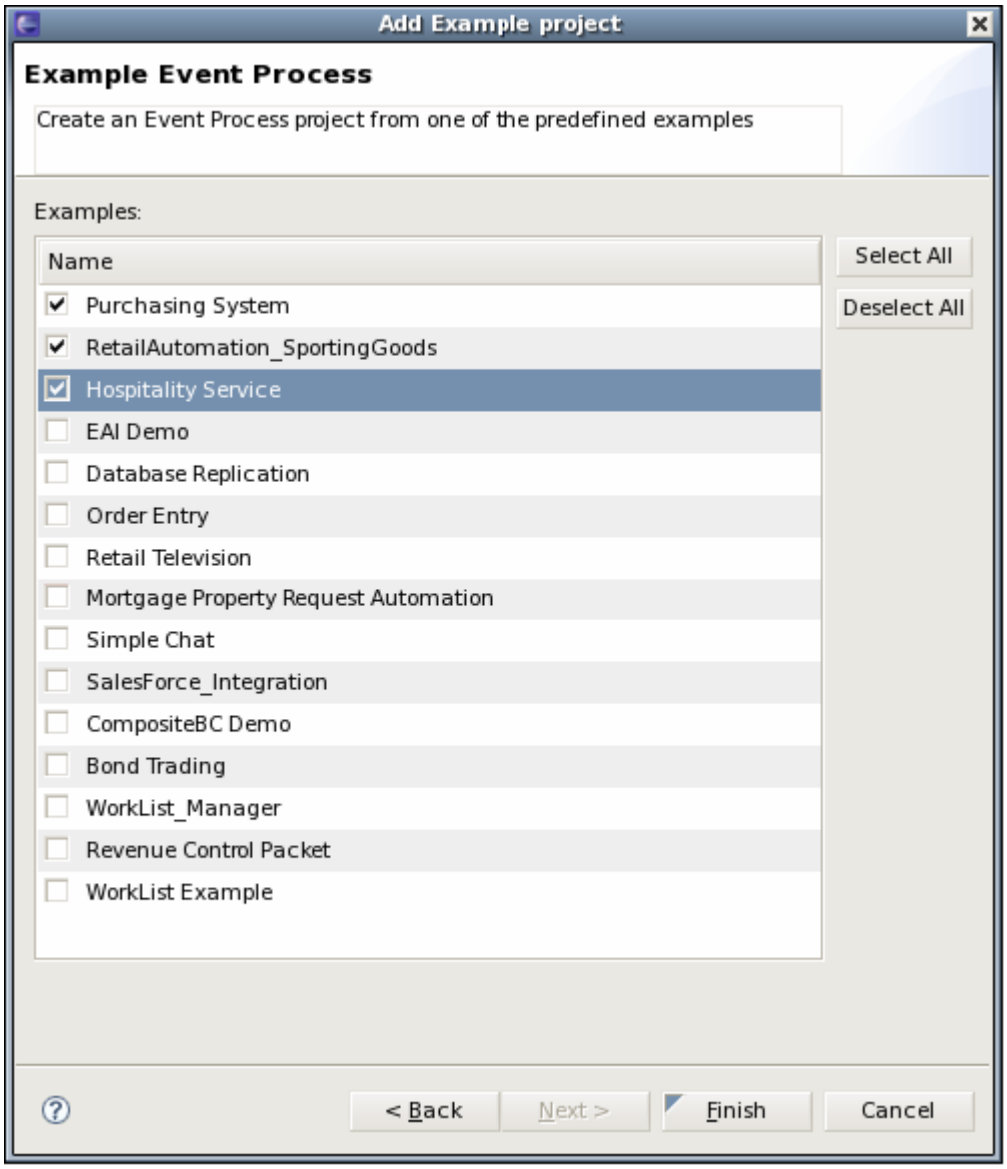
To open a pre-configured sample event process, perform the following steps:

1. Navigate to File -> New -> Example and select Fiorano -> Event Process and click **Next**. A list of pre-configured Event Processes is displayed.



2. Select the Event Process(s) you want to open by selecting the check box against each entry and then click **Finish**. Each selected Event Process appears under Event Process repository node of Event Process Repository view irrespective of the perspective in which they are added.

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



Note: The samples which are added to the repository already will not be visible in the add example wizard.

2.2 Import and Export Event Processes

You can import and export an Event Processes in eStudio. The following sections describe the procedure for exporting and importing an event process.

2.2.1 Exporting an Event Process

Event Process can be exported to local disk in both online event process development perspective and offline event process development perspective. Where as Event process can be exported to server only in offline event process development perspective.

To export an Event Process onto a local disk (in both, that is, online event process development perspective and offline event process development perspective), perform the following steps:

1. Right-click the Event Process to be exported from the Event Process Repository view and select Export from the menu (Figure 2.2.1). The Export dialog box appears.

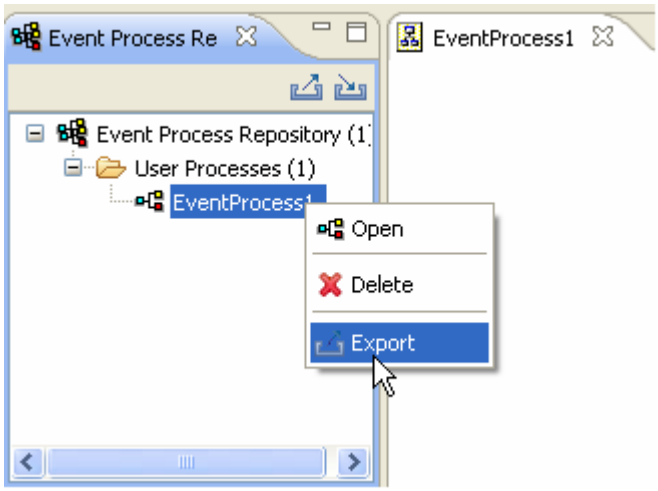


Figure 2.2.1: Export Event Process to local disk

2. Specify the file name and location to save and click OK. Event Process project will be saved as a zip file.

To export an Event Process to Server (in offline event process development perspective), perform the following steps:

1. Click on **Export Application** icon present on Event Process Repository view tool bar as shown in the figure 2.2.2. The **Select Application to be exported** dialog box appears listing all the event processes and servers specified in server preferences.

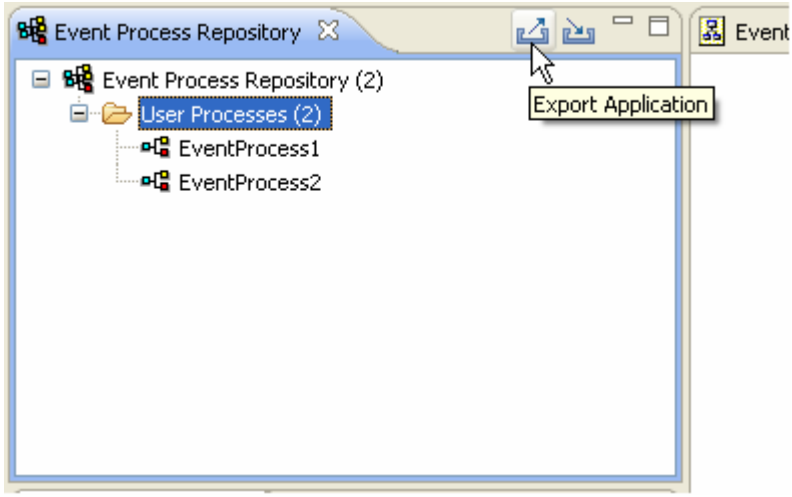


Figure 2.2.2: Export Application

2. Select the event process to be exported and server on to which Event Process has to be exported and click **OK**.

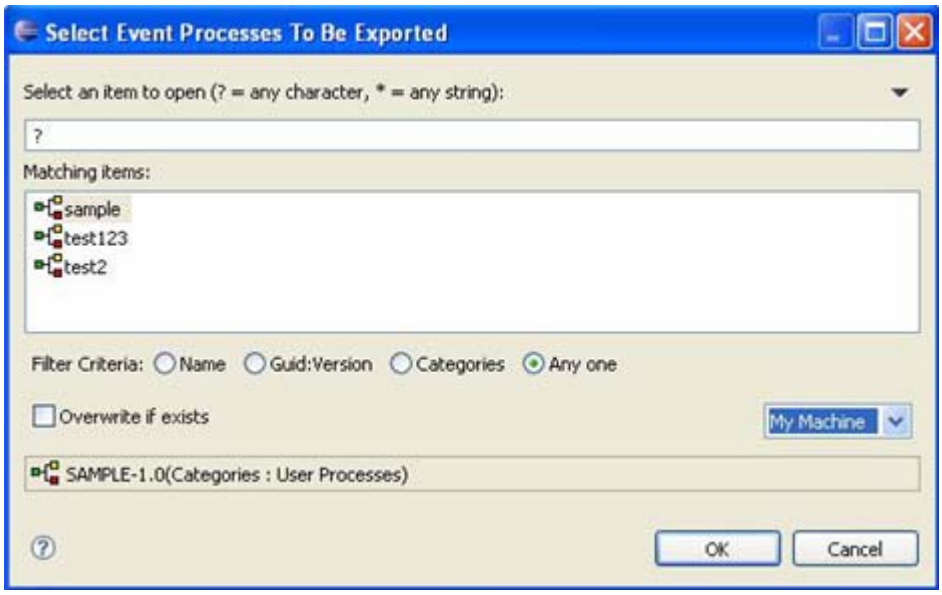


Figure 2.2.3: Select Application to export

2.2.2 Importing an Event Process

Event Process can be imported from local disk in both online event process development perspective and offline event process development perspective. Where as, an Event process can be imported from server only in an Offline Event Process Development perspective.

To import an Event Process from local disk (in both, that is, online event process development perspective and offline event process development perspective), perform the following steps:

1. Right-click the **Event Process Repository** node in the **Server Explorer** view and select **Import Event Process** from the menu list as shown in figure 2.2.4.

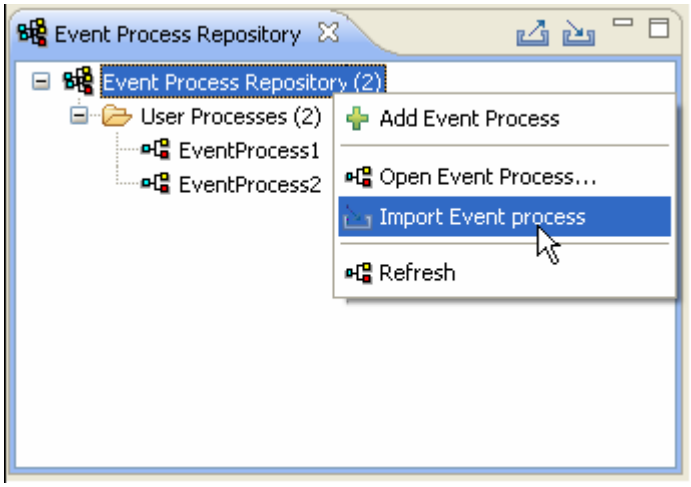


Figure 2.2.4: Import Event Process

2. Specify the location of event process zip file and click **OK**. Event Process project will be imported to event processes repository.

To import an Event Process from Server (in offline event process development perspective), perform the following steps:

1. Click on **Import Application** from Server icon present on **Event Process Repository** view tool bar as shown in the figure 2.2.5.

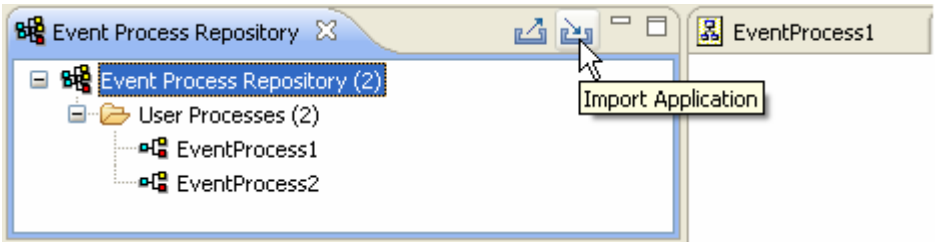


Figure 2.2.5 Import Event Process from Server

The Select a Sever dialog appears listing all servers specified in server preferences.

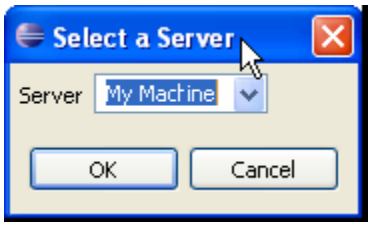


Figure 2.2.6: Select Server

2. Select server from which Event Process has to be imported and click **OK**. The Select Application to be imported dialog box appears as shown in Figure 2.2.7.
3. Select an Event Process to be imported and click **OK**. Event Process project will be imported to event processes repository.

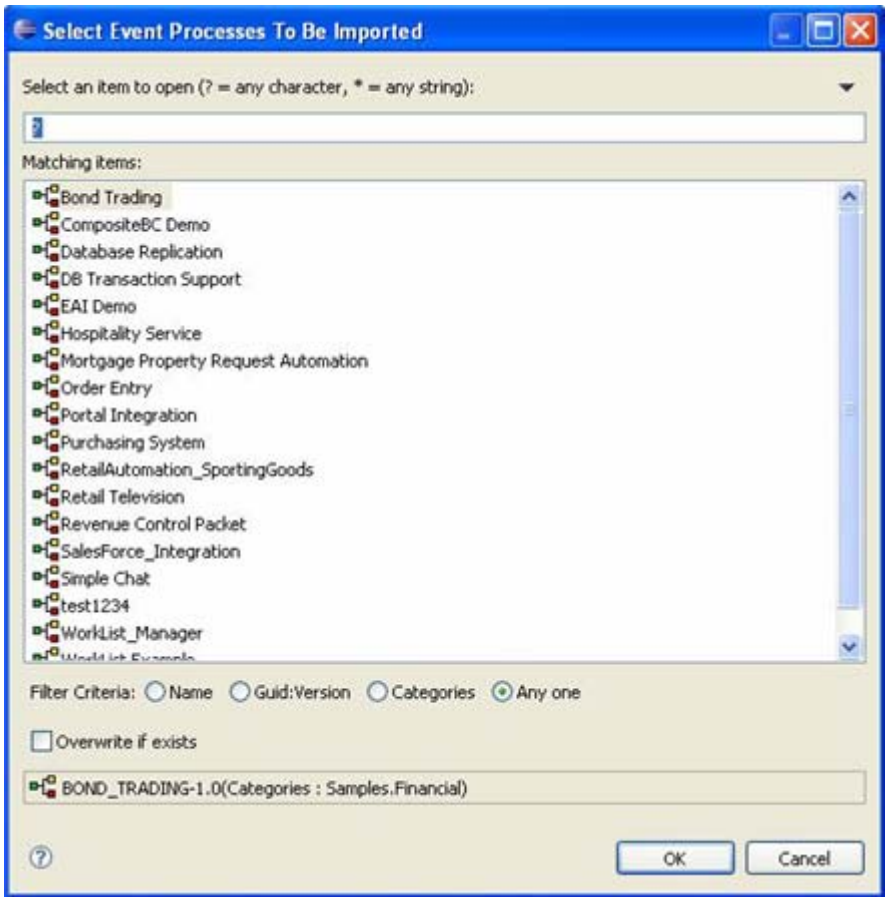


Figure 2.2.7: List of Event Process in server

2.2.3 Importing nStudio Event Processes

Event Processes which are exported from nStudio can be imported to eStudio using the Import Event Process (nStudio) option present on the right click menu of Event Process Repository node in any of the event process development perspectives.

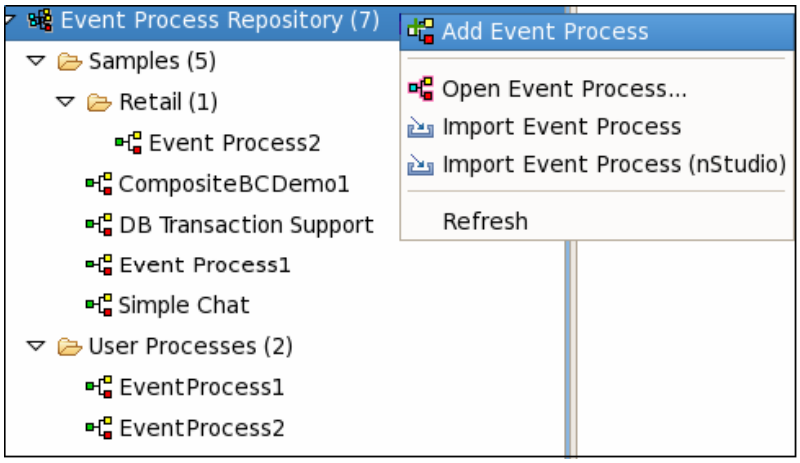


Figure 2.2.8: Add Event Process

Selecting the import option opens an Import Wizard as shown in Figure 2.2.9.

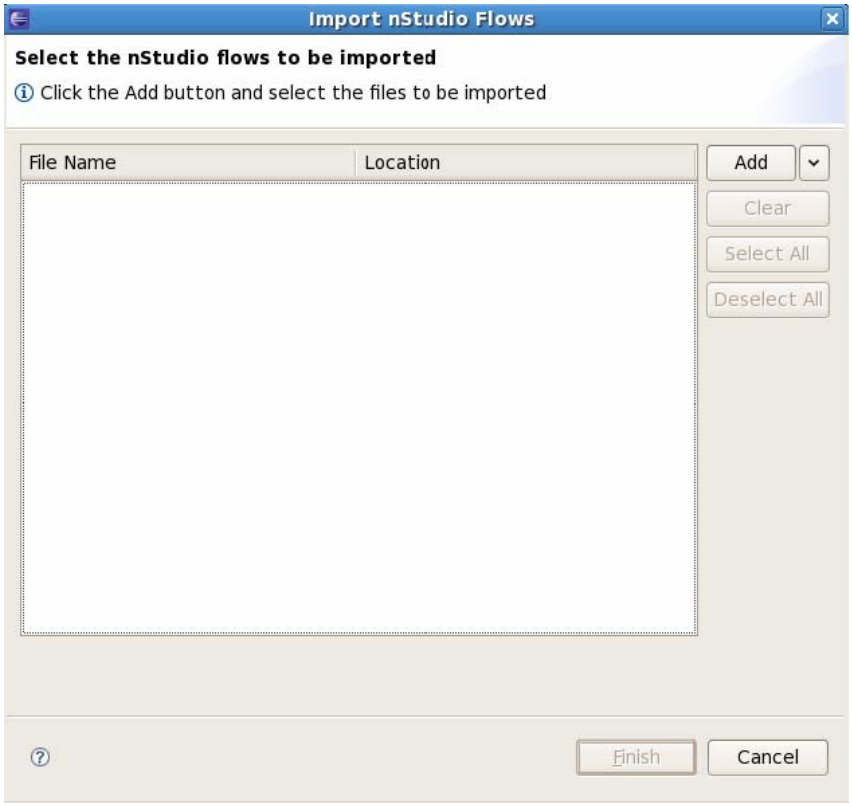


Figure 2.2.9: Import nStudio Flows

The files to be imported can be added to the table containing the files list by using the add button. When add button is pressed a dialog opens up requesting for the folder containing the nStudio flows. Here the folder containing the files can be selected. All the supported flows present in the folder and all of its sub-folders will be added to the table. Flows can also be added to the table by using the **add from files** option present in the drop-down button located on the right side of the add button.

The state of the wizard after adding the flows is shown in Figure 2.2.10:

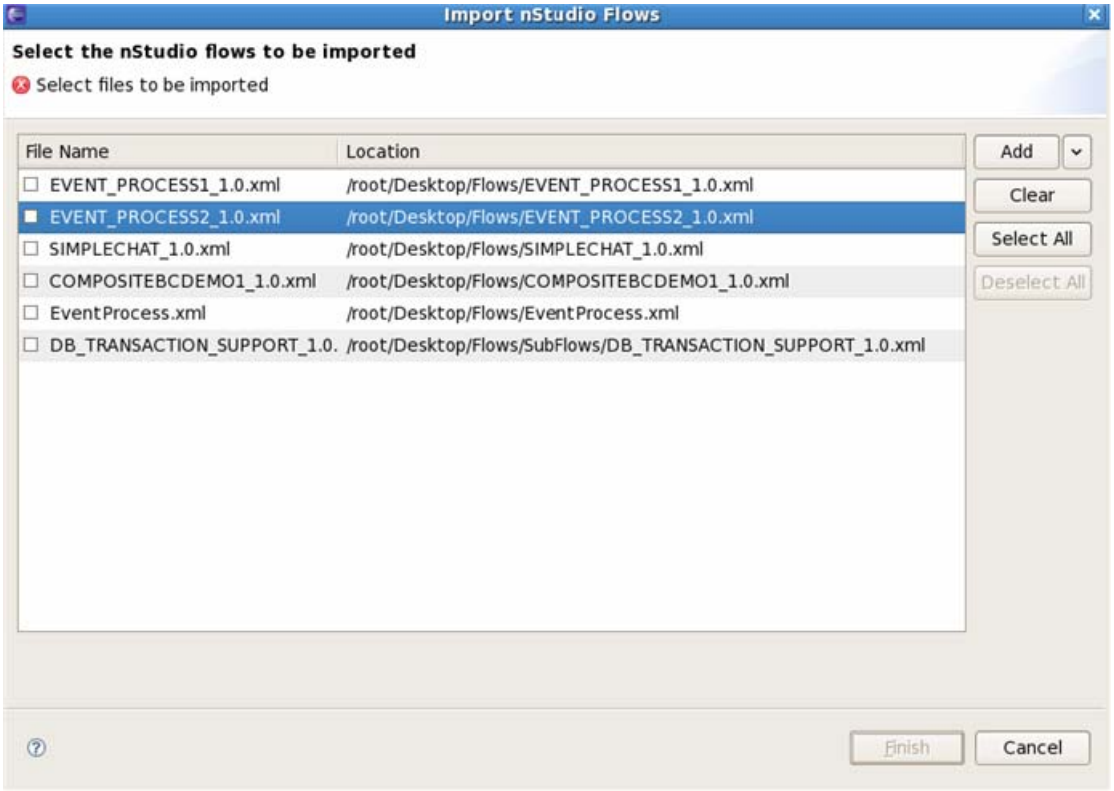


Figure 2.2.10: Select nStudio flows to import

After adding the selected files to the table the flows which have to be imported can be selected using the check box against each entry present in the table. Press the finish button to import the selected flows to the current repository which can be either offline or online repository depending on the repository node in which the import option is selected. If any of the selected flows already exist in the repository user will be prompted with a dialog having options to overwrite/ignore/rename the flow. The imported flows can be viewed under the repository node to which they have been added.

2.3 Creating New Event Process

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

1. From the **File** menu, select New -> Event Process Project.
2. Specify the name, version, category of the Event Process project and click **Finish**. The specified Event Process appears under **Event Process Repository** node of Server Explorer view or Event Process Repository view depending on user working in online application development or offline application development perspectives respectively.

   To see the graphical view of an Event Process, double-click the event process node which opens **Fiorano Orchestration** editor. For information on composing an Event Process, see Chapter 3: Composing an Event Process.

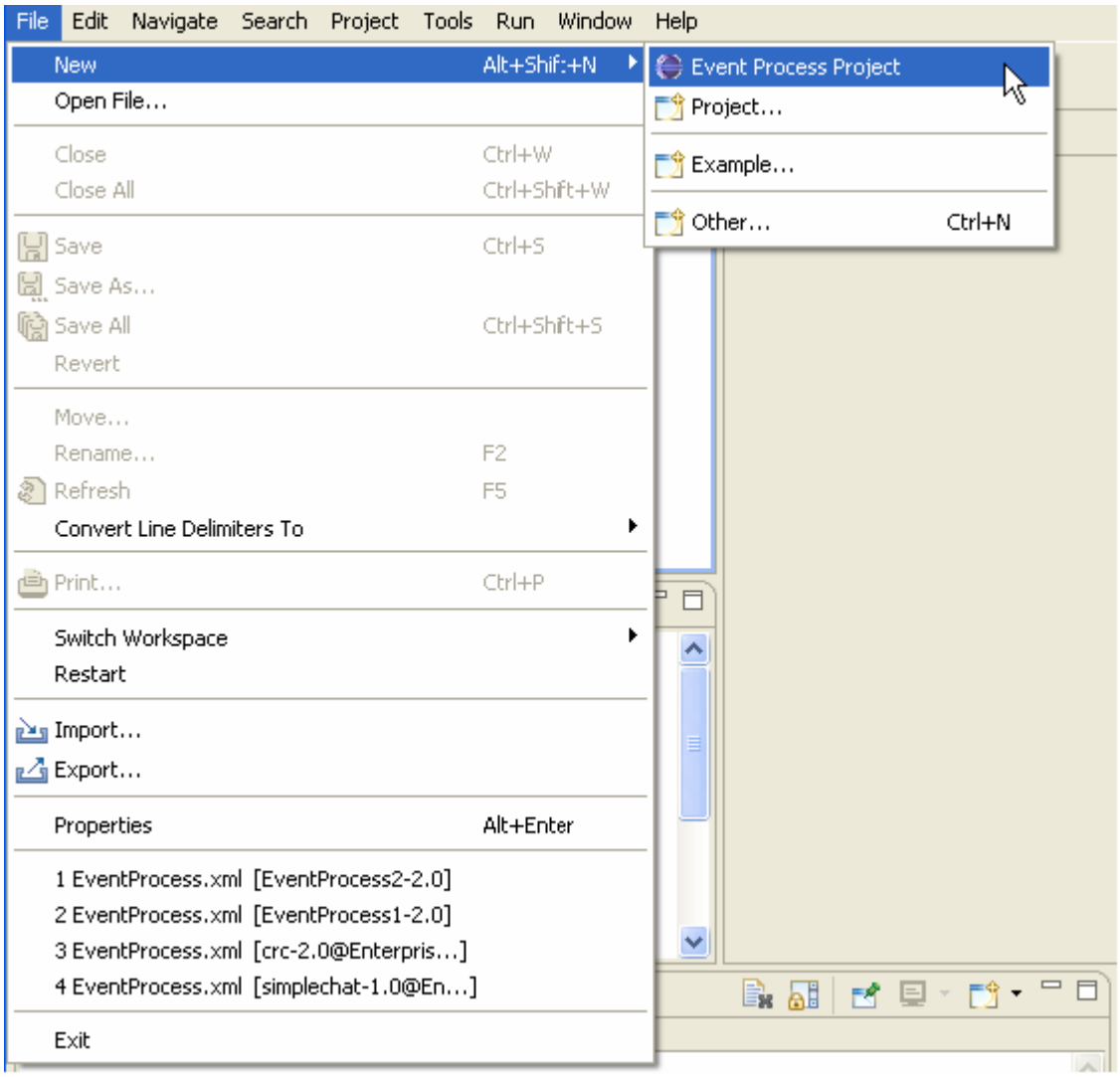


Figure 2.3.1: New Event Process Project option

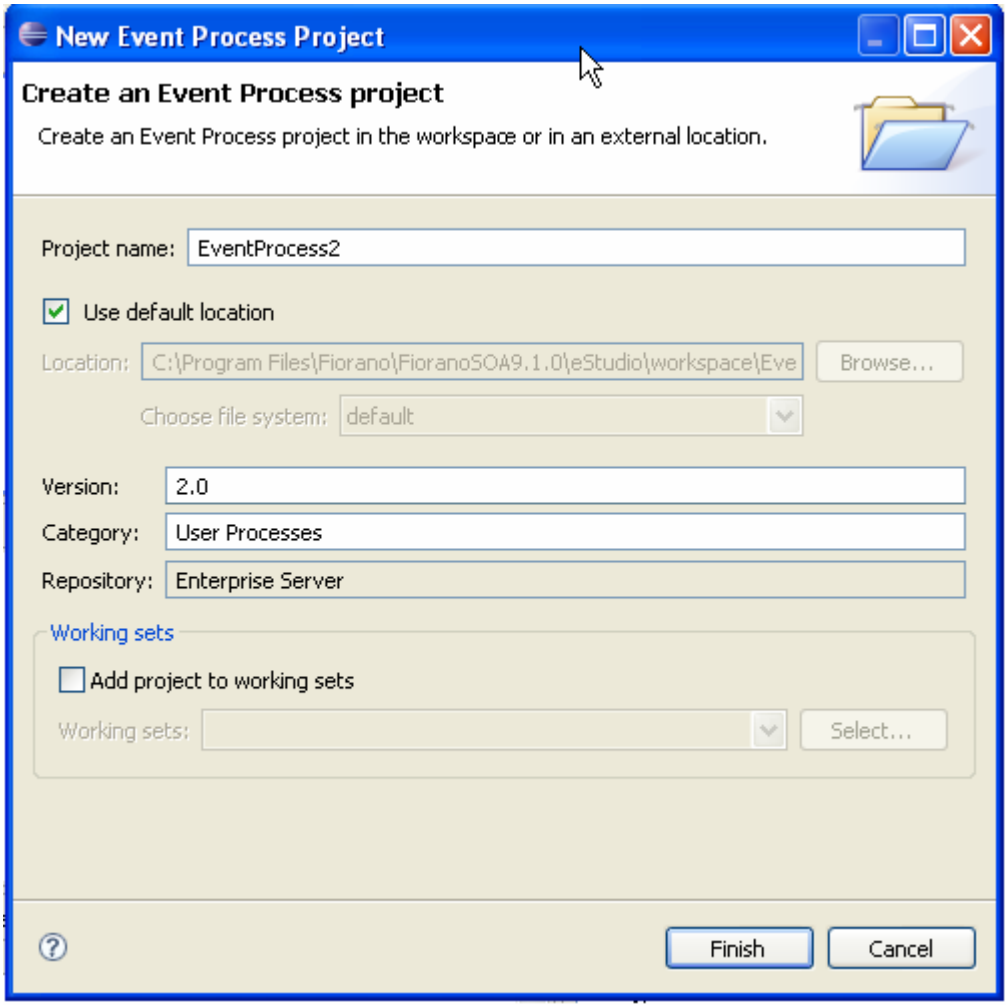


Figure 2.3.2: Creating an Event Process Project

Chapter 3: Composing Event Processes

Composition of Event Processes is based on the 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** of service components. The components are customized by configuration rather than custom code. The routes between components are drawn by visually connecting the component ports. Every component instance in the flow can be configured to be deployed on different nodes of the ESB network.

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.

3.1 Adding Components

To add components, perform the following steps:

1. Open the **Service Palette** and click the **Category** tab (say **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 Business Component. By default, the name of each instance of the Business Component is the Business Component GUID followed by the instance ID count. You can rename the Business Component instance, when required.

3.2 Connecting Routes

For data to flow between two Business Component 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 from output Event Port of the source Business Component and end at input Event Port of the target Business Component.

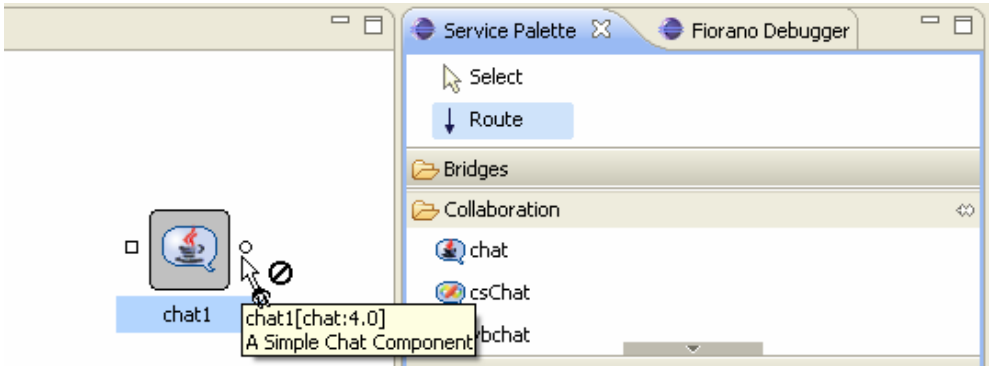


Figure 3.2.1: Component ports

Connect the Route from the output channel (OUT_PORT) of Chat1 Business Component icon to the input channel (IN_PORT) of the Chat2 Business Component icon and vice versa, as shown in the Figure 3.2.2. 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.

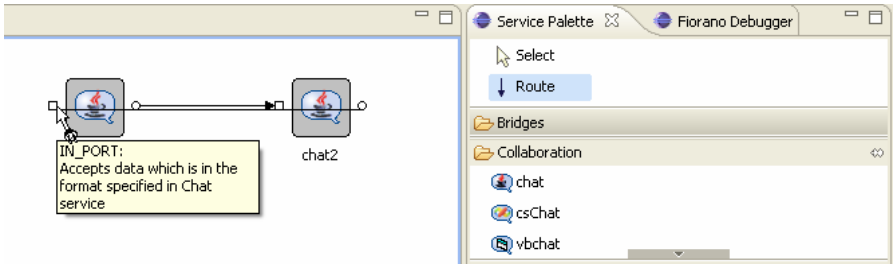


Figure 3.2.2: Connecting components through routes

3.3 Adding Remote Service Instance

The Fiorano SOA Platform allows the user to compose an Event Process with Business Component instances from other Event Processes. This enables the Fiorano Event Process Orchestrator from reusing a single Business Component instance across Event Processes to reduce Business Component instance redundancy.

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

1. Click the **Insert Element into Event Process** icon.

Figure 3.3.1: Insert Remote Service Instance option

2. Click the **Insert Remote Service Instance** option from the drop-down list. The External Business Component Selection wizard starts, as shown in Figure 3.3.2.

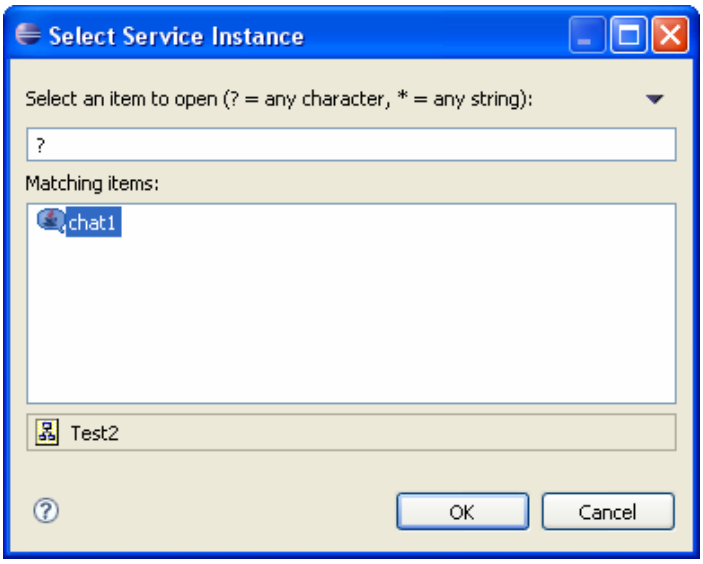


Figure 3.3.2: Select Service Instance dialog box

3. 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 3.3.3.

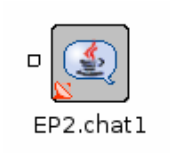


Figure 3.3.3: Remote service added

You can use the External Business Component instance similar to normal Business Component instance. You can create routes between any of the Event Process Business Component instances and the input ports of the External Business Component.

3.4 Adding External Event Process

To add an External Event Process, perform the following:

1. Click the **Insert Element into Event Process** icon and select the **Insert Event Process** option from the drop-down list. The Select Event Process dialog box appears as shown in Figure 3.4.1.

Figure 3.4.1: Insert Event Process option

2. Select the Event Process from the list and click the **OK** button.

Figure 3.4.2: Select Event Process dialog box

The Event process instance representation appears on the **Event Process** editor, as shown in Figure 3.4.3.

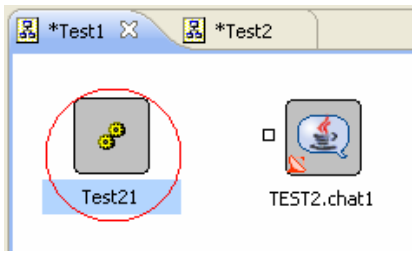


Figure 3.4.3: Event Process editor

By default, no input and output ports are shown for an Event Process instance. The user can add the required input and output ports to the Event Process instance from the Properties tab as shown in Figure 3.4.4.

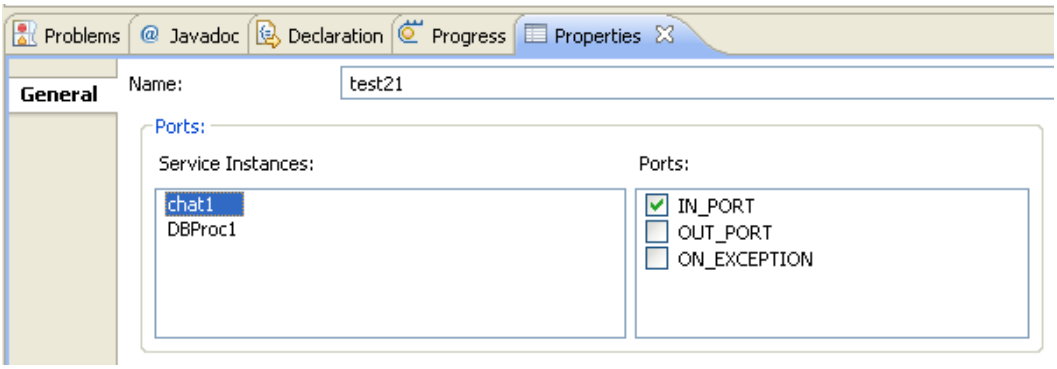


Figure 3.4.4: Properties tab

3.5 Configuring Document Tracking

eStudio provides you with a state-based workflow view that enables tracking and monitoring of documents from one state to another. Additionally, you can define the state an Event Process should be enabled or disabled for tracking.

3.5.1 Defining Document Tracking

A workflow is defined as an Event Process scope within which a large number of documents and messages flow. 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 purpose.

Each workflow instance contains information about documents that flow through a particular workflow instance. Each time a document passes through a track-able state, a state event is generated and the document is given a new Document ID by that track-able state. You can view all the information related to a document, using Fiorano Web Console.

3.5.2 Enabling State Based Tracking

To enable tracking for a state, perform the following steps:

1. Select the Event Port you want to configure. The Properties pane appears (if the Properties pane does not appear, go to Window->Show View-> Others-> and select Properties).
2. Enable the **Enable Document Tracking** option in **Doc Tracking** tab.

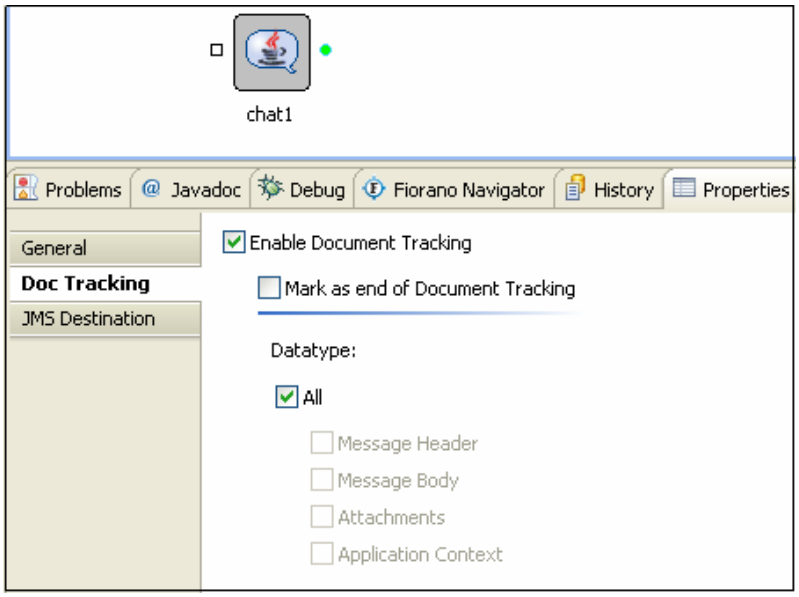


Figure 3.5.1: Doc Tracking tab

3. Ports with enabled document tracking are filled with **green** color as shown in the Figure 3.5.2. This indicates that state tracking for that particular Event Port is enabled. All messages passing in and out of that Event Port will be tracked.

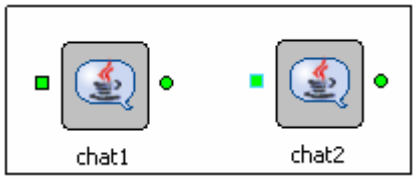


Figure 3.5.2: Ports in green color

In the Chat Event Process as shown in the Figure 3.5.2, the state tracking for the following states is enabled: IN_PORT of chat1 OUT_PORT of chat1 IN_PORT of chat2 OUT_PORT of chat2.

3.5.3 Setting the End State

To track a workflow, you need to mark an end state for the workflow. The end state, as the name implies, marks the end of the workflow.

To enable document tracking using the button provided in toolbar, perform the following steps:

1. Click the **Toggles the Document Tracking Mode**  button from the toolbar and click the ports of documents to be tracked.

Figure 3.5.3: Toggles the Document Tracking Mode button

2. Click the Event Port to be configured. In the **Doc Tracking** tab, enable the **Mark as end of Document Tracking** option. The last port which has been selected will be filled with red color, meaning end of document tracking.

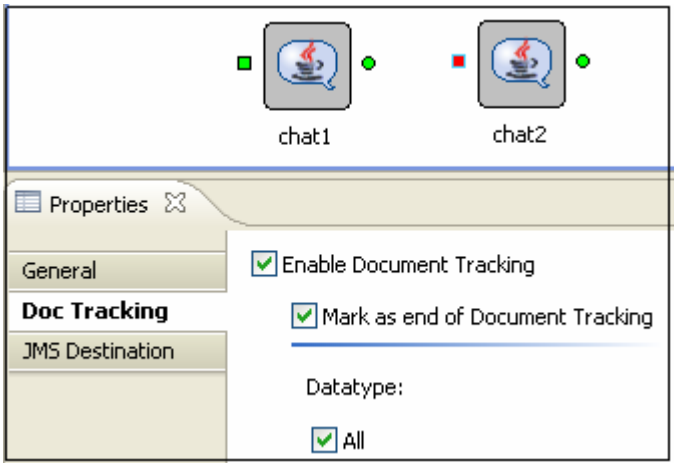


Figure 3.5.4: Mark as end of Document Tracking option

3.6 Configuring Selectors on Routes

The eStudio allows you to define Selectors for the flow of data through an event route. Take the example of an Event Process containing two instances of a Chat business component and an instance of a Display business component, as shown in Figure 3.6.1.

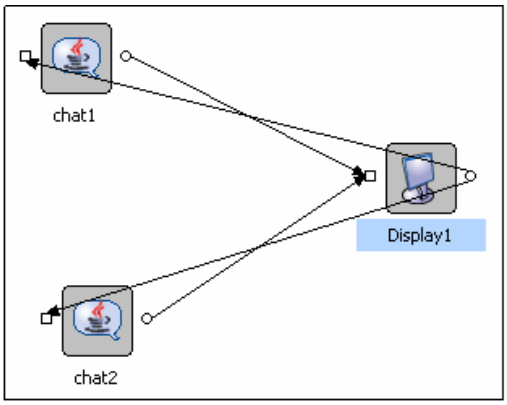


Figure 3.6.1: Out-port of chat1 and chat 2 to display in-port, out-port of display to in-port of Chat1 and chat2

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

Now, let us define conditional data flow from the Display business component instance. Assume that Display1 sends only those messages on Route1 which are meant for Chat2. In other words, it forwards only those messages on this Route, which have originated from Chat2. Similar conditions should also apply to Chat1. It should also receive only those messages that it sends to Display1.

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

1. Select **Route 1**, the properties of this route is displayed in the **Properties** tab.
2. Click **Selector** option and choose the option chat2 from **Sender** properties as shown in Figure 3.6.2.

This ensures that data intended for only Chat2 will travel through this route1. Similarly, set this value to Chat1 for Route2. This ascertains conditional flow of data.

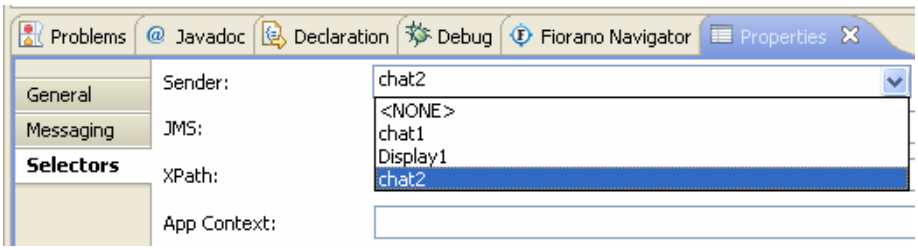


Figure 3.6.2: Selectors option

3.7 Configuring Application Context

Defining Application Context for an application:

1. In the **Event Process** project, click the **Orchestration Editor** and open the **Properties** view.

2. Click the **ApplicationContext** tab from the **Properties** tab.
3. Enable the **Application Context** option.
4. Provide DTD content in the **DTD** text section.
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.

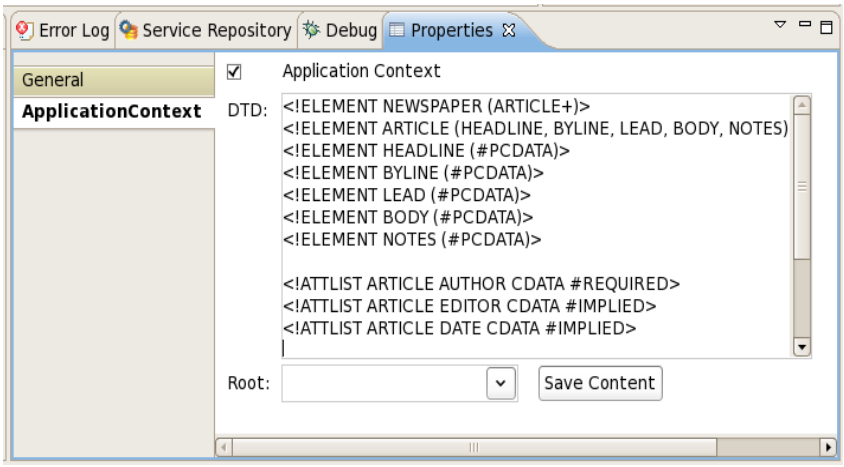


Figure 3.7.1: Apllication Context option

3.8 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 and/or IP addresses of the grid-nodes (Fiorano Peers) on which the components are to be deployed.

To do connectivity and resource check:

- Select the Event Process. Click the **Check Resource Connectivity** button from the tool bar as shown in Figure 3.8.1.

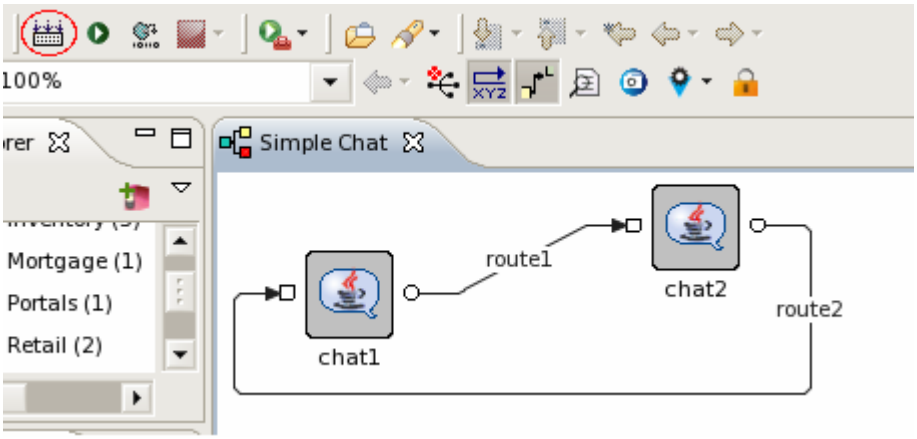


Figure 3.8.1: Check Resources and Connectivity

3.9 Running Event Process

- Select the Event Process. Click the **Launch Event Process** button as shown in Figure 3.9.1.

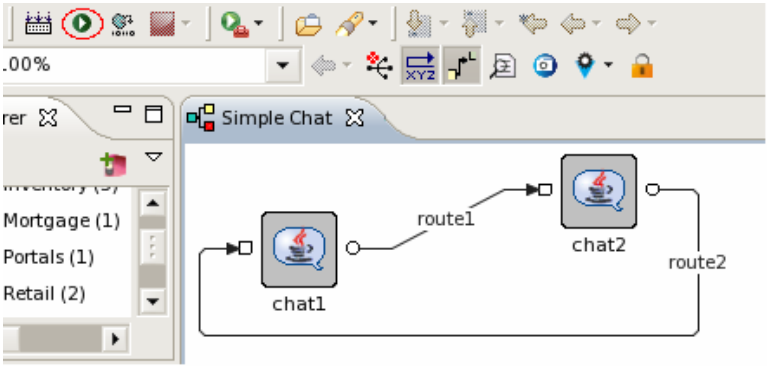


Figure 3.9.1: Launching an Event Process

3.10 Stopping an Event Process

- 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.

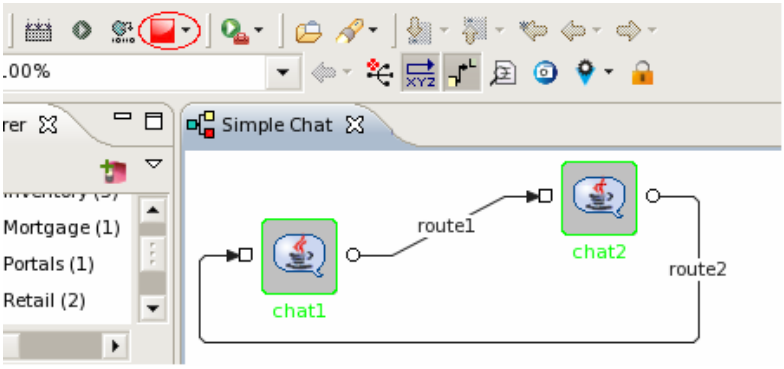


Figure 3.10.1: Stopping an Event Process

Note: When a component is killed, its name turns to red in the orchestration easel.

3.11 Synchronizing Event Processes

Fiorano has the capability to modify existing running applications on the fly. To do this, add another component-instance to the flow from the component palette, configure the component, and place and configure the appropriate routes. The application then needs to be synchronized to reflect the changes.

To synchronize event process, perform the following steps:

- Select the Event Process. Click the **Synchronize Event Process** button.

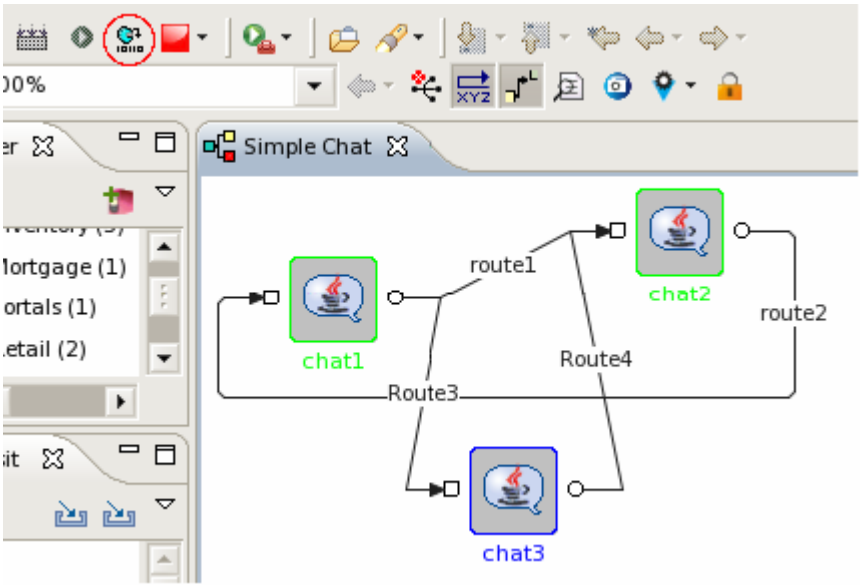


Figure 3.11.1: Event Process Synchronization

Chapter 4: 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 Process for various environments in a single Event Process.

4.1 Using Event Process Life Cycle Management

4.1.1 Setting Properties of Service Instances for Different Environments

Select the **Target Environment** in the **Environment Properties** tab of the Event Process comprising the service instance. Hereafter, 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 properties of an event process. 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 development environment and from a test.txt file in testing environment.

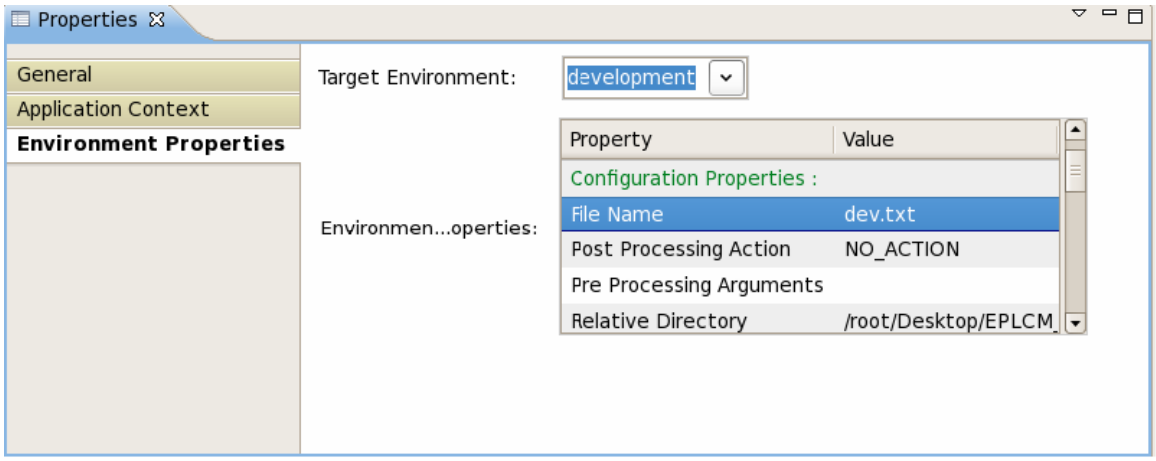


Figure 4.1.1: Environment Properties Tab

4.1.2 Running an Event Process on an Environment

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

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

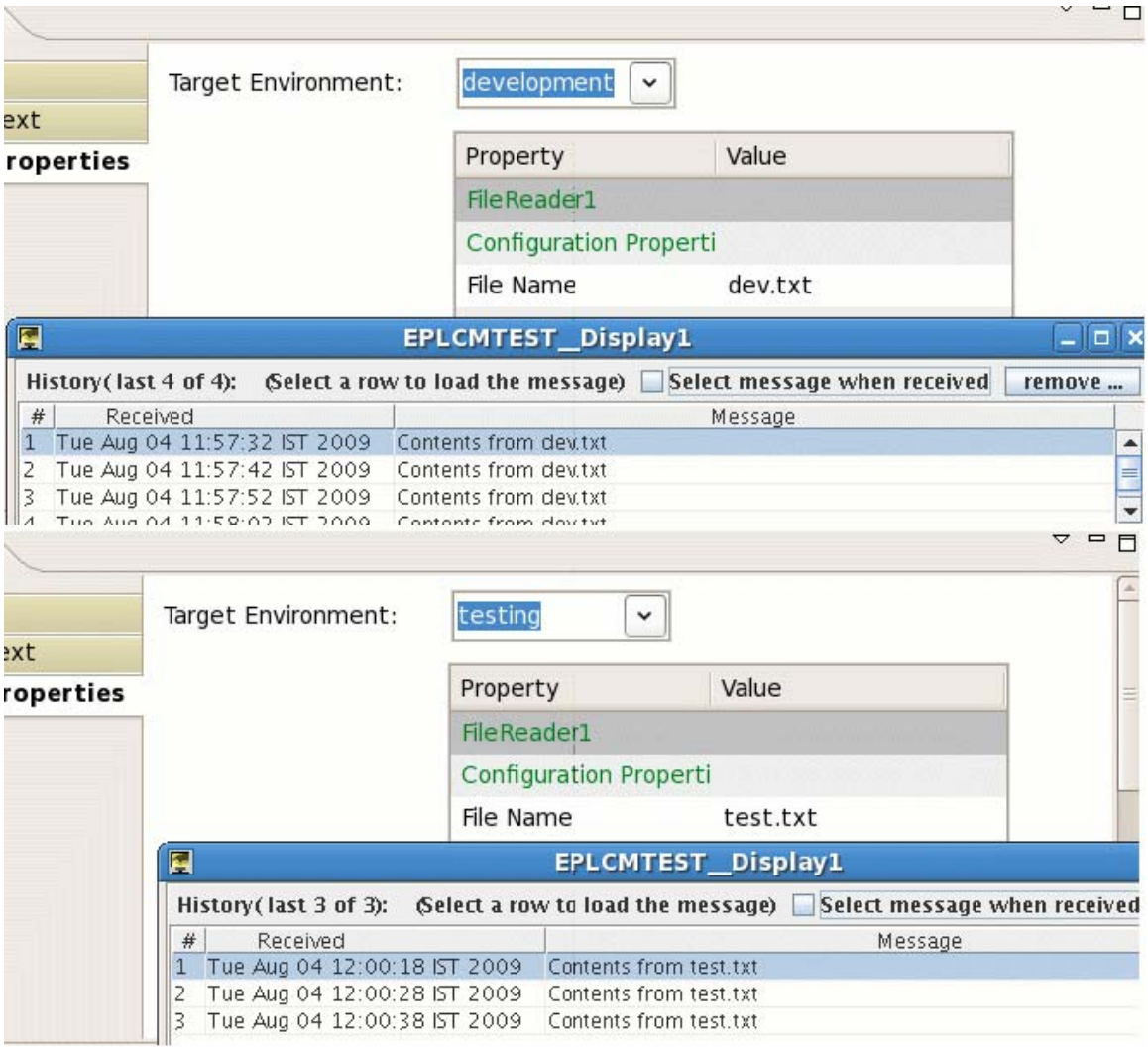


Figure 4.1.2: Display showing messages from file reader in different environments

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

Chapter 5: Debugging Event Process

5.1 Adding Breakpoint

To add a breakpoint to a route, perform the following steps in Fiorano Debugger View:

1. Go to Fiorano Debugger pane and click the **Add BreakPoint** button as shown in Figure 5.1.1 listing of all the available routes for selected Event Process appears as shown in Figure 5.1.2.

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

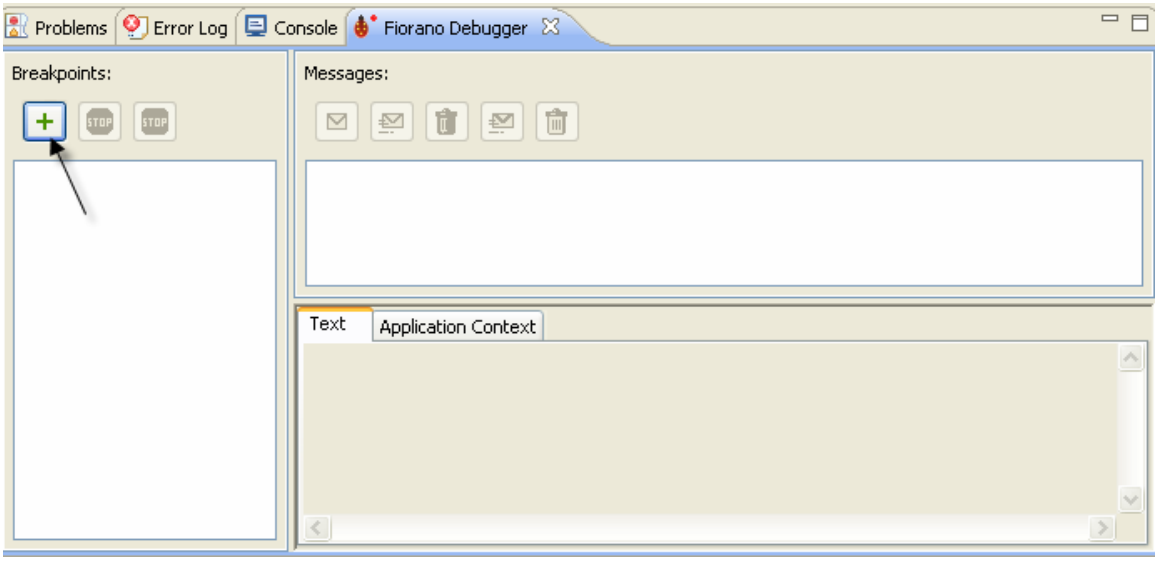


Figure 5.1.1: Adding break point in debugger

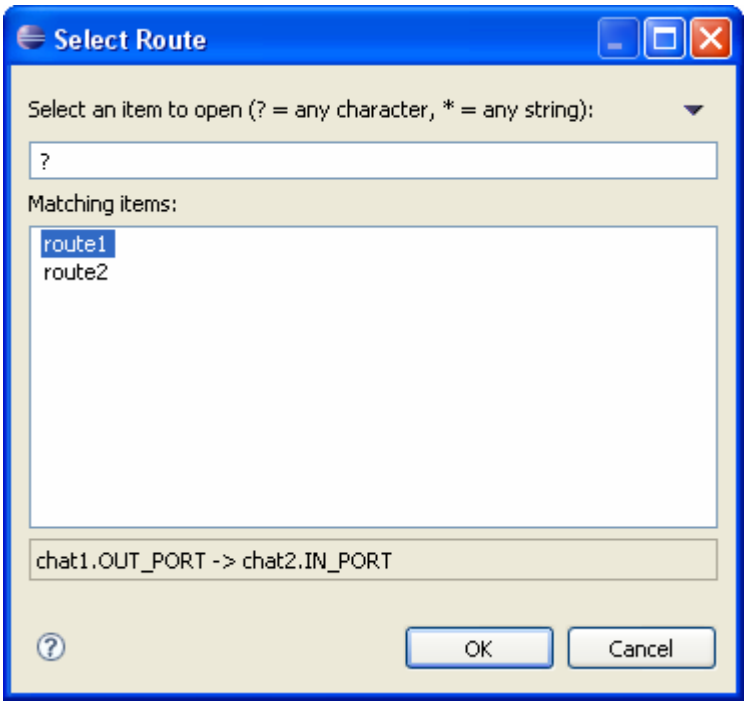


Figure 5.1.2: Select route to add breakpoint

2. Select the route on which you want to add breakpoint and the click **OK**. The breakpoint is added.

   Or

1. Right-click on route and select Inspector and click on the **Add Breakpoint** option as shown in Figure 5.1.3.

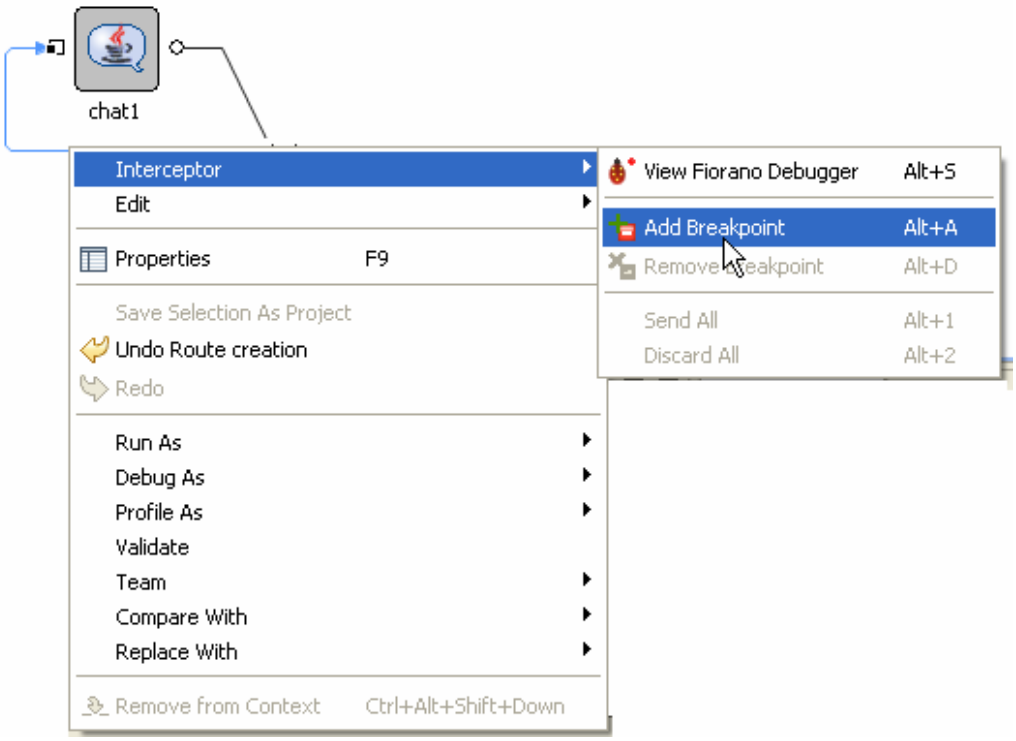


Figure 5.1.3: Adding breakpoint through route

5.2 Viewing Messages at Breakpoint

All the pending messages sent to a route which has breakpoint set on it are visible when clicked on that particular route in the breakpoint view as shown in Figure 5.2.1.

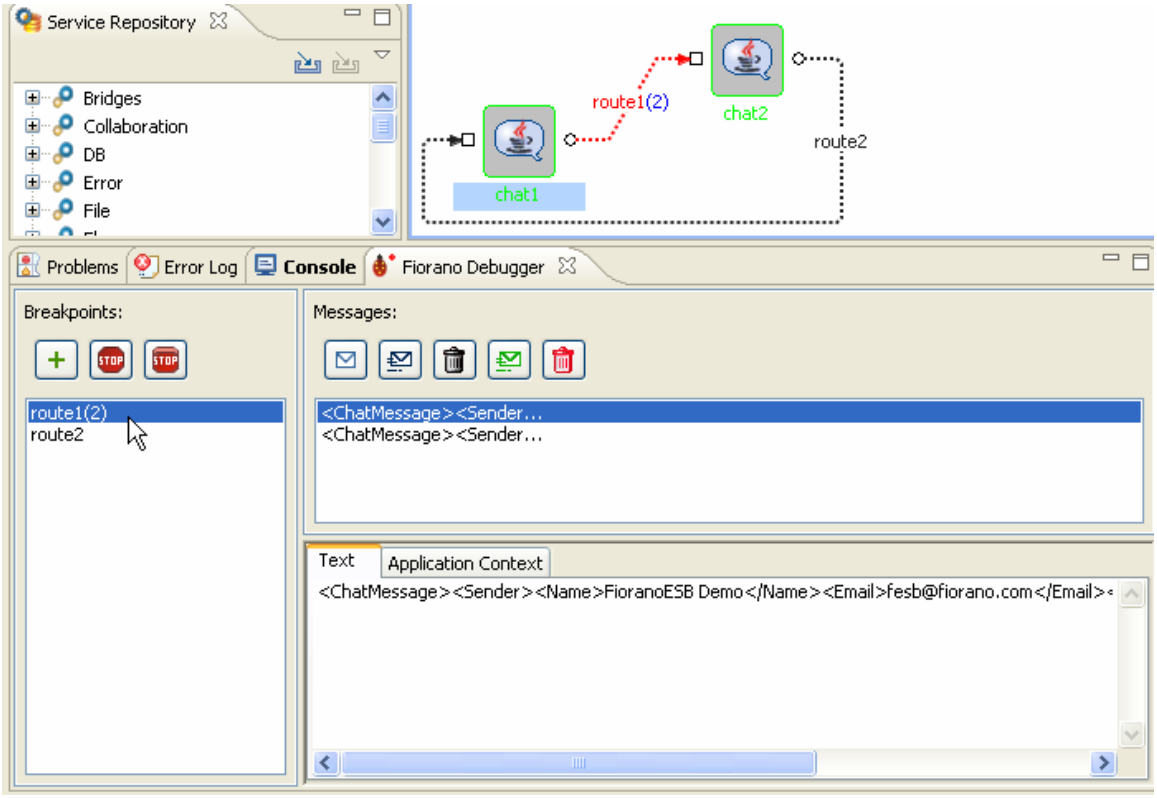


Figure 5.2.1: Message at breakpoint in Fiorano debugger

5.3 Editing Messages at Breakpoint

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

- Select the message which you want to edit and edit it in **Text** section as shown in Figure 5.3.1.

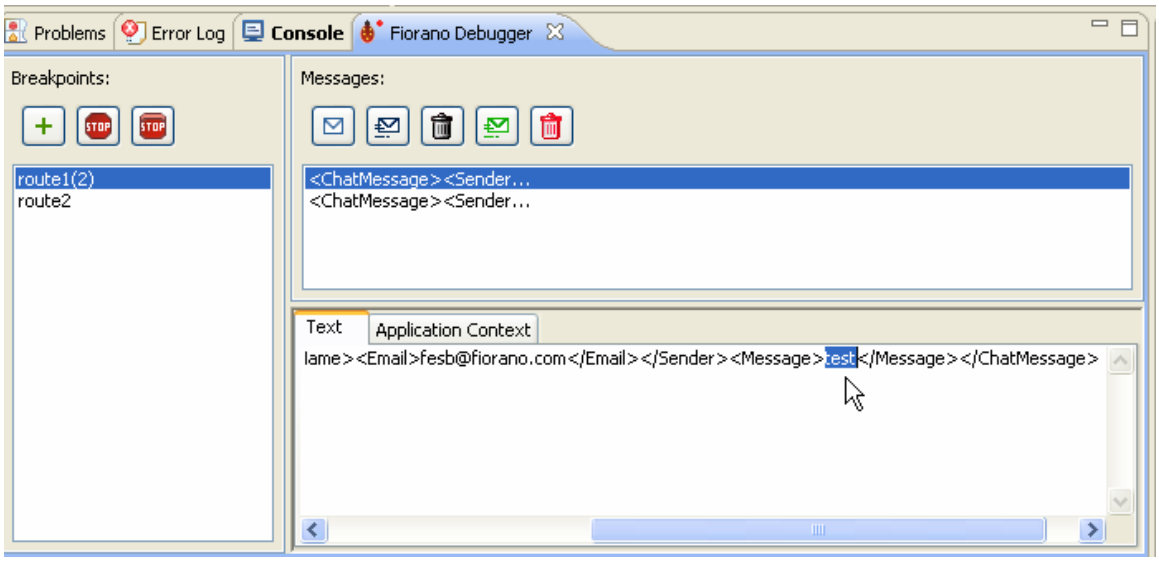


Fig 5.3.1: Edit message in Fiorano debugger

5.4 Inserting Messages into Breakpoint

You can choose to insert messages 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 5.4.1.

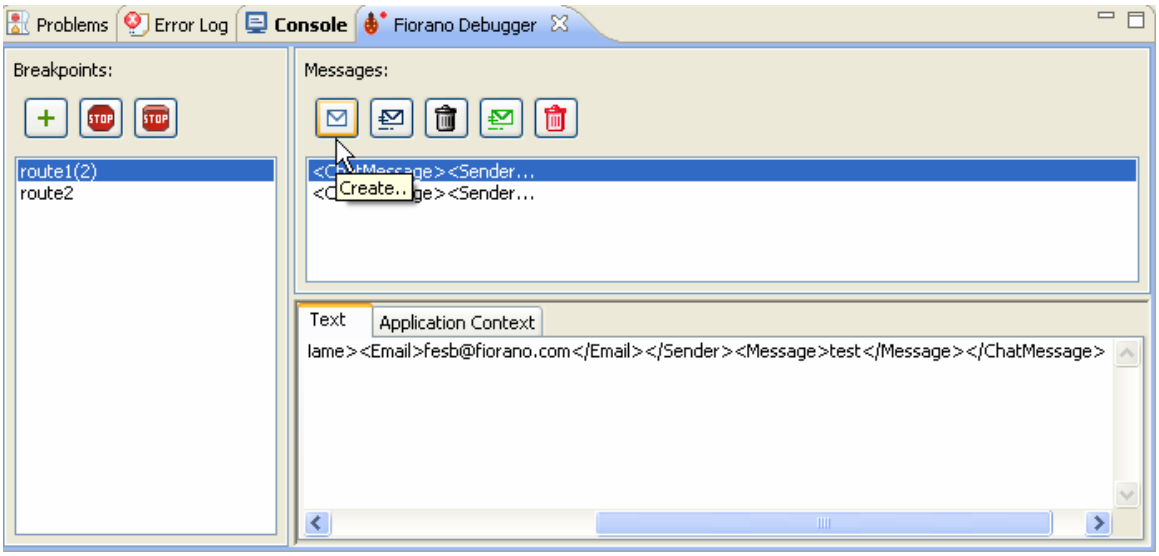


Fig 5.4.1 Create message in Fiorano debugger

2. Choose the type of message, you want to create (either XML or Text message) as shown in Figure 5.4.2 and click **OK** .The new message appears on the breakpoint in the Messages view.

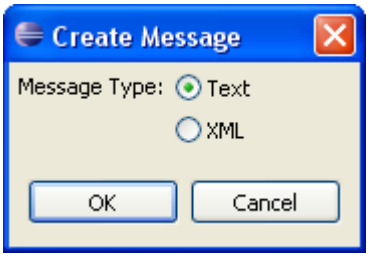


Fig 5.4.2 select type of new message

5.5 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 you want to release and click **Send** the button shown in Figure 5.5.1. The sent message will now arrive at the destination port of the route selected.

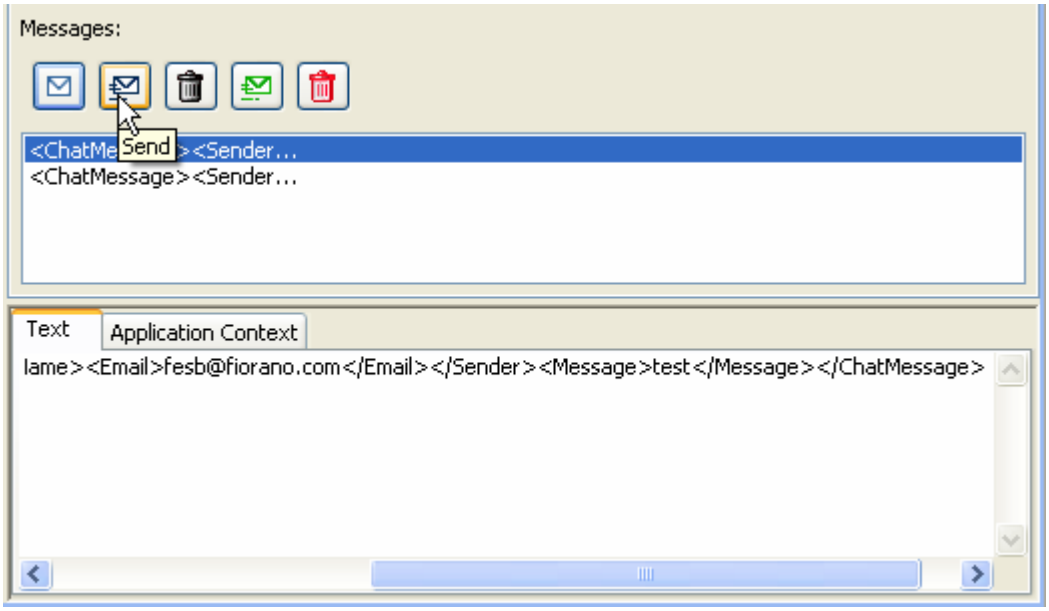


Figure 5.5.1: Send message in Fiorano Debugger

2. All messages on Breakpoint can be send all at a time by clicking on send all button as shown in Figure 5.5.2.

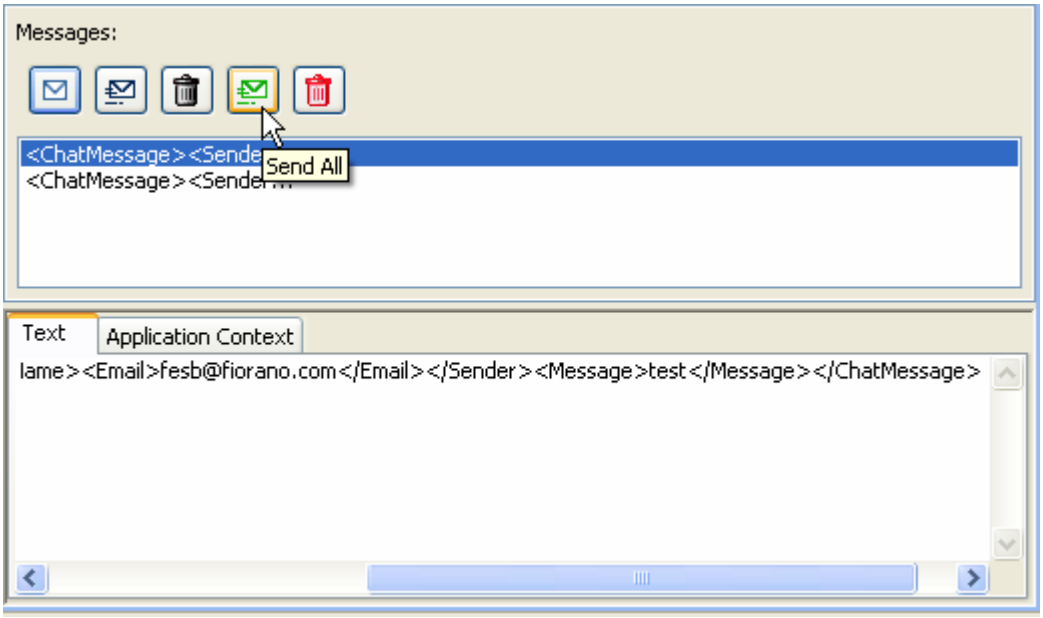


Figure 5.5.2: send all messages in Fiorano debugger

5.6 Discard Messages from Breakpoint

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

1. Select the message, you want to discard and click the **Discard** button shown in Figure 5.6.1. The discarded message will be removed from Breakpiont.

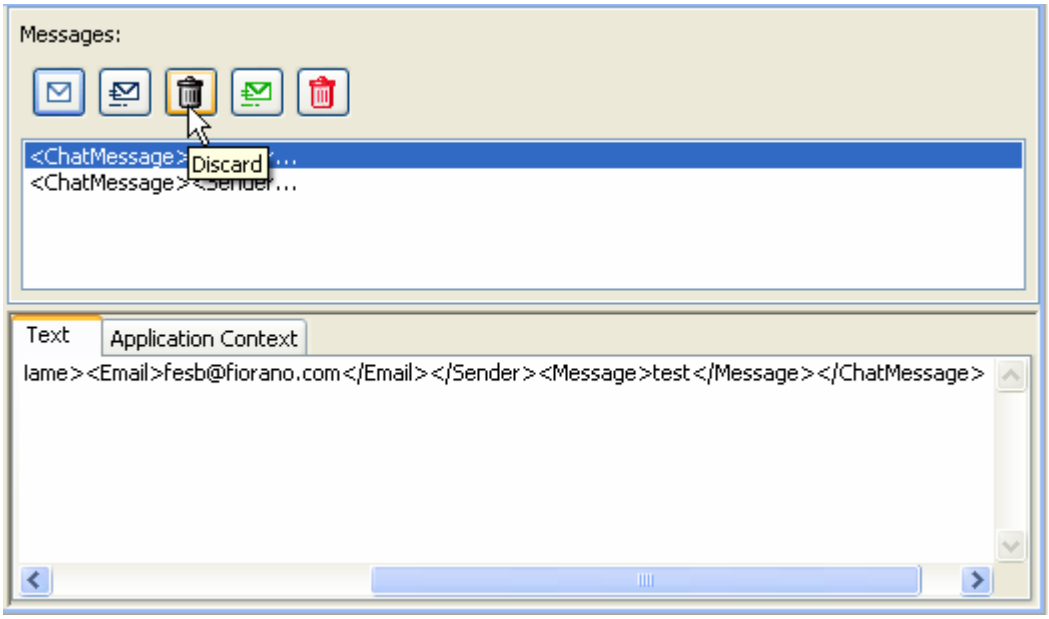


Figure 5.6.1: Discard message in Fiorano debugger

2. All messages on Breakpoint can be discarded all at a time by clicking on **Discard All** button as shown in Figure 5.6.2.

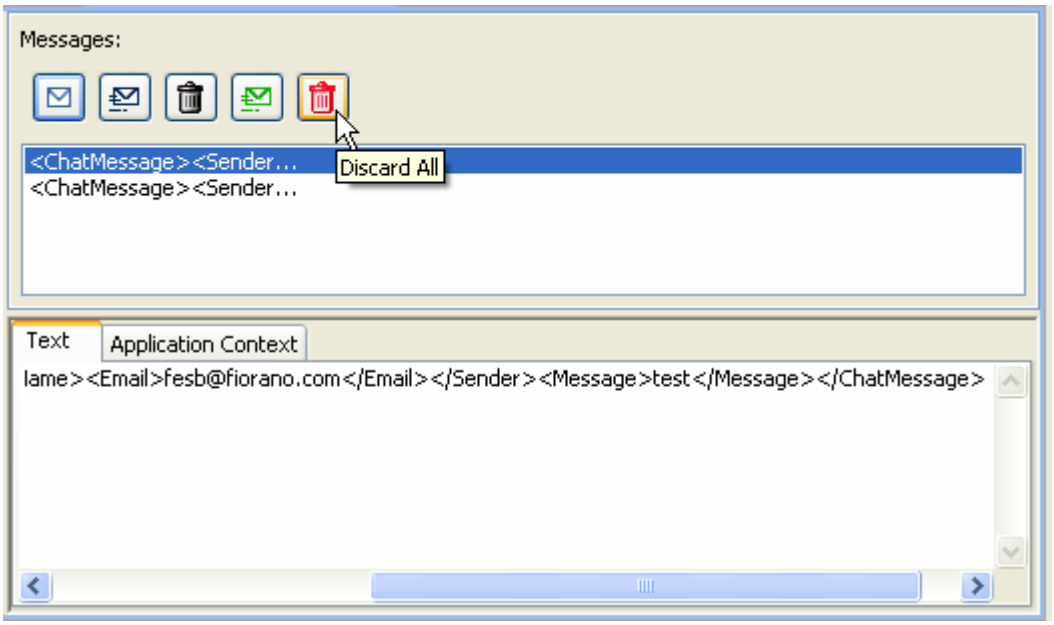


Figure 5.6.2: Discard All messages in Fiorano debugger

Chapter 6: Services

6.1 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 Service Repository view and select the **`Edit...`** option as shown in Figure 6.1.1.

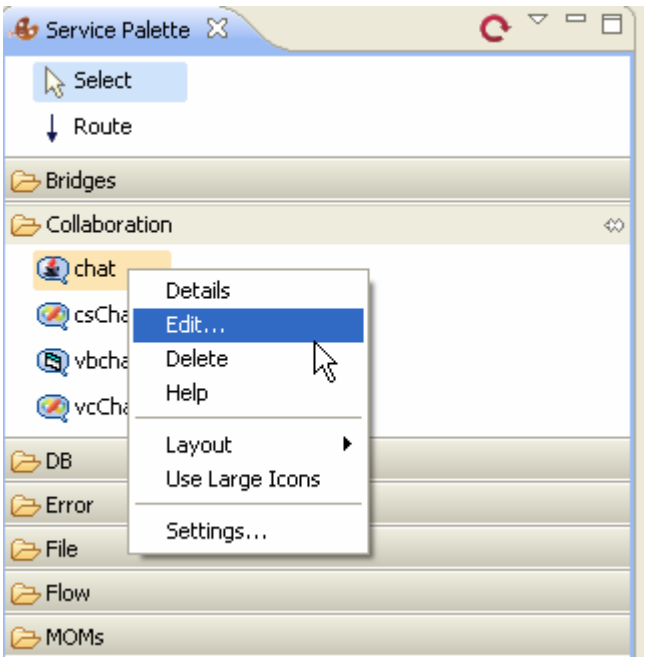


Figure 6.1.1: Edit option

2. The **`ServiceDescriptor.xml`** of the selected service is opened in the Service Descriptor Editor as shown in Fig 6.1.2.

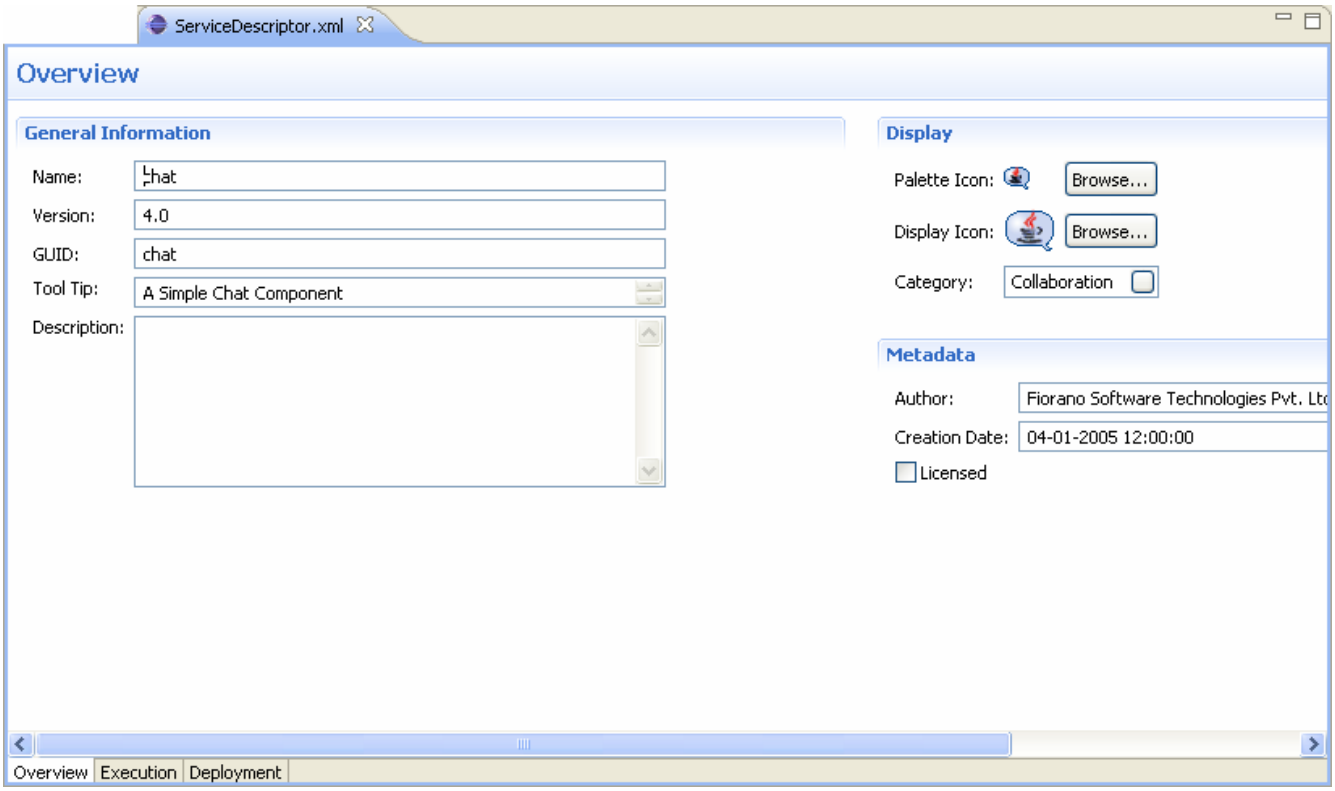


Fig 6.1.2: ServiceDescriptor.xml

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 6.1.3.

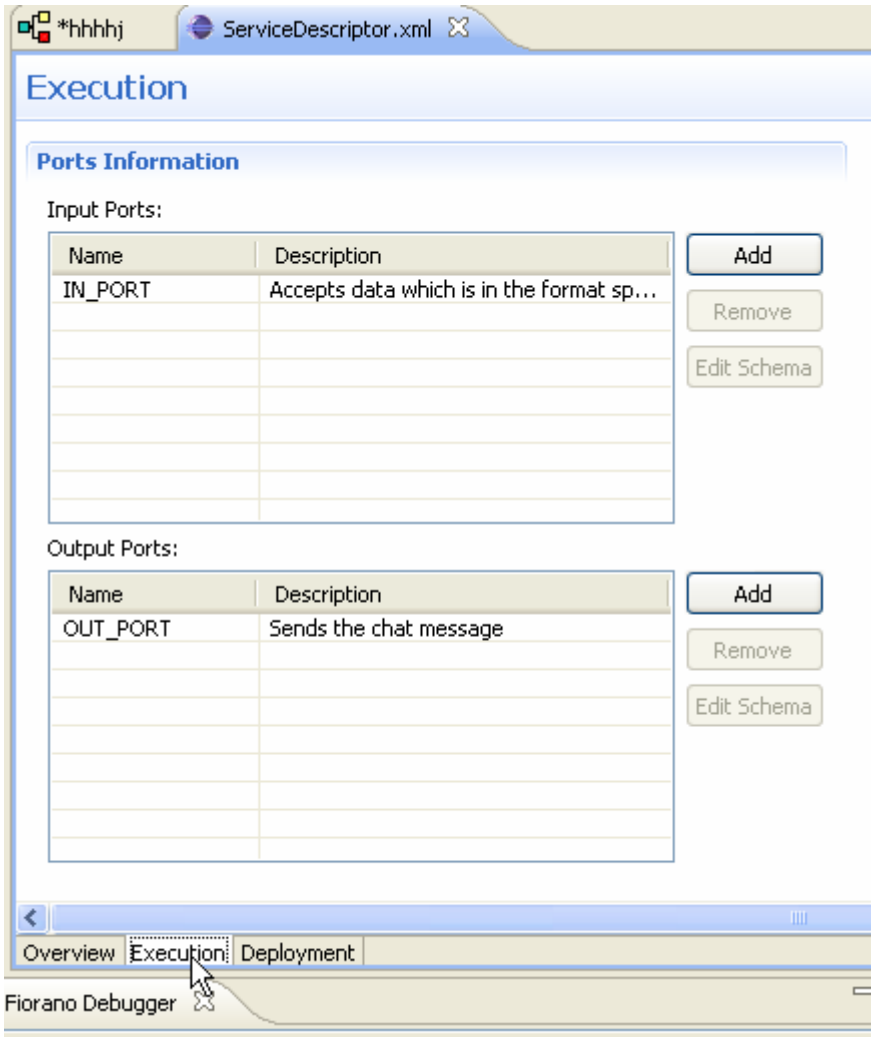


Figure 6.1.3: Sections under Service Descriptor

6.1.1 Overview Section

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

The information used to identify the service is shown under General Information section. The user can change the Name, Version, GUID, Tool Tip, and Description of the component in this section. Figure 6.1.4 illustrates the General Information section.

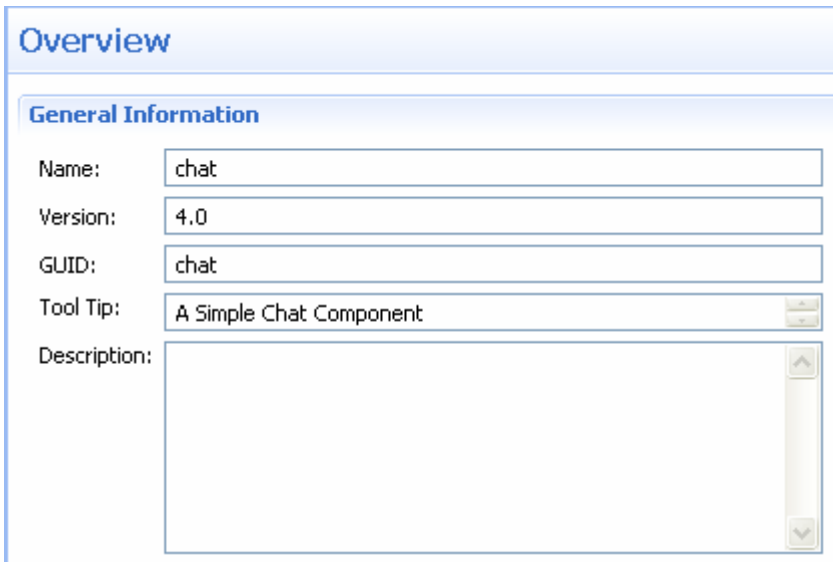


Figure 6.1.4: General Information

In the Display section, the 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 6.1.5).

In the Metadata section, the information about authors of the service, creation date and time of the service and licensing mode are provided (Figure 6.1.5).

Note: The Creation Date field cannot be manually changed.

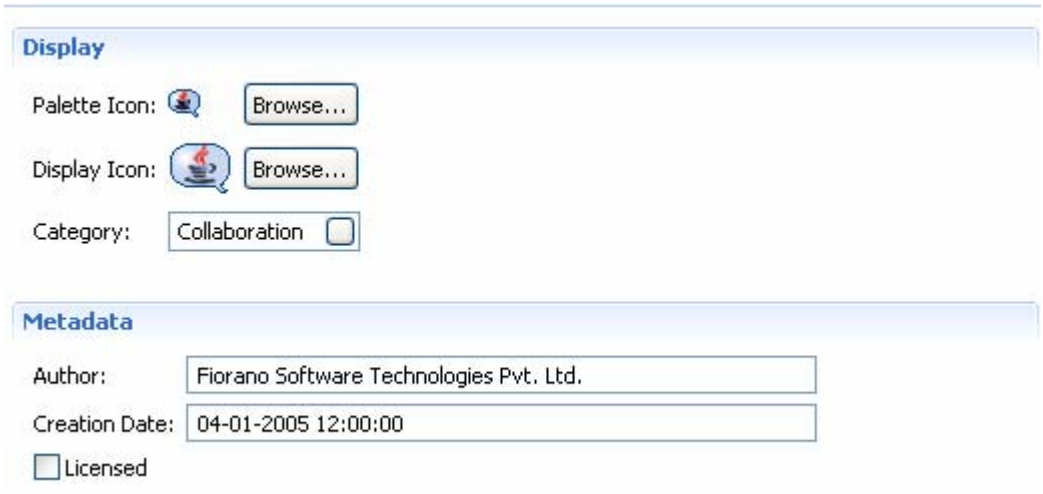


Figure 6.1.5: Display and Metadata sections

6.1.2 Execution Section

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

6.1.2.1 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 Port Information section (Figure 6.1.6).

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

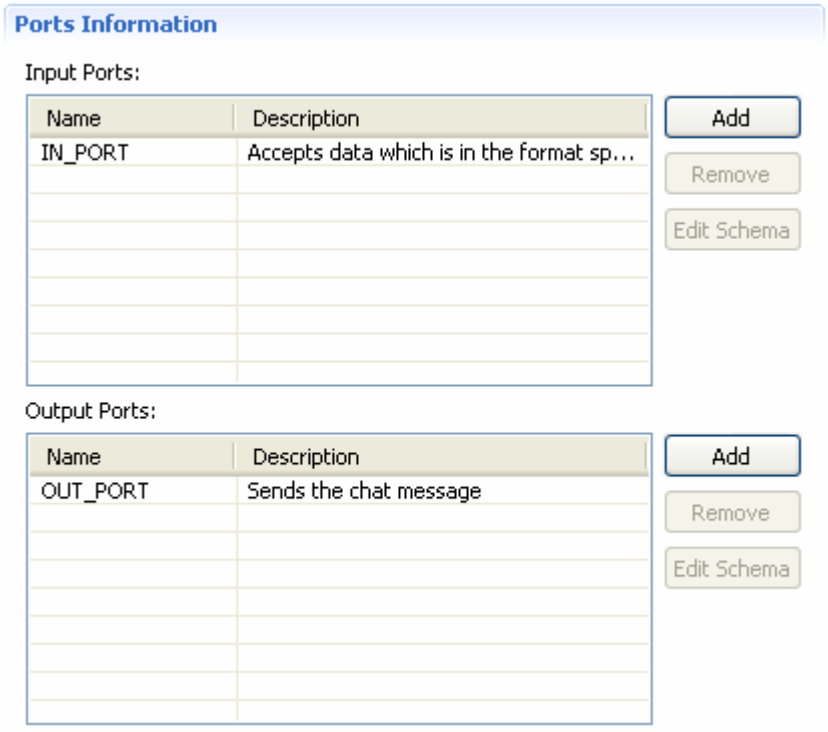


Figure 6.1.6: Port Information section

6.1.2.2 Support

In the Support section, Failover Supported and Transaction Supported options are available as shown in Figure 6.1.7.

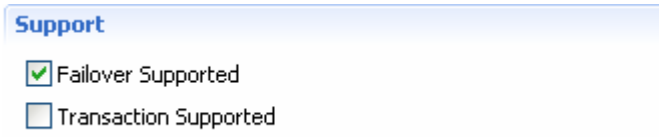


Figure 6.1.7: Support section

Failover Supported

If the Failover Supported option is selected, during 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 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.

6.1.2.3 Launch Configuration

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

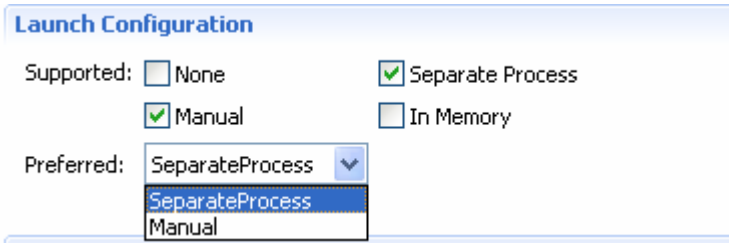


Figure 6.1.8: Launch Configuration section

6.1.2.4 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 level which are lower than the selected log level will not be written to log files.

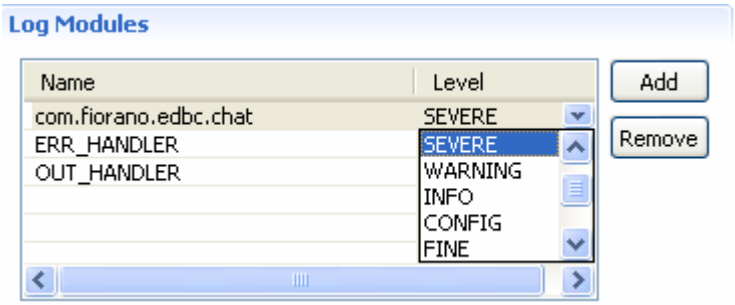


Figure 6.1.9: Log Modules section

6.1.2.4 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 Iin-memory mode.

Working directory specifies the directory which will be the service's runtime directory when launched in 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.

By defining the command line arguments that can be passed to the component during the launch of the component, these command line arguments are captured as runtime arguments in this panel.

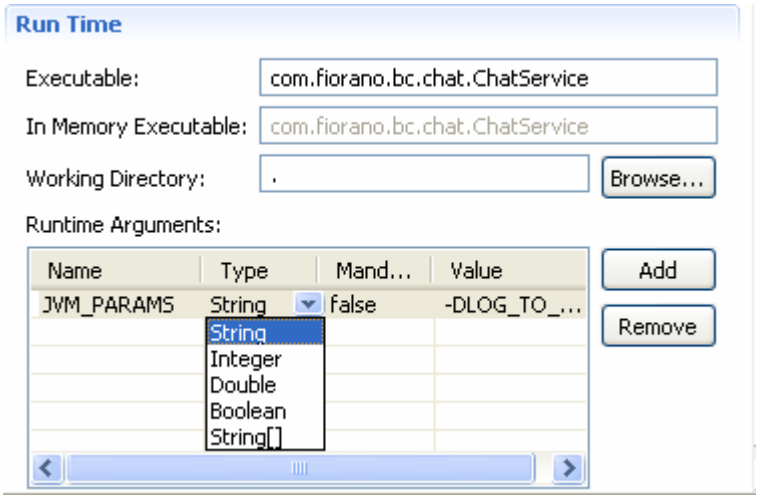


Figure 6.1.10: Runtime section

6.1.3 Deployment Section

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

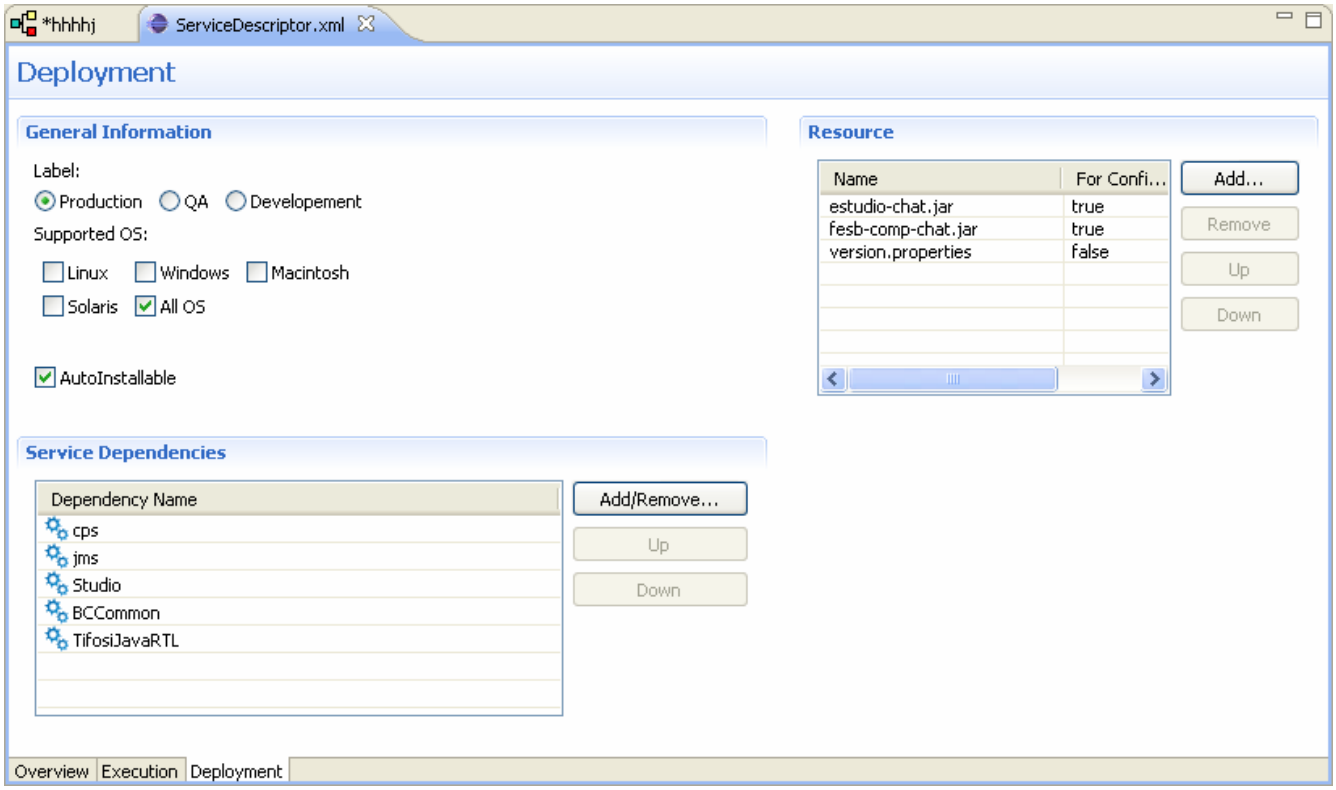


Figure 6.1.11: Deployment page

6.1.3.1 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 icon. The order is used to determine the classpath of the service

6.1.3.2 Service Dependencies

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

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

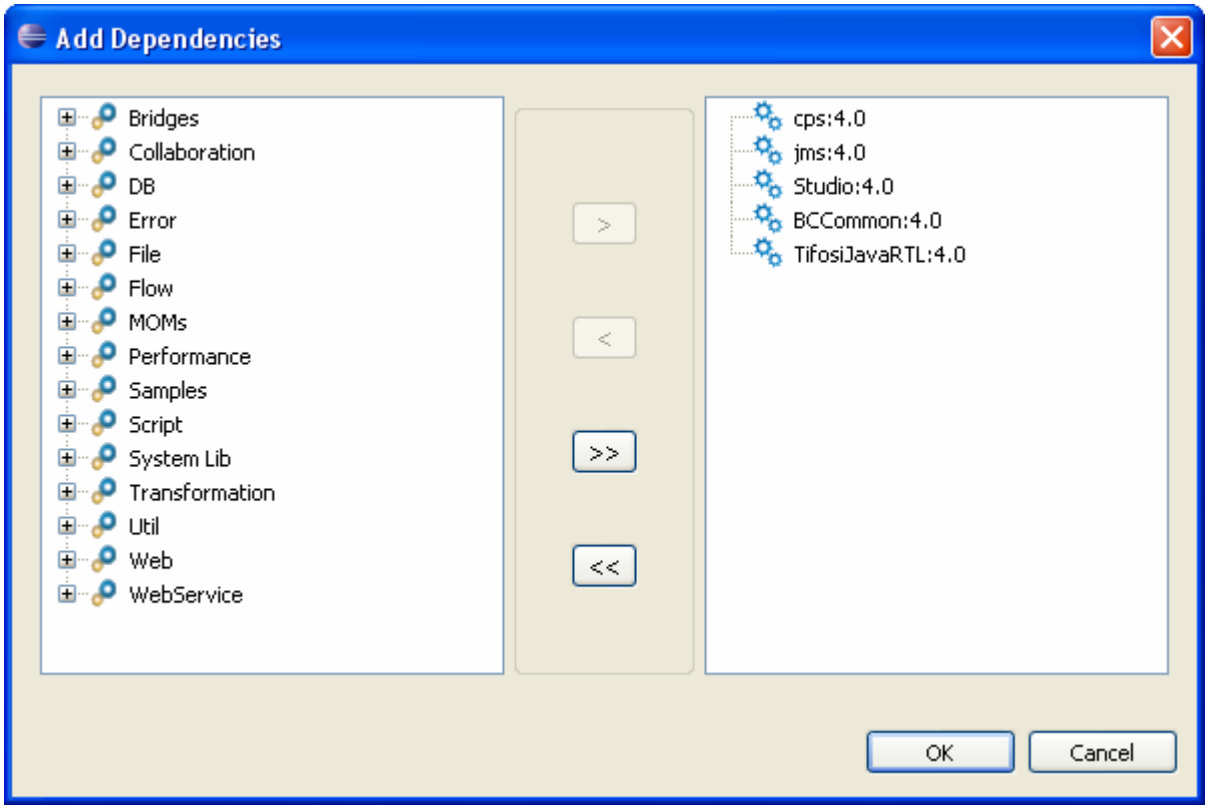


Figure 6.1.12: Service Dependencies section

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

6.2 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 6.2.1.

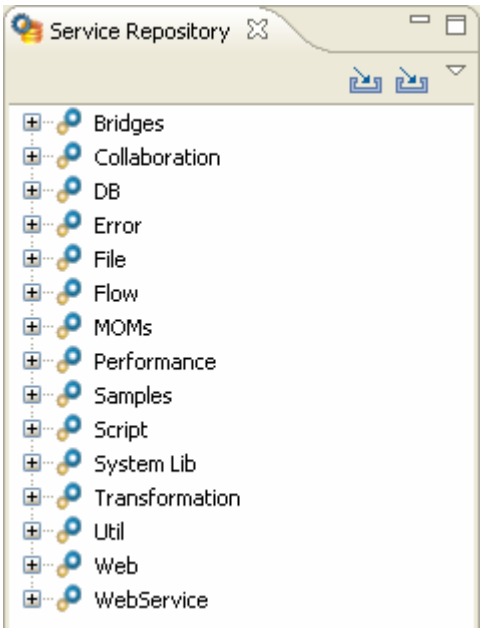


Figure 6.2.1: Service repository

6.2.1 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 to Server** from the context menu. This opens **Export service to server** dialog box as shown in Figure 6.2.2.

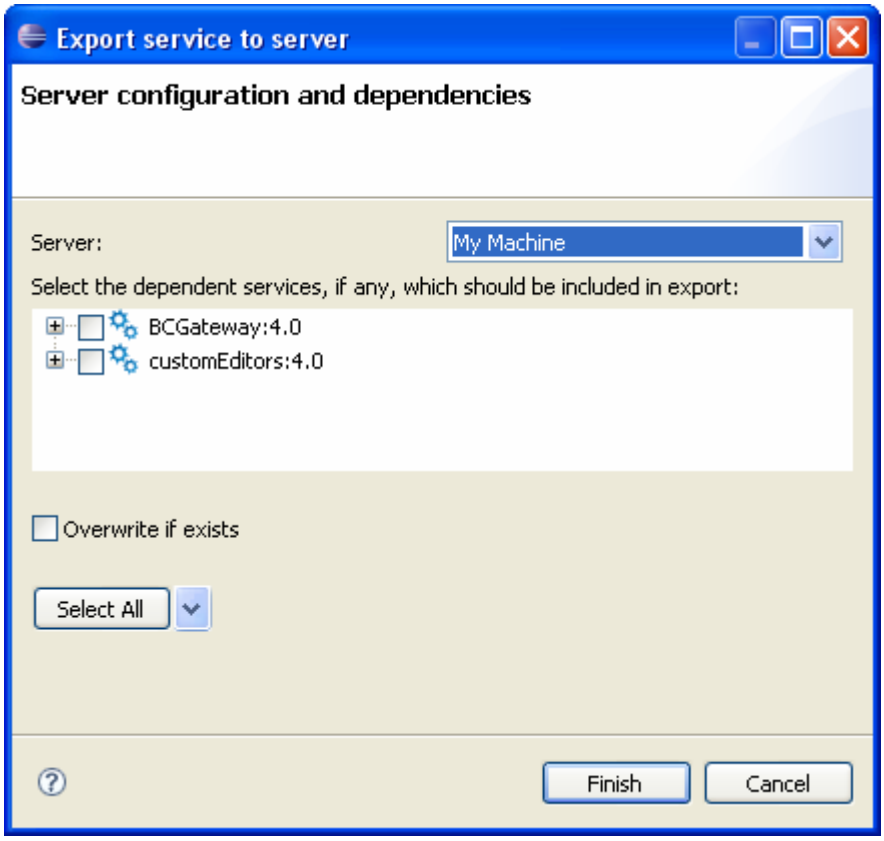


Figure 6.2.2: Export service to server

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

If **Overwrite if exists** checkbox is selected, the services in the server will be over written by the one in service repository, otherwise conflicting services will not export to the server.

6.2.2 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 6.2.3.

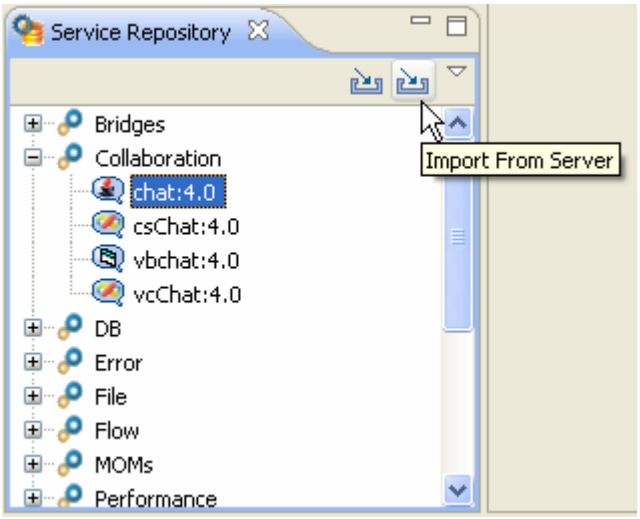


Figure 6.2.3: Import from Server option

This opens **Import service from server** dialog box as shown in Figure 6.2.4.

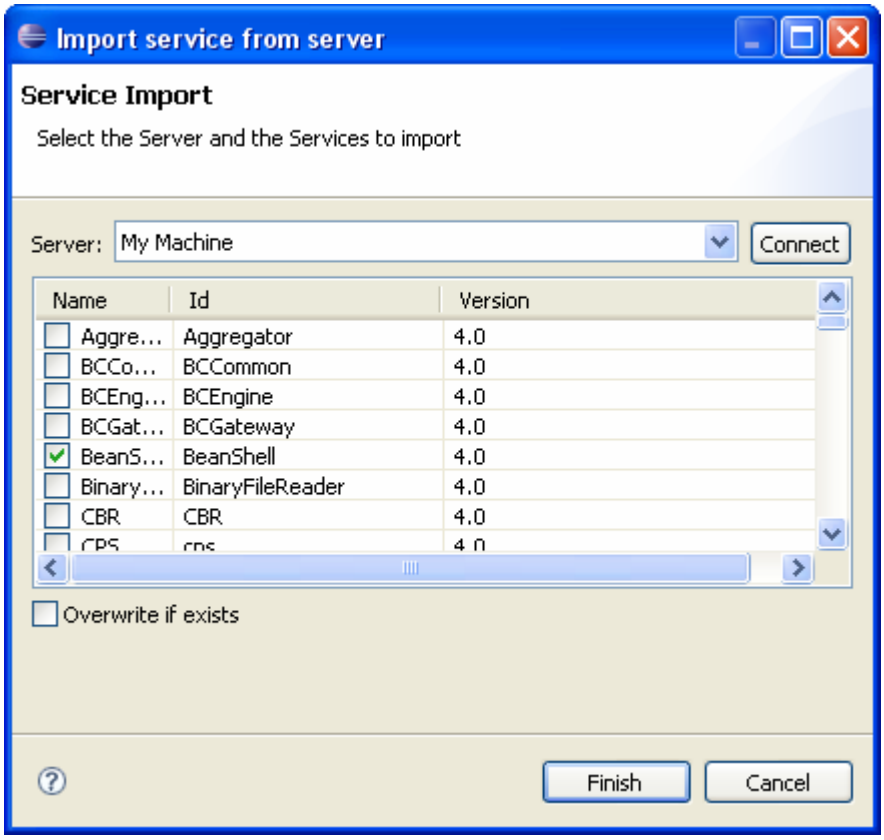


Figure 6.2.4: Import service from server

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

If **Overwrite if exists** checkbox is selected, service in the service repository will be over written by the one in server, otherwise conflicting services are not imported from server.

6.2.3 Exporting Services to Local Disk

The Services in Service Repository can be exported to a local disk by right-clicking the service and selecting **Export service to local disk** option from context menu. This opens **Export service to local disk** dialog box as shown in Fig 6.2.5.

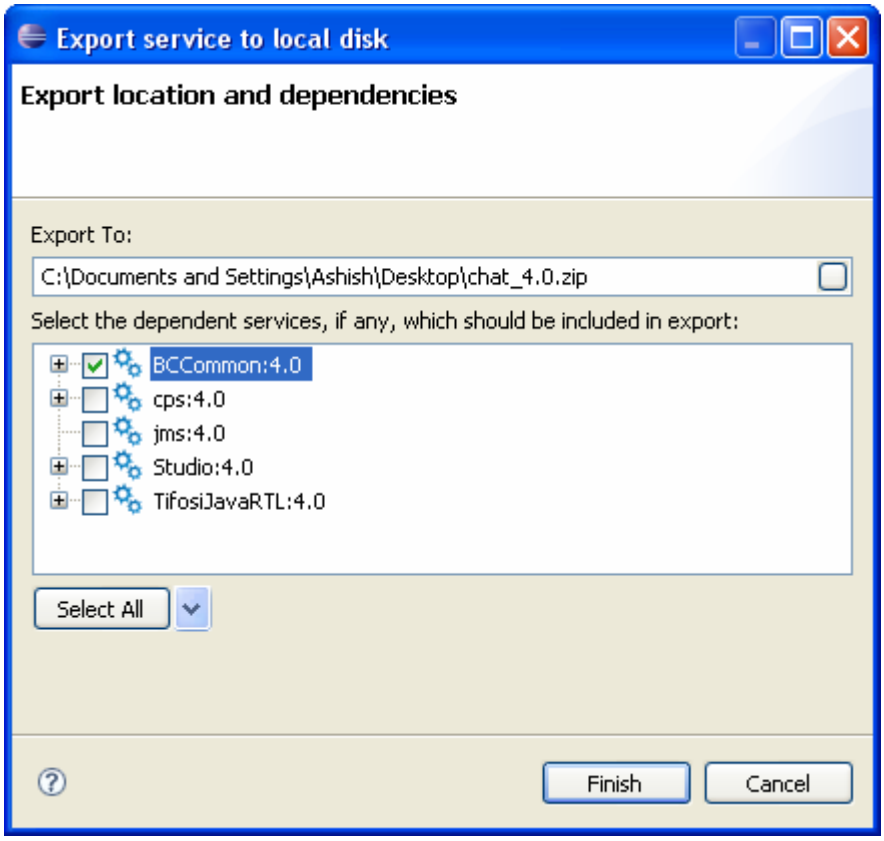


Figure 6.2.5: Export service to Local Disk

You can chose the export location, by default the selected service gets included in the export. Select the services from the tree to be exported and click the **Finish** button as shown in Figure 6.2.5.

6.2.4 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 Fig 6.2.6.

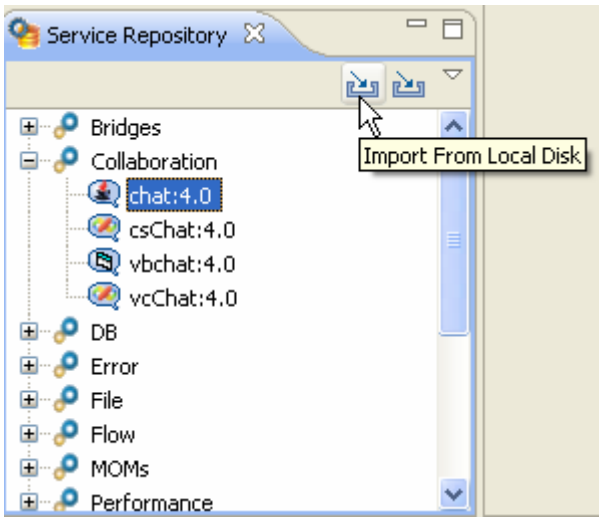


Figure 6.2.6: Import from Local Disk button

This opens **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 a dependency tree as shown in Fig 6.2.7.

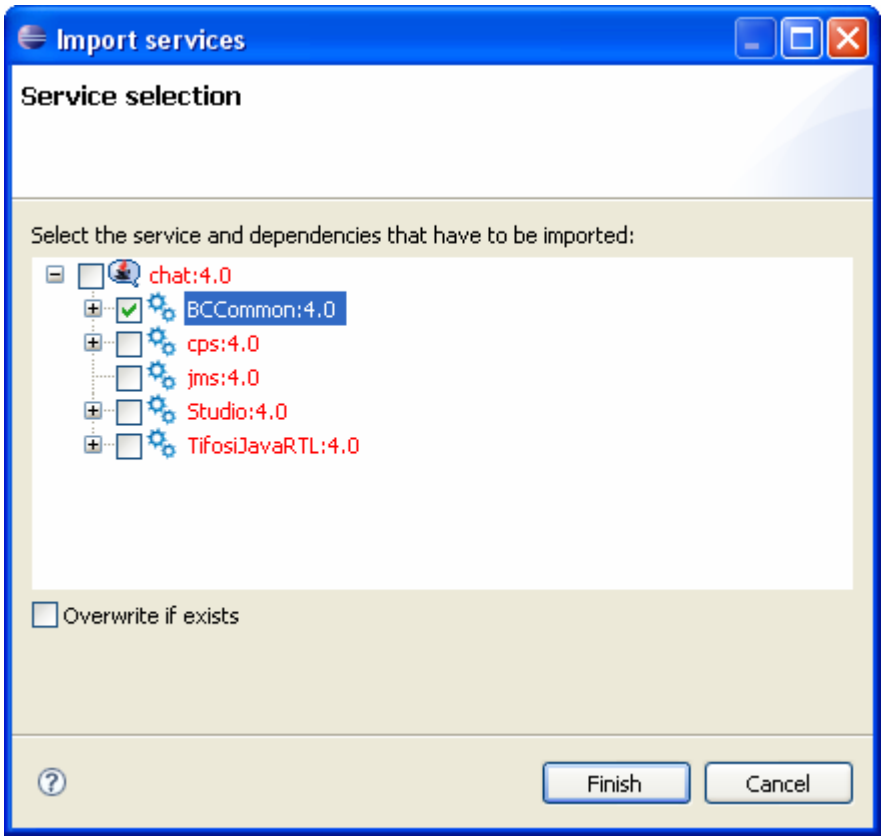


Figure 6.2.7: Import services dialog box

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

If 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 local disk.

6.3 Service Repository (Online Event Process Development)

In Online Event Process Development perspective, the services selected in server explorer are shown in service repository.

The service repository can be viewed by opening the Service Repository view which displays categorized services as shown in Fig 6.3.1.

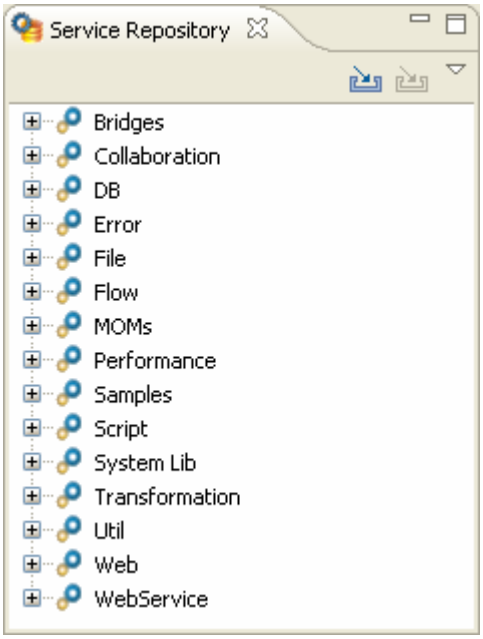


Figure 6.3.1: service repository (Online Event Process Development perspective)

6.3.1 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 6.3.2.

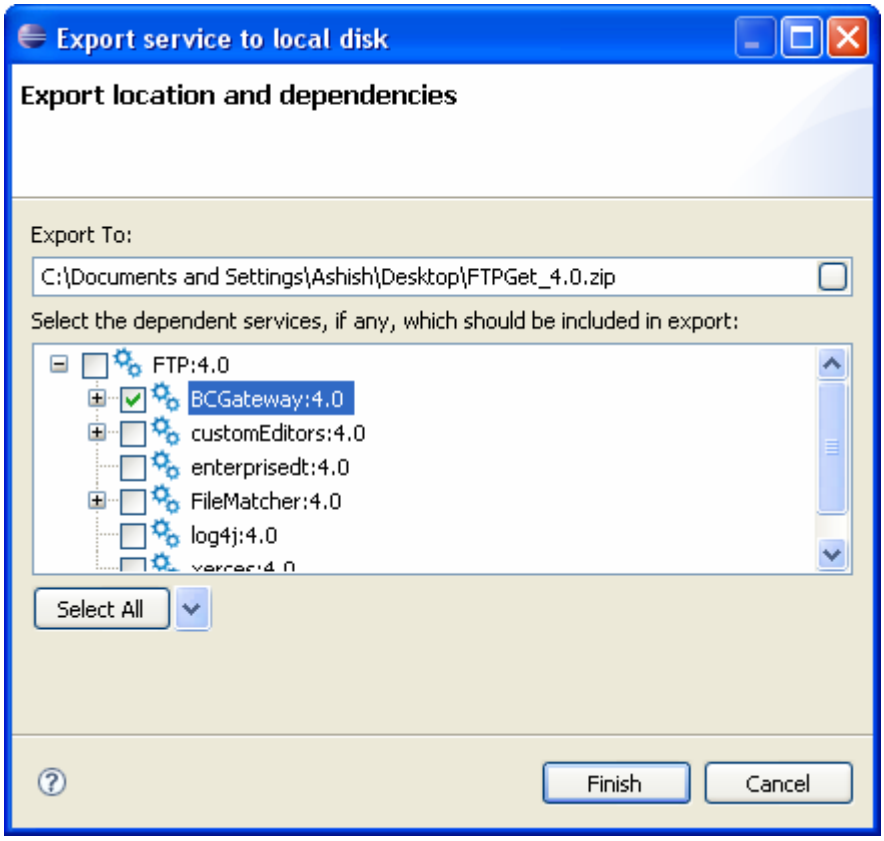


Figure 6.3.2: Export service to Local Disk

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, they have to be selected from the tree as shown in Figure 6.3.2.

6.3.2 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 6.3.3.

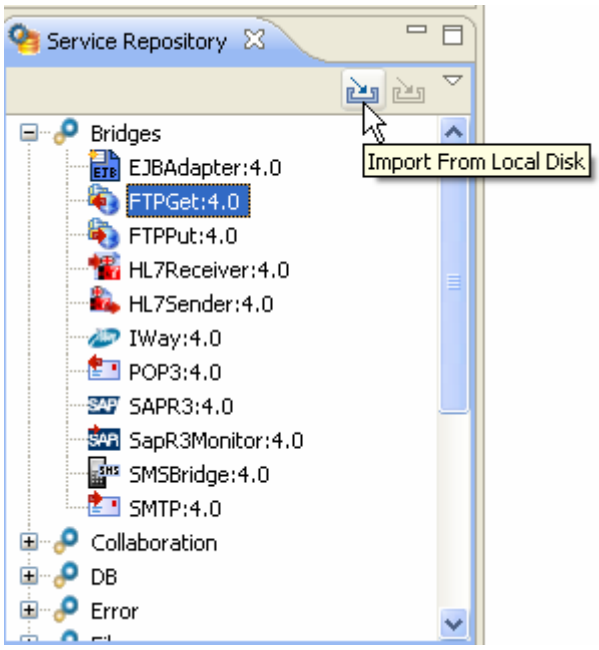


Figure 6.3.3: Import from Local Disk button

This opens a file selection dialog box with which the zip file containing services on the disk is selected. Upon selection, Import services dialog box appears in which the services in the zip file are shown in a dependency tree format as shown in Figure 6.3.4.

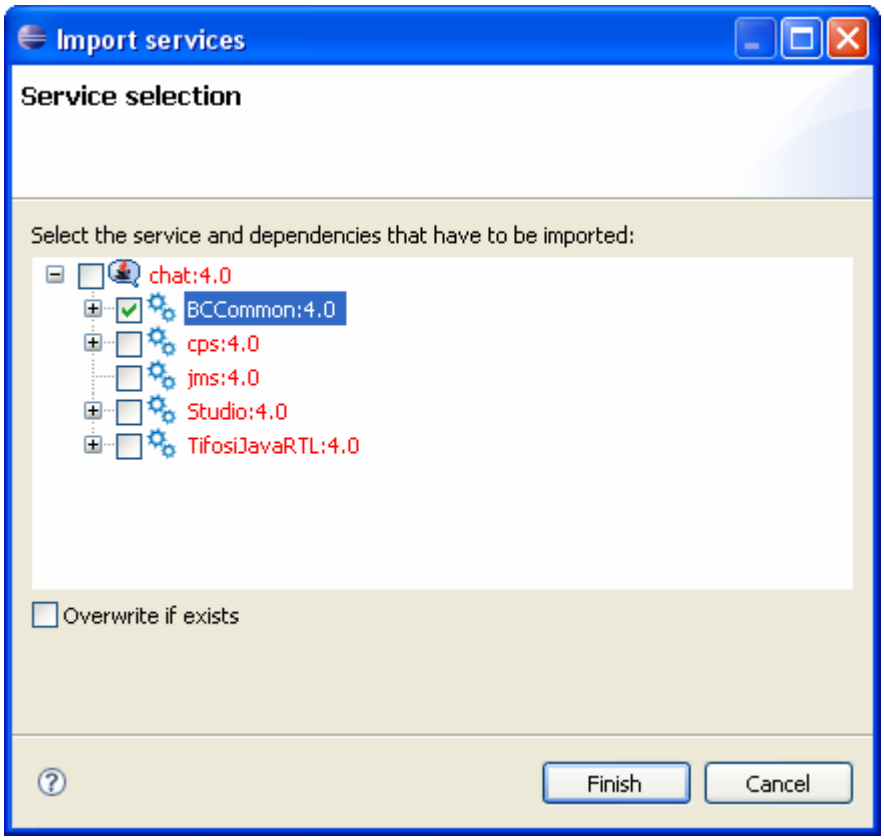


Figure 6.3.4: Import services dialog box

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

If **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 local disk.

Chapter 7: Service Creation

Apart from the exhaustive list of pre-built services, custom services can be written, built, and deployed into 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)

7.1 Service Generation

To create a new service - **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.

7.1.1 Service Location

The destination folder in which the component has to be created 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.

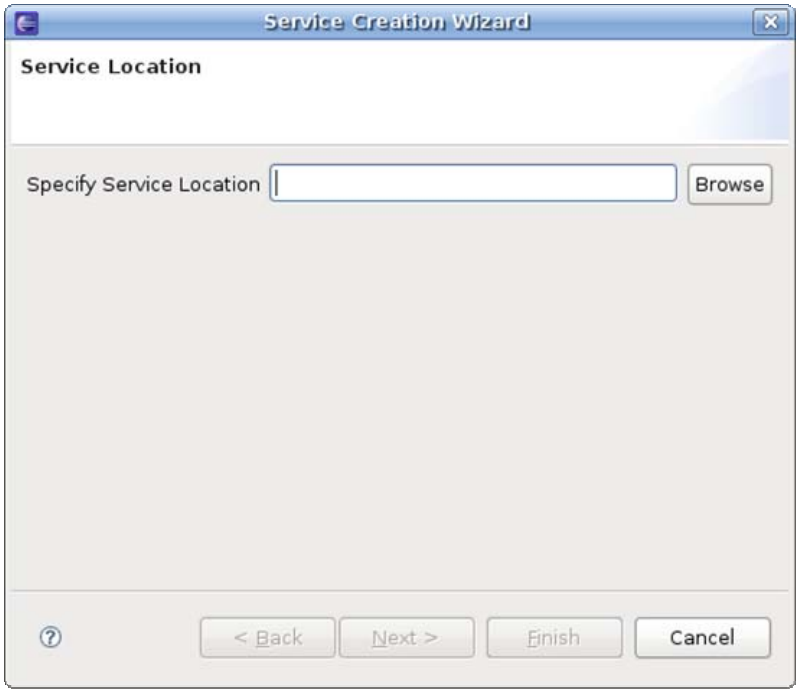


Figure 7.1.1: Specific Service Location

7.1.2 Basic Details

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

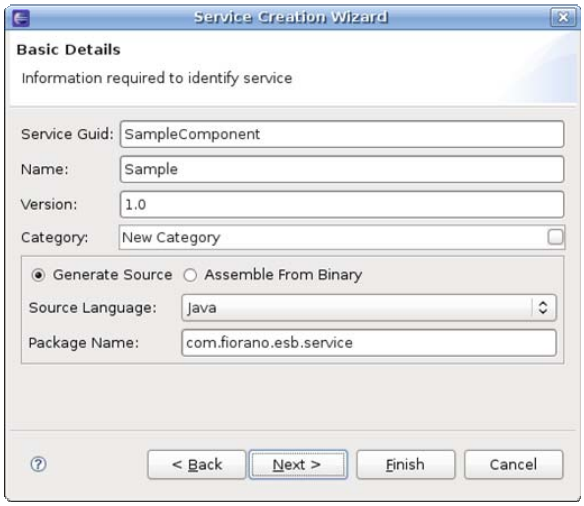


Figure 7.1.2: Service creation wizard

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 against the Category field. On clicking ellipsis, the **Category Selection** dialog box appears as shown in Figure 7.1.3. Multiple Categories can also be selected in the Category Selection dialog box.

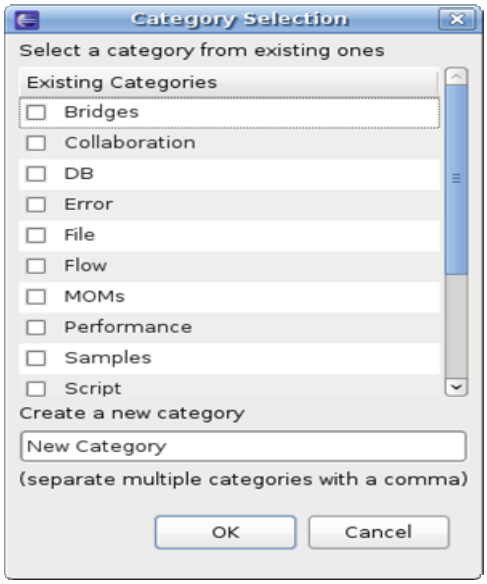


Figure 7.1.3: Category Selection dialog box

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

7.1.3 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.

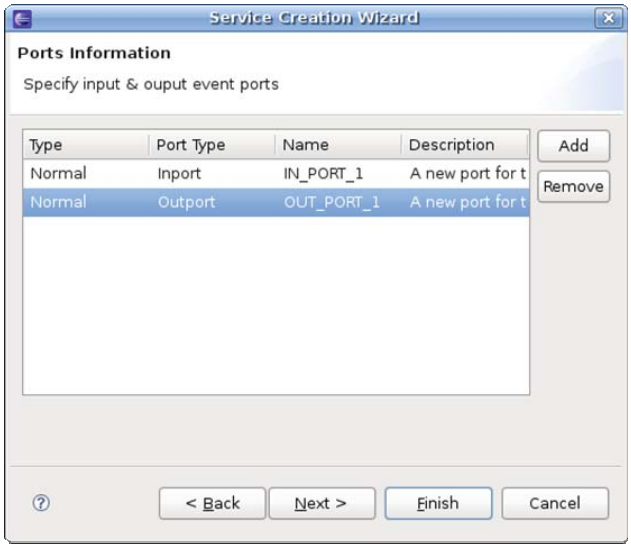


Figure 7.1.4: Ports Information

7.1.4 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.

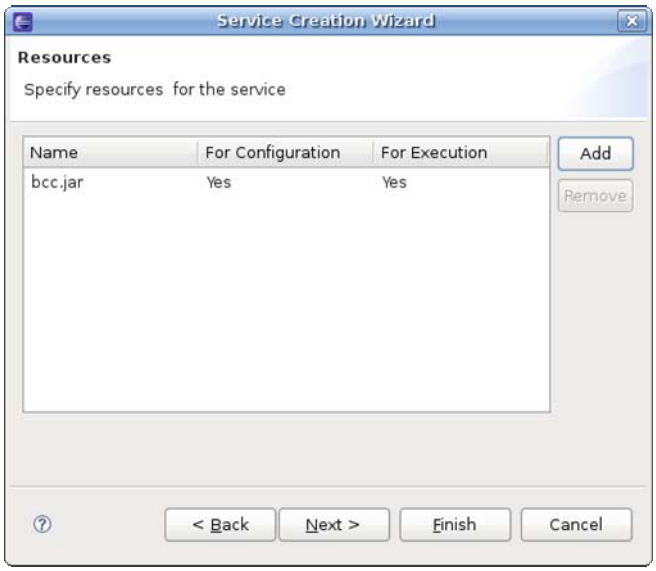


Figure 7.1.5: Resources section

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

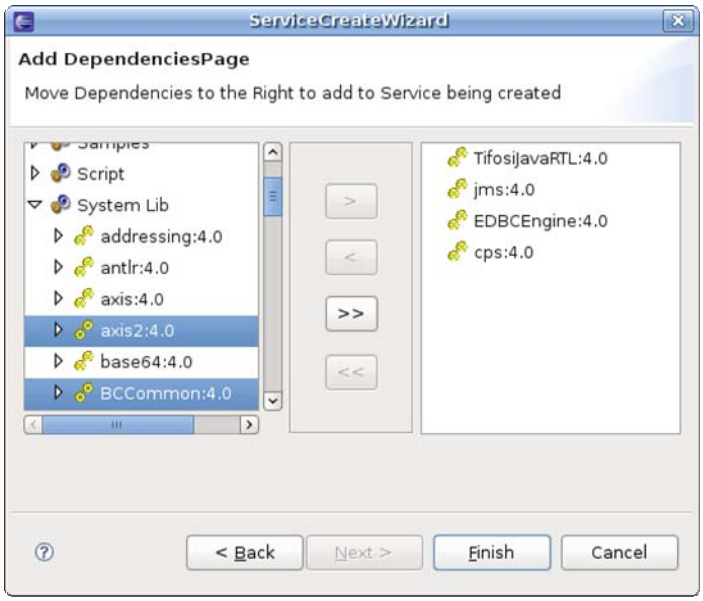


Figure 7.1.6: Dependencies

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

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

7.2 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:

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

Figure 7.2.1: ant register

4. This builds the service's sources and registers the service with the Enterprise Server.

To view the service in eStudio service palette, perform the following steps:

1. Execute the command ant **deployToStudio**.

Figure 7.2.2: deployToStudio

2. The service is now available in eStudio Service Palette and can be used in composing Event Processes.

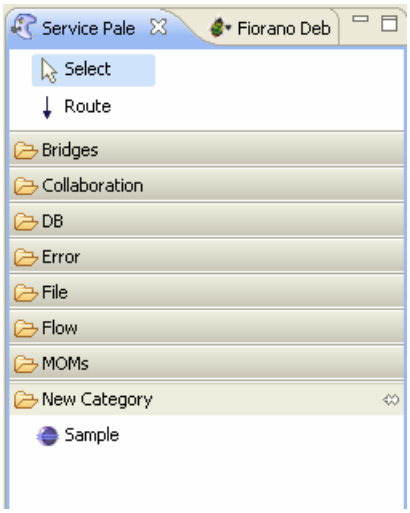


Figure 7.2.3: Service Palette

Chapter 8: eeMapper

The Fiorano eeMapper 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 eeMapper 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.

8.1 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.
- Validate the transformation.
- Define the transformation (mapping) with simple drag-and-drop actions.

8.2 Fiorano eMapper Environment

The Fiorano eMapper tool consists of the following interface elements:

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

The interface of the Fiorano eMapper tool is displayed in Figure 8.2.1.

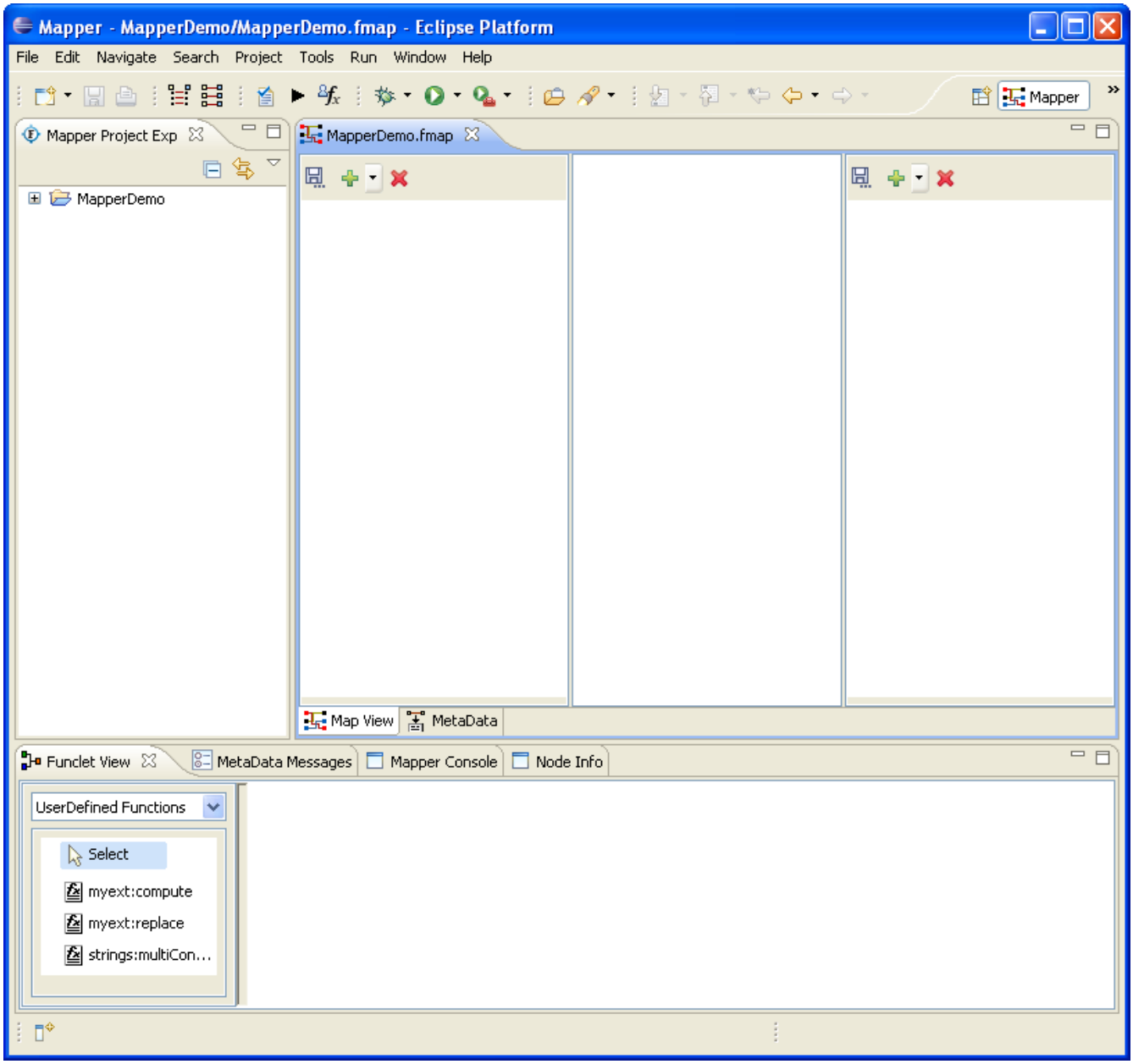


Figure 8.2.1: eMapper Perspective

8.2.1 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 8.2.2 shows a sample eMapper project as shown in the eMapper Projects Explorer.

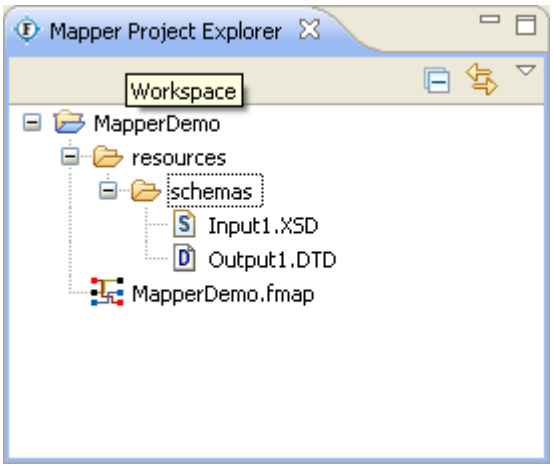


Figure 8.2.2: eMapper Projects

A 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>**.

8.2.2 eMapper Editor

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

8.2.2.1 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 function(s) as shown in Figure 8.2.3.

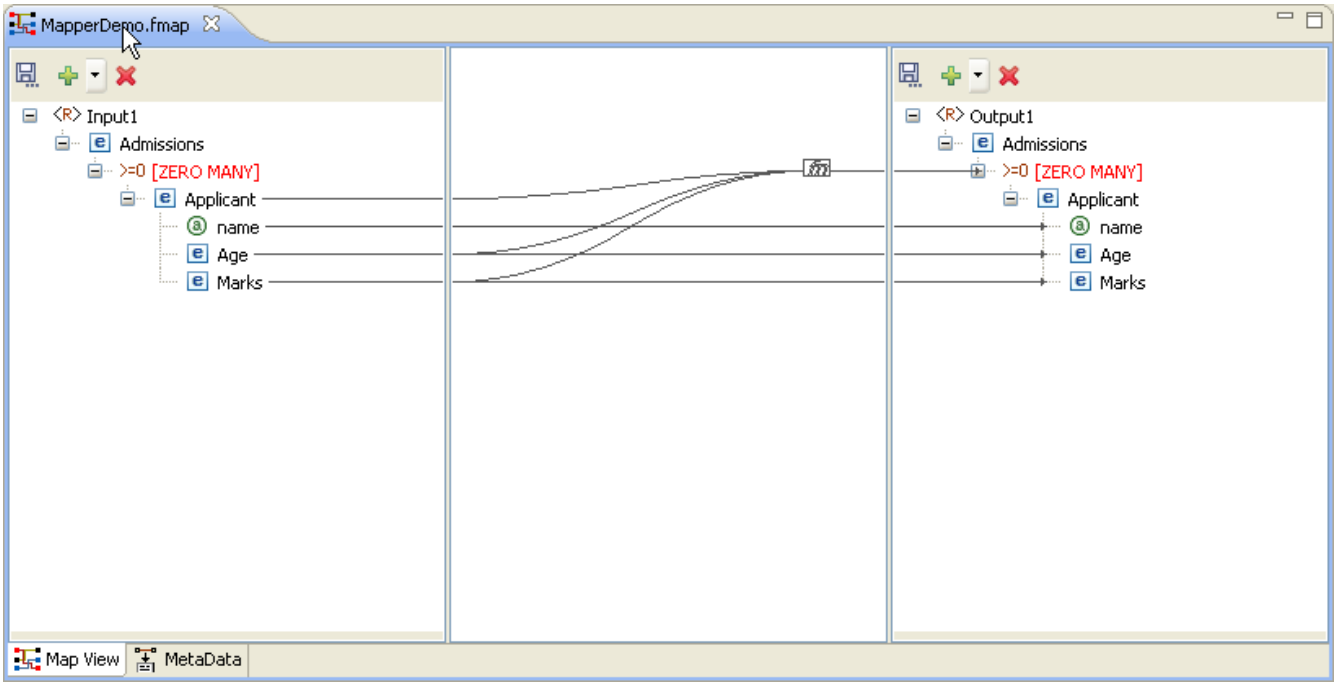


Figure 8.2.3: Map View

Output Structure Panel

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

8.2.2.2 MetaData tab

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

8.2.3 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 8.2.4. It also shows the functions and their linkage 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.

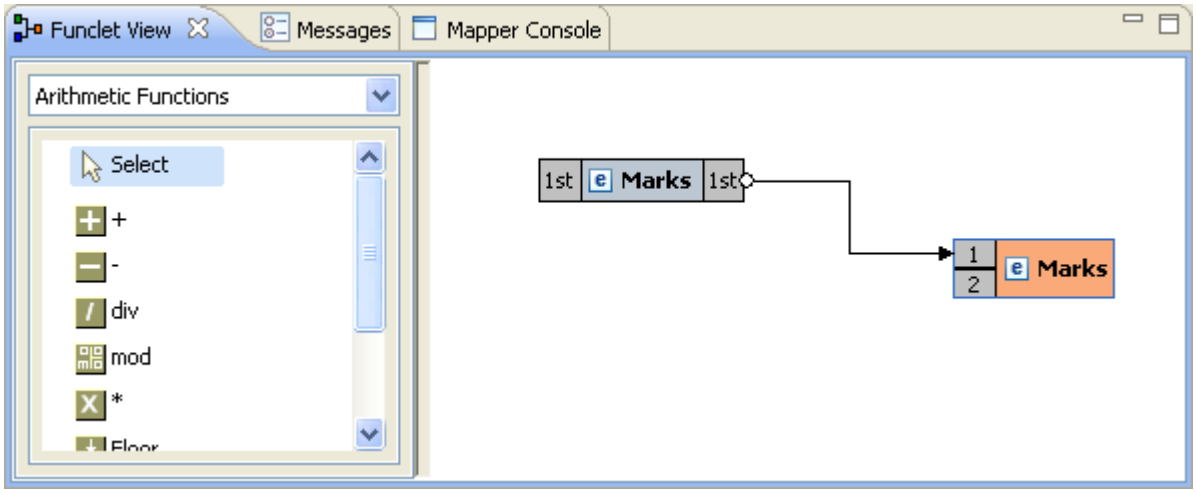


Figure 8.2.4: Funclet VIew

8.2.4 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.

8.2.5 MetaData Messages View

Error or Warning Messages (if any) thrown while generating the transformation XSL are displayed in the MetaData Messages View. The view appears, as shown in Figure 8.2.5.

8.2.6 Node Info View

The Node Info View shows the information about nodes in the Input and Output Structures. The view is shown in Figure 8.2.6. It has two panels that provide the data type and cardinality information about the selected input and output structure node/element.

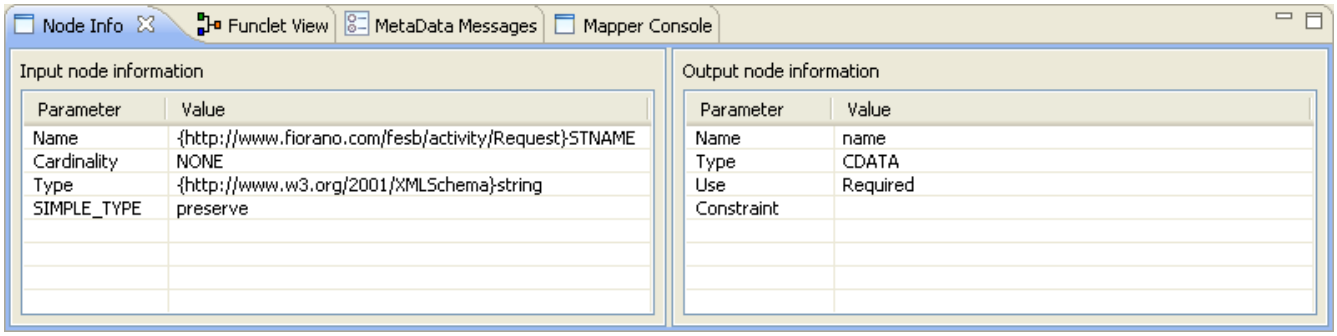


Figure 8.2.6: Node Info View

8.3 Working with Input and Output Structures

8.3.1 Loading Input/Output Structure

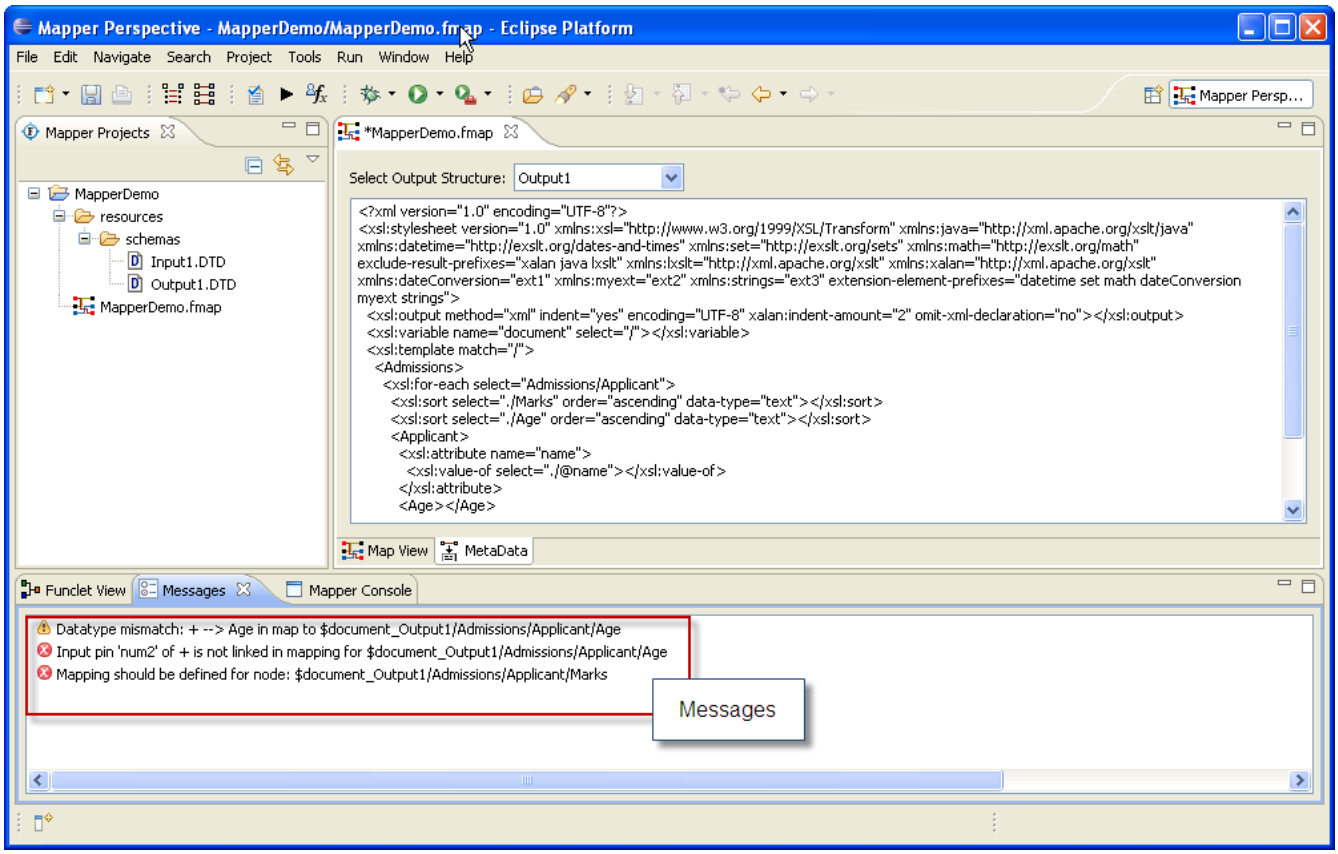


Figure 8.2.5: Meta Data and MetaData Messages view

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

8.3.1.1 Load Input/Output Structure From an XSD document

Select XSD from the **Add Structure** menu. The **Load Input/Output XSD Structure Wizard** would appear as shown in the Figure 8.3.1. The wizard contains two pages, **Structure Selection page** and **External XSDs page**.

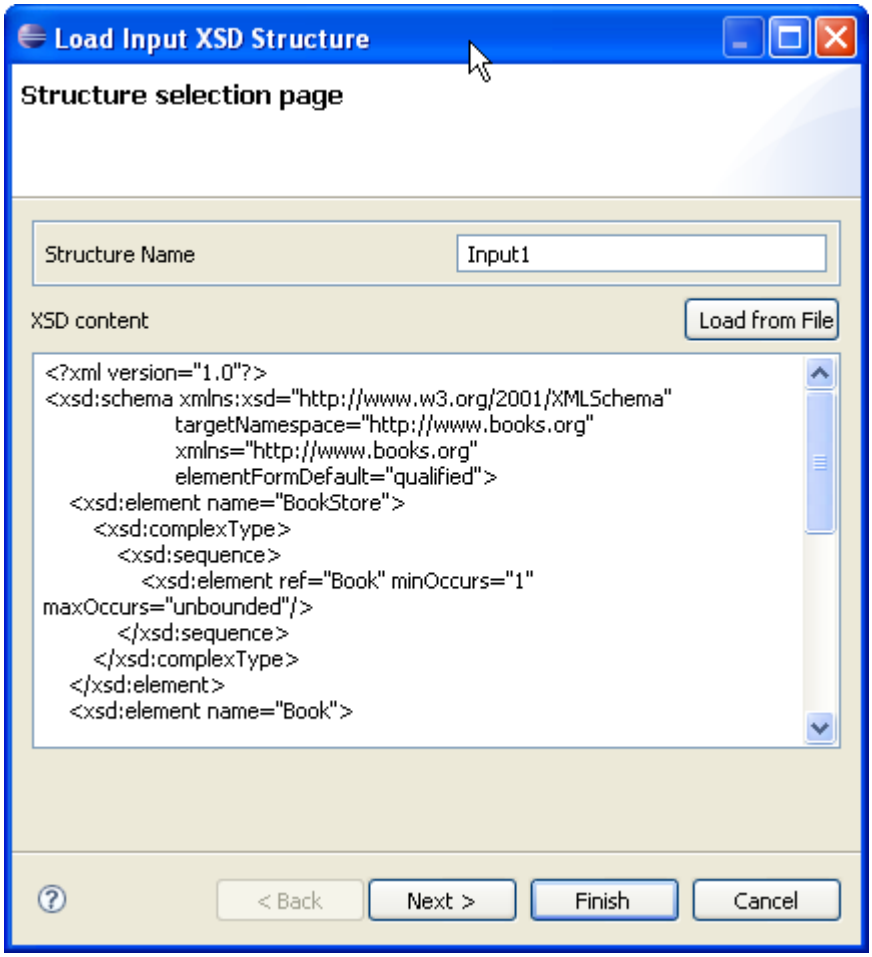


Figure 8.3.1: Structure Selection Page

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 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 using which one can browse through the file system to choose an existing file. Modifications, if any, to the schema loaded from the file from this page.

External XSDs Page

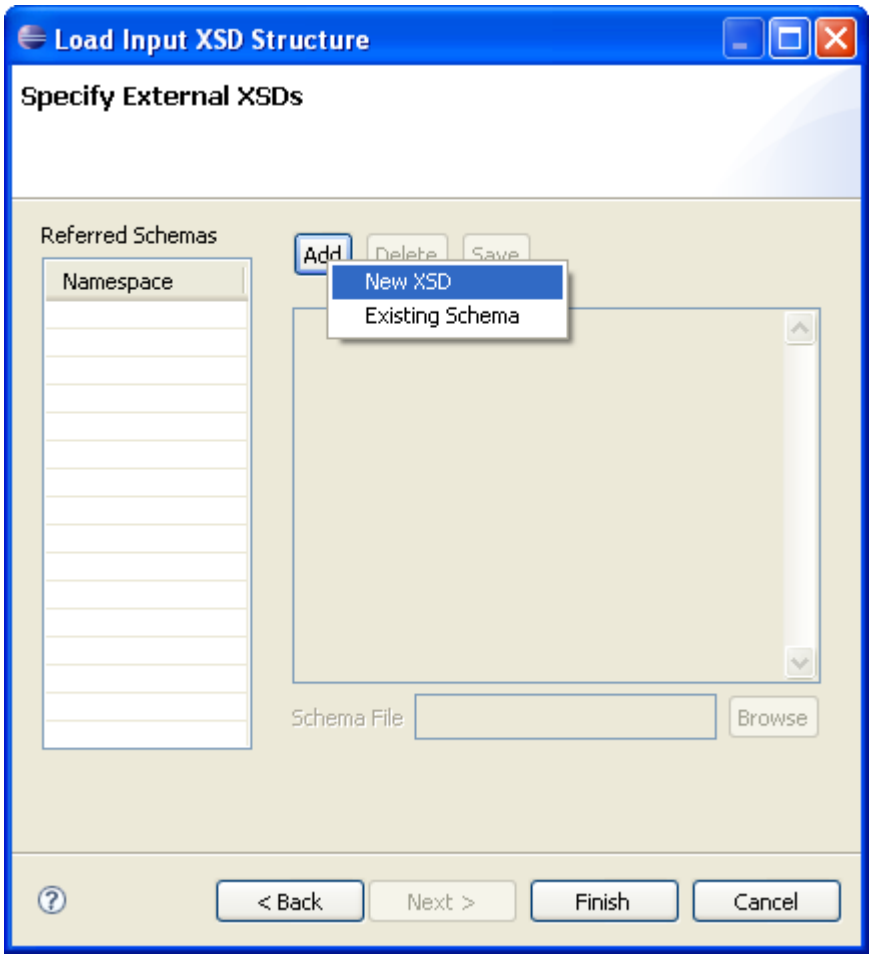


Figure 8.3.2: Adding External XSDs

Any external XSDs used by a structure can be added from this page. Figure 8.3.2 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 8.3.2. 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 shall be saved with this name 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 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 to refer these schemas, 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.

8.3.1.2 Load Input/Output Structure from a DTD document

Select DTD from the **Add Structure** menu. The **Load Input/Output DTD Structure Wizard** would appear as shown in the Figure 8.3.3. 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.

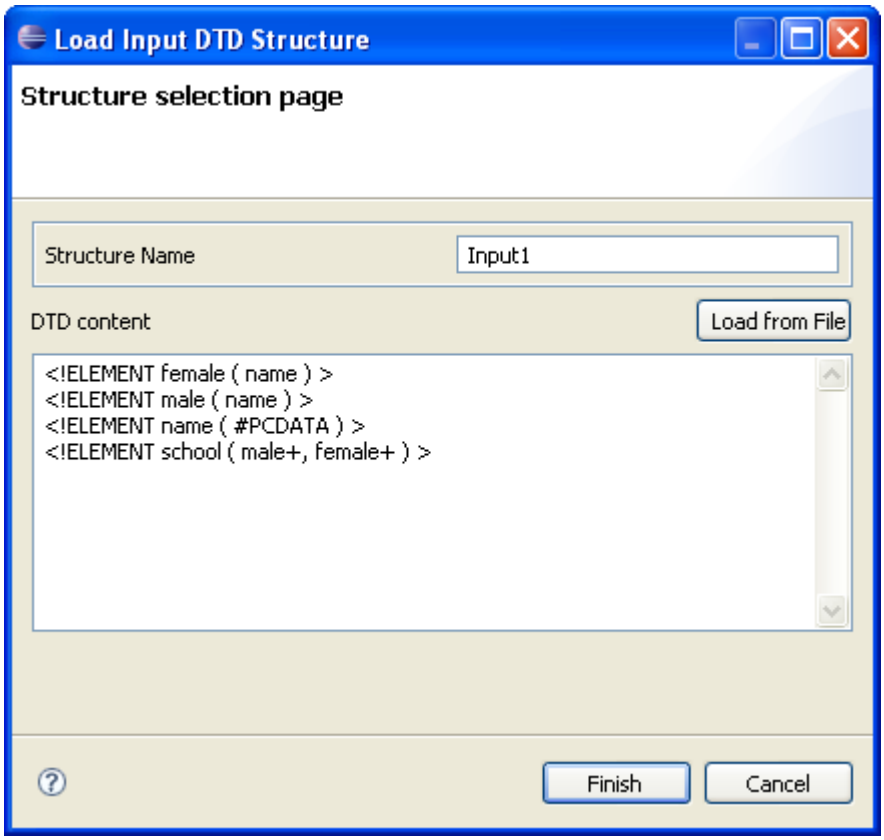


Figure 8.3.3: Load Input DTD Structure

8.3.1.3 Load Input/Output Structure from an XML document

Select XML from the **Add Structure** menu. The **Load Input/Output XML Structure Wizard** would appear as shown in the Figure 8.3.3. The dialog contains two panes viz. **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 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 to the **Error Log** view.

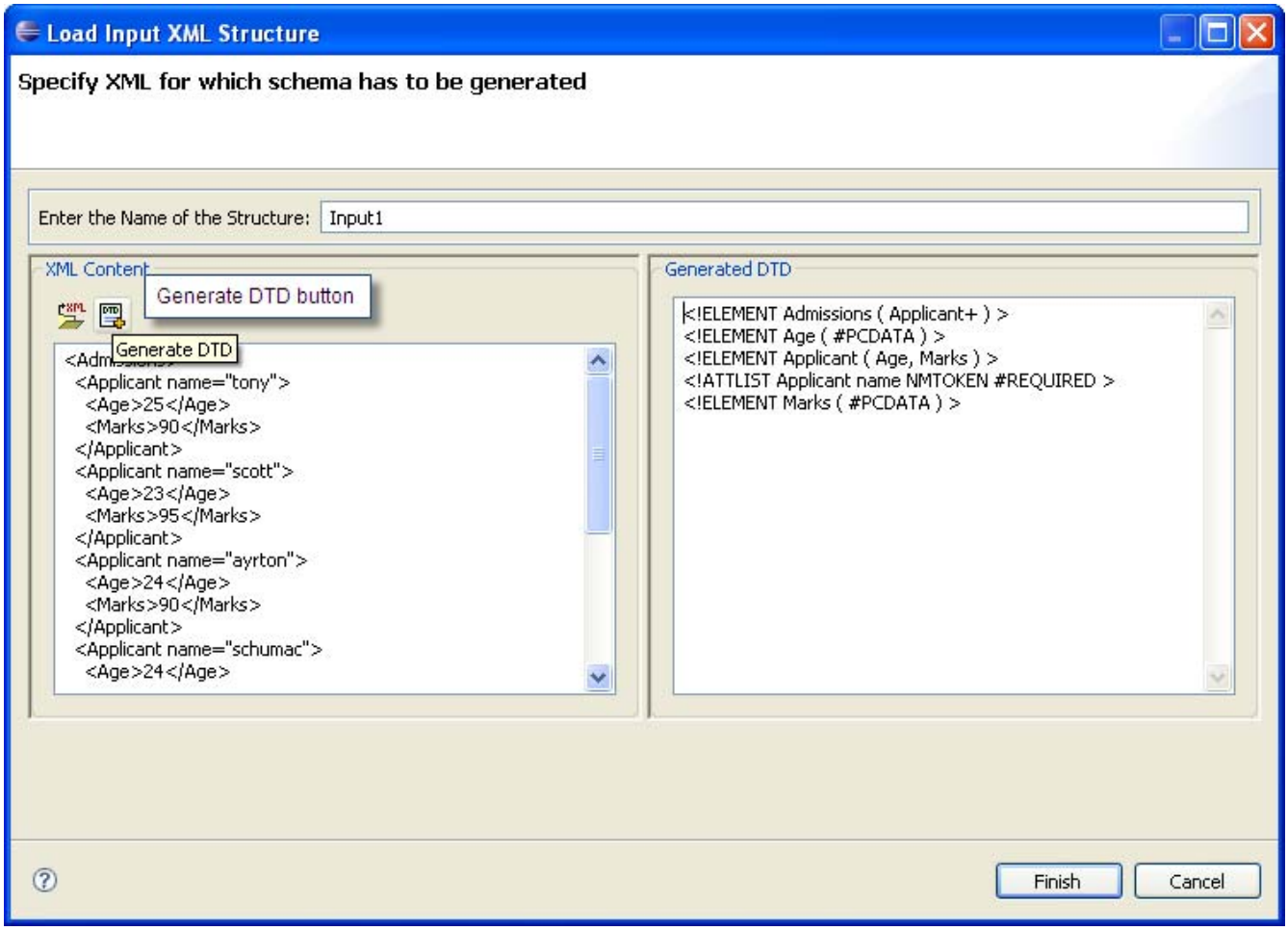


Figure 8.3.4: Load Input XML Structure

8.3.2 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.

8.3.3 Edit Structure

To edit Input/Output Structure:

1. Right-click the structure and click the **Edit Structure** option. The Edit Structure dialog is opened as shown in the Figure 8.3.5.
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 **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 is shown and the modifications are ignored.

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

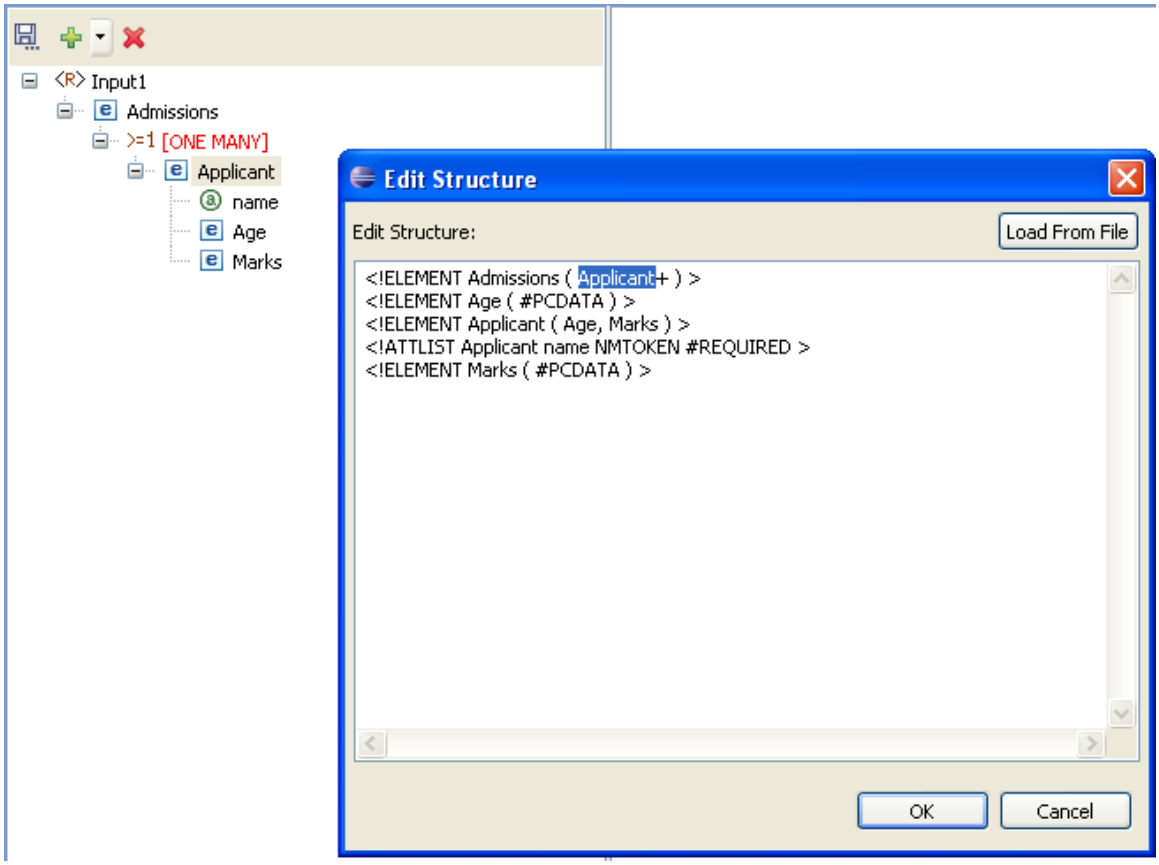


Figure 8.3.5: Edit Structure Dialog

8.4 Working with the Visual Expression Builder

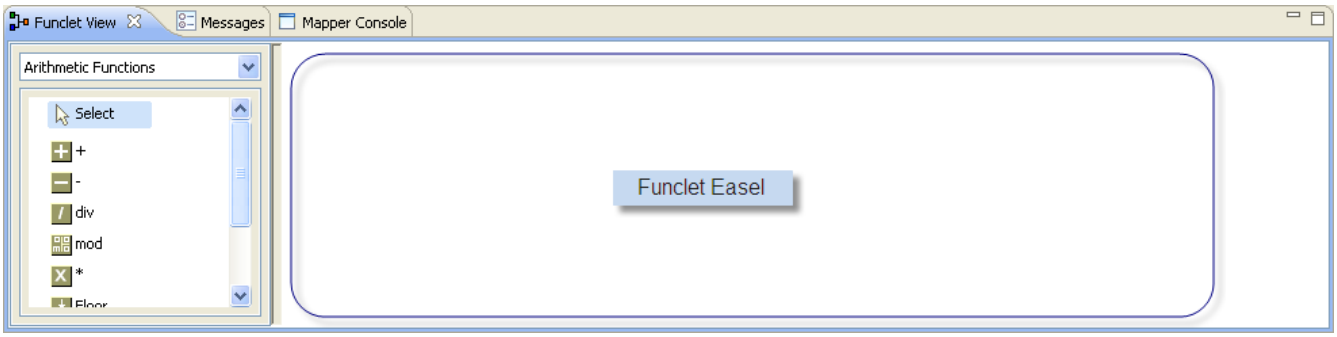
Fiorano eMapper provides an easy to use graphical user interface – the Visual Expression Builder, for building simple or complex expressions using several predefined functions. All this can be done by performing simple drag-n-drop of required functions, input nodes and connecting them visually.

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

The Visual Expression Builder consists of two areas:

Function palette

Funclet easel



Funclet View

8.4.1 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 ceilinged to 26.

Round

Visual representation 

Description: This function rounds off the value of the preceding node or a number to the nearest integer.

Input: A number constant or an input structure node.

Output: Number

Example: The number 4.8 is rounded off to 5 and 4.2 is rounded off to 4.

Number Function

Visual representation 

Description: This function converts the input to a number according to the XPath specifications.

Input: A number constant or an input structure node.

Output: Number based on the following rules:

- Boolean true is converted to 1, and false is converted to 0.
- A node-set is first converted to a string and then converted in the same way as a string argument.

- A string that consists of optional whitespace followed by an optional minus sign followed by a number followed by whitespace is converted to the IEEE 754 number that is nearest to the mathematical value represented by the string; any other string is converted to NaN.
- An object of a type other than the four basic types is converted to a number in a way that is dependent on that type.

8.4.1.2 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.

8.4.1.3 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 a form a single concatenated string.

Input: Two or more string constants or input structure nodes.

Output: A concatenated string.

Example:`Concat ("abc", "xyz") returns "abcxyz".`

Constant

Visual representation 

Description; This function creates a constant building block with a string literal.

Input: String

Output: String

Length

Visual representation 

Description: This function returns the length of a string.

Input: A string constant or an input structure node.

Output: Number

Example: `Length ("abcd") returns 4`

Normalize_Space

Visual representation 

Description: This function accepts a string as an argument and removes leading, trailing, and enclosed spaces in the specified string. The unnecessary white spaces within the string are replaced by a single white space character.

Input: A string or an input structure node.

Output: String with no whitespace before, after, or within it.

Example: Normalize_Space(" 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"`

8.4.1.4 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 node and the second argument is a Boolean value. It filters out and returns the nodes for which the second input value is TRUE.

Input: Filter (node set, bool)

Output: Nodes for which the second input value is true as Loop Source.

8.4.1.5 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

8.4.1.6 Advanced Functions

Fiorano eMapper provides a number of advanced functions. This section explains all these functions.

CDATA Function

Visual representation 

Description: This function accepts a string as an argument and specifies the character data within the string.

Input: String argument or input structure node.

Output: Input string or node text enclosed within the CDATA tag.

Example: `CDATA ("string") returns <![CDATA[ string]]>`

Position

Visual representation 

Description: This function is available for the RDBMS-Update or RDBMS-Delete Output structures only and returns the current looping position.

Input: None

Output: The position of the element in the parent tree.

Example: In an XML tree that has three elements, `Position()` returns

0 for the first element

1 for the second, and

2 for the third.

Format-Number

Visual representation 

Description: This function converts the first argument to a string, in the format specified by the second argument. The first argument can be a real number or an integer, and can be positive or negative.

Input: Two values: The first input is a number, and the second, a string of special characters that specifies the format. These special characters are listed in the following table:

Representation	Signifies	Example
#	a digit [0-9]	###
.	the decimal point	###.##
,	digit separator	###, ###.##
0	leading and trailing zeros	000.0000
%	inserts a percentage sign at the end	###.00%
;	a pattern separator	##.00;##.00

The format string is created by using these characters in any order.

Output: String with the number in the specified format.

Node-Name

Visual representation 

Description: This function accepts an element or attribute and returns the name of the particular element or attribute.

Input: A single element or attribute of any type

Output: A string

Count

Visual representation 

Description: This function accepts an element or attribute and returns the number of instances of a particular element or attribute.

Input: A single element or attribute of any type

Output: A number

Deep-Copy

Visual representation 

Description: Copies the current node completely including the attributes and sub-elements.

Input: An Input structure node

Output: All the contents of the Input structure node – including its attributes and sub-elements.

Param

Visual representation 

Description: This function is used to access the runtime parameters by its name. Various properties of Tifosi Document (such as header, message, and attachments) are available as runtime parameters at runtime. The names of these parameters follow the convention given below:

Header Properties	_TIF_HEADER_<HEADERNAME>
Message (text)	_TIF_BODY_TEXT_
Message (byte)	_TIF_BODY_BYTE_
Attachment	_TIF_ATTACH_<NAME>

Input: Name of the parameter

Output: Value of the parameter specified

8.4.1.7 Date-Time Functions

Date-Time functions include:

Date

Visual representation 

Description: The Date function returns the date part in the input date-time string or the current date if no input is given. The date returned format is: CCYY-MM-DD

If no argument is given or the argument date/time specifies a time zone, then the date string format must include a time zone, either a Z to indicate Coordinated Universal Time or a + or - followed by the difference between the difference from UTC represented as hh:mm. If an argument is specified and it does not specify a time zone, then the date string format must not include a time zone.

Input: Optionally, a string that can be converted to a date (the string should have the date specified in the following format: CCYY-MM-DD)

Output: A date in the format: CCYY-MM-DD

DateTime

Visual representation 

Description: This function returns the current date and time as a date/time string in the following format:

CCYY-MM-DDThh:mm:ss

Where,

CC is the century

YY is the year of the century

MM is the month in two digits

DD is the day of the month in two digits

T is the separator between the Date and Time part of the string

hh is the hour of the day in 24-hour format

mm is the minutes of the hour

ss is the seconds of the minute

The output format includes a time zone, either a Z to indicate Coordinated Universal Time or a + or - followed by the difference between the localtime from UTC represented as hh:mm.

Input: This function has no input.

Output: The current date-time in the following format: CCYY-MM-DDThh:mm:ss as described above.

DayAbbreviation

Visual representation 

Description: This function returns the abbreviated day of the week from the input date string. If no argument is given, then the current local date/time is used as the default argument.

Input: Optionally, a date-time string

Output: The English day of the week as a three-letter abbreviation: 'Sun', 'Mon', "Tue', 'Wed', 'Thu', 'Fri', or 'Sat'.

DayInMonth

Visual representation 

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

Input: A date-time string in any of the following formats:

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

If no input is given, then the current local date/time is used.

Output: A number which is the day of the month in the input string.

DayInWeek

Visual representation 

Description: This function returns the day of the week given in a date as a number. If no argument is given, then the current local date/time is used the default argument.

Input: A date string in any of the following formats:

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

Output: The day of the week as a number - starting with 1 for Sunday, 2 for Monday and so on up to 7 for Saturday. If the date/time input string is not in a valid format, then NaN is returned.

DayInYear

Visual representation 

Description: This function returns the day of a date as a day number in a year starting from 1.

If no argument is given, then the current local date/time, as returned by date-time is used the default argument.

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

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

Output: A number representing the day in a year.

Example: The DayInYear for 2003-01-01 returns 1, where as for 2003-02-01 it returns 32.

DayName

Visual representation 

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

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

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

Output: An English day name: 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday' or 'Friday'.

DayOfWeekInMonth

Visual representation 

Description: This function returns the occurrence of that day of the week in a month for a given date as a number. If no argument is given, then the current local date/time is used as the default argument.

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

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

Output: A number that represents the occurrence of that day-of-the-week in a month.

Example: DayOfWeekInMonth returns 3 for the 3rd Tuesday in May.

HourInDay

Visual representation 

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

Input: A date string in any one of the following formats:

CCYY-MM-DDThh:mm:ss

hh:mm:ss

If the date/time string is not in one of these formats, then NaN is returned.

Output: The hour of the day or NaN if the argument is not valid.

LeapYear

Visual representation 

Description: This function returns TRUE if the year given in a date is a leap year. If no argument is given, then the current local date/time is used as the default argument.

Input: Date string in any of the following formats:

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

If the date/time string is not in one of these formats, then NaN is returned.

Output: Boolean value (TRUE/FALSE)

MinuteInHour

Visual representation 

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

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

CCYY-MM-DDThh:mm:ss
hh:mm:ss

Output: The minute of the hour or NaN if the argument is not valid.

MonthAbbreviation

Visual representation 

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

Input: Date string in any of the following formats:

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

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

MonthInYear

Visual representation 

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

Input: Date string in any of the following formats:

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

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

Output: A number representing the month in a year.

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

MonthName

Visual representation 

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

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

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

OutputThe English month name: 'January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November' or 'December'.If the date/time string is not valid, then an empty string ('') is returned.

SecondInMinute

Visual representation 

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

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

```
CCYY-MM-DDThh:mm:ss
hh:mm:ss
```

Output: The second in a minute as a number. If the date/time string is not valid, then NaN is returned.

Time

Visual representation 

Description: This function returns the time specified in the date/time string that is passed as an argument. If no argument is given, the current local date/time is used as the default argument. The date/time format is basically CCYY-MM-DDThh:mm:ss.

If no argument is given or the argument date/time specifies a time zone, then the time string format must include a time zone, either a Z to indicate Coordinated Universal Time or a + or - followed by the difference between the difference from UTC represented as hh:mm. If an argument is specified and it does not specify a time zone, then the time string format must not include a time zone.

Input: Optionally, a date/time string in the following format:

CCYY-MM-DDThh:mm:ss

Output: The time from the given date/time string in the following format:

hh:mm:ss

If the argument string is not in this format, this function returns an empty string ('').

WeekInYear

Visual representation 

Description: This function returns the week of the year as a number. If no argument is given, then the current local date/time is used as the default argument. Counting follows ISO 8601 standards for numbering: week 1 in a year is the week containing the first Thursday of the year, with new weeks beginning on a Monday.

Input: Optionally, a date/time string in any of the following format:

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

Output: The week of the year as a number. If the date/time string is not in one of these formats, then NaN is returned.

Year

Visual representation 

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

Input: Optionally, a date/time string in any of the following format:

CCYY-MM-DDThh:mm:ss

CCYY-MM-DD
CCYY-MM
CCYY

Output If the date/time string is not in one of these formats, then NaN is returned.

8.4.1.8 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 non-numeric value, then NaN is returned.

Input: A node set

Output: A numerical value

MAXIMUM

Visual representation 

Description: The maximum function returns the node with the maximum numerical value within the given node set. If the node set is empty, or if any of the nodes in the node set has non-numeric value, then NaN is returned.

Input: A node set

Output: A numerical value

8.4.1.9 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 example of Chat.dtd as Input Structure and Valid.dtd as Output Structure. Suppose we want to filter out mails that do not have message body and the email address is not equal to admin@nobody.com. That is, we want that the **isValid** node of the Output Structure takes the value true if the length of the **Message** node of the Input Structure is not equal to zero and the value of the **Email** node is equal to admin@nobody.com**.** Therefore,

1. Load **Input Structure** and **Output Structure**.
2. Map the **Message** node and **Email** node of Input Structure to the **isValid** node of the Output Structure.
3. Invoke the Function Wizard by right clicking on the **isValid** node.
4. The Function Easel opens with the existing mappings.
5. Place the **BOOL** node on the Function Easel.

   Link the output of the **Message** node to the input of the **BOOL** node, as shown in Figure 8.4.1.

Figure 8.4.1: 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 8.4.2.

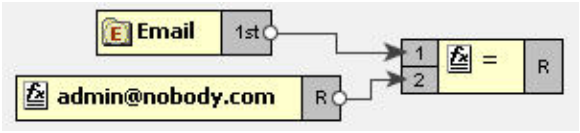


Figure 8.4.2: Linking the Email and Constant node outputs

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 8.4.3.

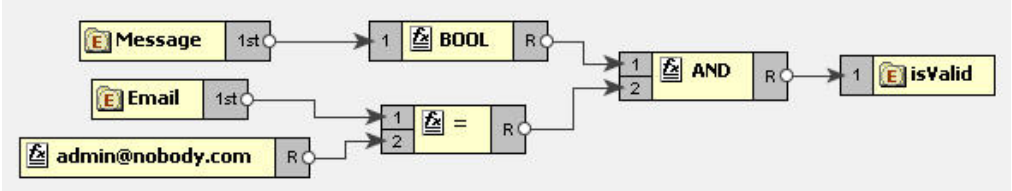


Figure 8.4.3: Linking the AND and = node outputs

10. This completes the desired mappings.

BOOL

Symbol: BOOL

Description:

This function converts its argument to a boolean according to the XPath specifications which are as follows:

- A number is TRUEif and only if it is neither positive or negative zero nor NaN.

- A node-set is TRUE if and only if it is non-empty.

- A string is TRUE if and only if its length is non-zero.

- An object of a type other than the four basic types is converted to a boolean in a way that is dependent on that type.

Input: BOOL (Object)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of Chat.dtd as Input Structure and Valid.dtd as Output Structure. Suppose we want to filter out mails that do not have message body. That is, we want that the **isMessageExist** node of the Output Structure takes the value true if the length of the Message node of the Input Structure is not equal to zero. The **BOOL** function returns true for a string of length non-zero. Therefore,

1. Load **Input Structure** and **Output Structure**.
2. Map the **Message** node of Input Structure to the **isMessageExist** node of the Output Structure.
3. Invoke the Function Wizard by right clicking on the **isMessageExist** node.
4. The Function Easel opens with the existing mappings.
5. Place the **BOOL** node on the Function Easel.

   Link the output of the **Message** node to the input of the **BOOL** node, as shown in Figure 8.4.4.

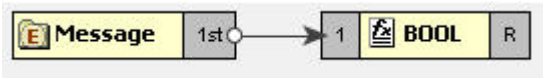


Figure 8.4.4: Linking Message and BOOl nodes

   Link the output of the **BOOL** node to the input of the **isMessageExist** node, as shown in Figure 8.4.5.

Figure 8.4.5: Linking BOOL and IsMessageExist nodes

6. This completes the desired mappings.

Equal

Symbol: =

Description: This function returns TRUE if both the inputs are equal.

Input: = (Object = Object)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of Chat.dtd as Input Structure and Valid.dtd as Output Structure. Suppose we want to filter mails coming from a particular email address. That is, we want that the **isFromAdmin** node of the Output Structure takes the value true if the **Email** node of the Input Structure has the email address as **admin@nobody.com**. Then,

1. Load **Input Structure** and **Output Structure**.
2. Map the Email node of Input Structure to the **isFromAdmin** node of the Output Structure.
3. Invoke the Function Wizard by right clicking on the **isFromAdmin** node.
4. The Function Easel opens with the existing mappings.

   Now place a **Constant** building block on the Function Easel and set its value equal to admin@nobody.com, as shown in Figure 8.4.6.

Figure 8.4.6: 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 8.4.7.

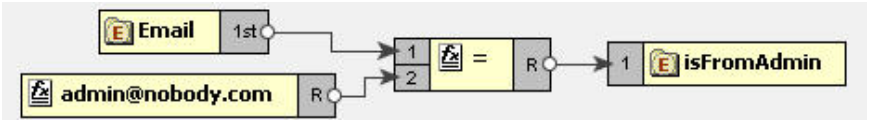


Figure 8.4.7: Linking = and isFromAdmin node

7. This completes the desired mappings.

Less Than

Symbol: <

Description: This function returns TRUE if the first input is less than the second input value.

Input: < (Number < Number)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of Numbers.dtd as Input Structure and Results.dtd as Output Structure. Suppose we want that **Result** node of Output Structure should have the value true if the value of **Number1** node is less than the value of the **Number2** node of the Input Structure. Then,

1. Load **Input Structure** and **Output Structure**.
2. Map the **Number1** and **Number2** nodes of Input Structure to the **Result** node of the Output Structure.
3. Invoke the Function Wizard by right clicking on the **Result** node.
4. The Function Easel shows the existing mappings.

   Place the < node on the Function Easel, as shown in Figure 8.4.8.

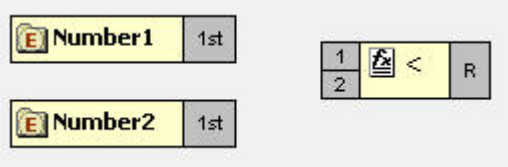


Figure 8.4.8: Placing a node on the Function Easel

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 8.4.8.

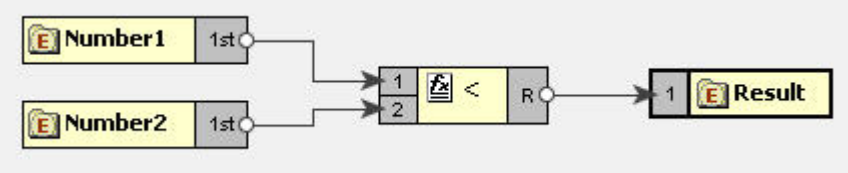


Figure 8.4.9: Linking < and Result node

6. This completes the desired mappings.

Greater than

Symbol: >

Description: This function returns TRUE if the first input is greater than the second input value.

Input: > (Number > Number)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of TotalMarks.dtd as Input Structure and Result.dtd as Output Structure. Suppose we want that the value of the PassStatus node is true if the value of the TotalMarks node of the Input Structure is greater than a constant value 150. Then,

1. Load **Input Structure** and **Output Structure**.
2. Map the **TotalMarks** node of Input Structure to the **PassStatus** node of the Output Structure.
3. Invoke the Function Wizard by right-clicking on the **PassStatus** node.
4. The Function Easel opens with the existing mappings.
5. Now place a **Constant** building block on the Function Easel and set its value equal to 150, as shown in Figure 8.4.10.

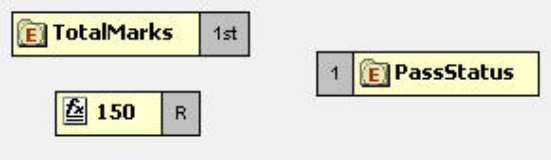


Figure 8.4.10: Setting the Constant building block to 150

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 8.4.11.

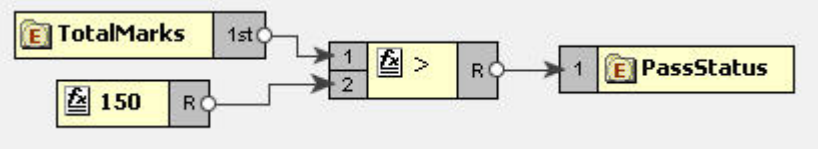


Figure 8.4.11: Linking the > and PassStatus node

8. This completes the desired mappings.

Greater than or Equal function

Function: >=

Input: >= (Number >= Number)

Description: True if the first input is greater than or equal to the second input.
Output: True/False

Example:

Consider example of TotalMarks.dtd as Input Structure and Result.dtd as Output Structure. Suppose we want that the value of the **PassStatus** node is true if the value of the **TotalMarks** node of the Input Structure is greater than or equal to a constant value **150**.Then,

1. Load **Input Structure** and **Output Structure**.
2. Map the **TotalMarks** node of Input Structure to the **PassStatus** node of the Output Structure.
3. Invoke the Function Wizard by right-clicking on the **PassStatus** node.
4. The Function Easel opens with the existing mappings.

   Now place a **Constant** building block on the Function Easel and set its value equal to 150, as shown in Figure 8.4.12.

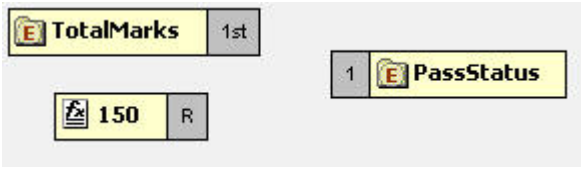


Figure 8.4.12: Setting the Constant building block to 150

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 8.4.13.

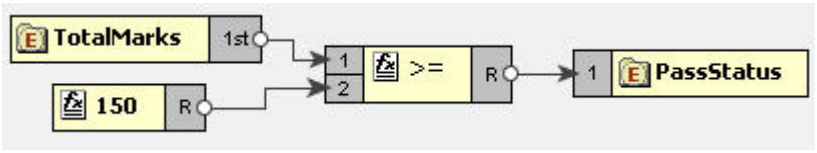


Figure 8.4.13: Linking the >= and PassStatus nodes

7. This completes the desired mappings.

OR

Symbol: OR

Description: This function accepts two boolean expressions as arguments and performs logical disjunction on them. If either expression evaluates to TRUE, the function returns TRUE. If neither expression evaluates to True, the function returns FALSE.

Input: OR (boolean OR boolean)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of Chat.dtd as Input Structure and Valid.dtdas Output Structure. Suppose we want to receive mails that are sent either from the address admin@nobody.com or aryton@nobody.com that is, we want that the **isValid** node of the Output Structure takes the value true if the **Email** node of the Input Structure has the value admin@nobody.com or aryton@nobody.com. Then,

1. Load **Input Structure** and **Output Structure**.
2. Map the **Email** node of Input Structure to the **isValid** node of the Output Structure.
3. Invoke the Function Wizard by right clicking on the **isValid** node.
4. The Function Easel opens with the existing mappings.
5. Place a **Constant** node on the Function Easel and set its value equal to admin@nobody.com.

   Place another **Constant** node and set its value equal to aryton@nobody.com, as shown in Figure 8.4.14.

Figure 8.4.14: Setting the Constant node value to aryton@nobody.com

Now place two = nodes on the Function Easel, and make links as shown in Figure 8.4.15.

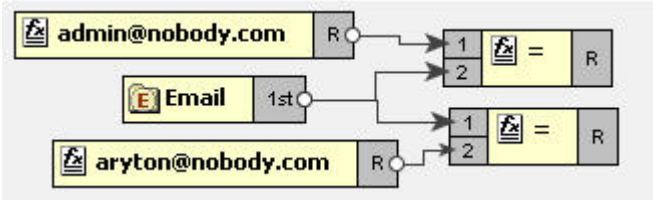


Figure 8.4.15: Placing two = nodes on the Function Easel

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 8.4.16.

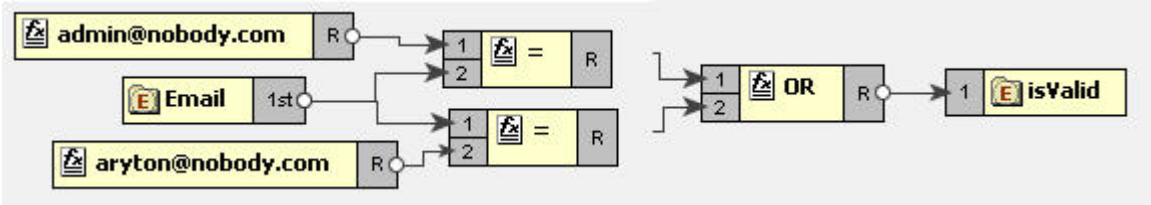


Figure 8.4.16: Linking the OR and isValid nodes

8. This completes the desired mappings.

Less Than or Equal

Symbol: <=

Description: This function returns TRUE if the first input is less than or equal to the second input.

Input: <= (Number <= Number)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of Numbers.dtd as Input Structure and Results.dtd as Output Structure. Suppose we want that **Result** node of Output Structure should have the value true if the value of **Number1** node is less than or equal to the value of the **Number2** node of the Input Structure. Then,

1. Load **Input Structure** and **Output Structure**.
2. Map the **Number1** and **Number2** nodes of Input Structure to the **Result** node of the Output Structure.
3. Invoke the Function Wizard by right-clicking on the **Result** node.
4. The Function Easel shows the existing mappings.

   Place the <= node on the Function Easel, as shown in Figure 8.4.17:

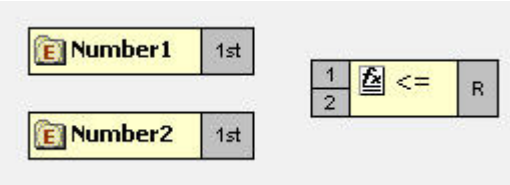


Figure 8.4.17: Placing <= node on the Function Easel

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 8.4.18:

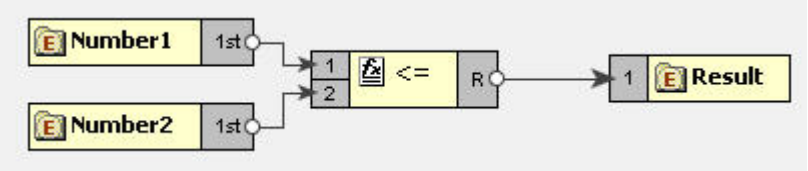


Figure 8.4.18: Linking outputs of the Number1 and Number2 nodes

6. This completes the desired mappings.

NOT

Symbol: NOT

Description: This function accepts a boolean expression as the argument and performs logical negation the expression. The result is a boolean value representing whether the expression is FALSE. That is, if the expression is FALSE, the result of this function is TRUE.

Input: NOT (boolean)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of Valid.dtd as Input and Output Structure. Suppose we want to make mails from email address admin@nobody.com as invalid. That is, we want that if the value of **isFromAdmin** node is true, then the value of **isValid** is set to false. Then,

1. Load **Input Structure** and **Output Structure**.
2. Map the **isFromAdmin** node of Input Structure to the **isValid** node of the Output Structure.
3. Invoke the Function Wizard by right-clicking on the **isValid** node.
4. The Function Easel shows the existing mappings.

   Now place a **NOT** node on the Function Easel, as shown in Figure 8.4.18.

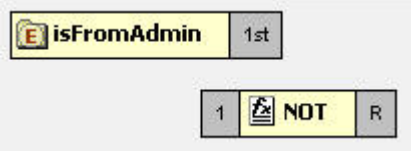


Figure 8.4.19: placing a NOT node on the Function Easel

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 8.4.20.

Figure 8.4.20: Linking theNOT and isValid nodes

6. This completes the desired mappings.

Not Equal

Symbol ! =

Description: This function returns TRUE if both the inputs are not equal.

Input != (Object = Object)

Output: Boolean value (TRUE/FALSE)

Example:

Consider example of Chat.dtd as Input Structure and Valid.dtd as Output Structure. Suppose we want to filter out mails that do not have message body. That is, we want that the **isMessageExist** node of the Output Structure takes the value true if the length of the Message node of the Input Structure is not equal to zero. Then,

1. Load **Input Structure** and **Output Structure**.
2. Map the **Message** node of Input Structure to the **isMessageExist** node of the Output Structure.
3. Invoke the Function Wizard by right clicking on the **isMessageExist** node.
4. The Function Easel opens with the existing mappings.
5. Now place a **Constant** building block on the Function Easel and set its value equal to 0.
6. Place a **Length** node on the Function Easel.

   Link the output of the **Message** node to the input of the **Length** node, as shown in Figure 8.4.21.

Figure 8.4.21: 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 8.4.22.

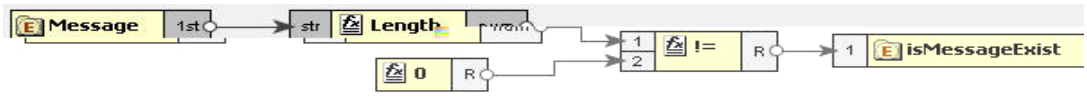


Figure 8.4.22: Linking the != and isMessageExist nodes

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)

8.4.1.10 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.

8.4.1.10.1 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 value of multiple columns. To get multiple columns, use the format <**column3**>,<**column4**>.
   For example: dvValueDA, dvDescription

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

4. Make sure the input value match the column length defined in the database. For example, if the dvSendCode is defined as char(10) in the database, the input value should be BLK followed by 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 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 8.4.23: 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>

8.4.1.10.2 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**.

8.4.1.11 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

8.4.1.11.1 Byte Content

Function: Byte Content

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

Output: Base64 encoded string value

8.4.1.11.2 Text Content

Function: Text Content

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

Output: String value

8.4.1.11.3 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

8.4.1.11.4 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

8.4.1.12 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

8.4.1.12.1 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
```

8.4.1.12.2 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

8.4.1.12.3 myExt:compute

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

Input: A valid javascript expression

Output: A number

8.4.1.12.4 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

8.4.2 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 8.4.24.

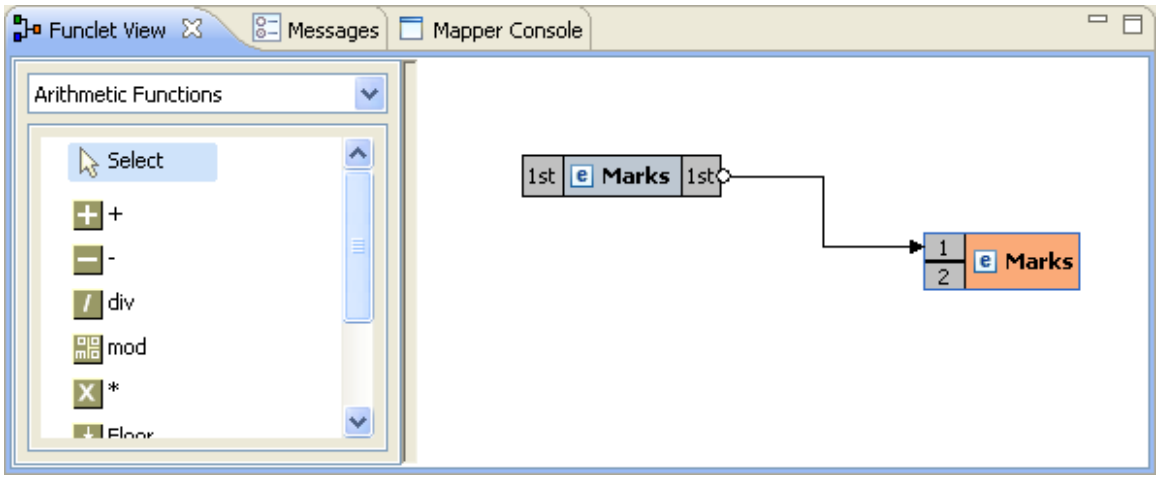


Figure 8.4.24: Funclet easel

8.4.2.1 Source Node

The Source node corresponds to a node in the Input Structure Panel. A Source node appears, as shown in Figure 8.4.25.

Figure 8.4.25: Source Node

8.4.2.2 Destination Node

The Destination node corresponds to a node in the Output Structure Panel. A Destination node appears, as shown in Figure 8.4.26.

Figure 8.4.26: 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 8.4.27. This represents the starting point of the link and the output box of the building block.

   Figure 8.4.27: 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 8.4.28.

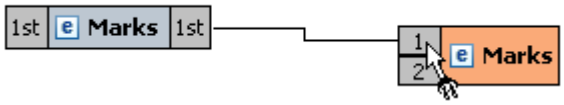
   

   Figure 8.4.28: Linking the Source and the Destination node

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 8.4.28.

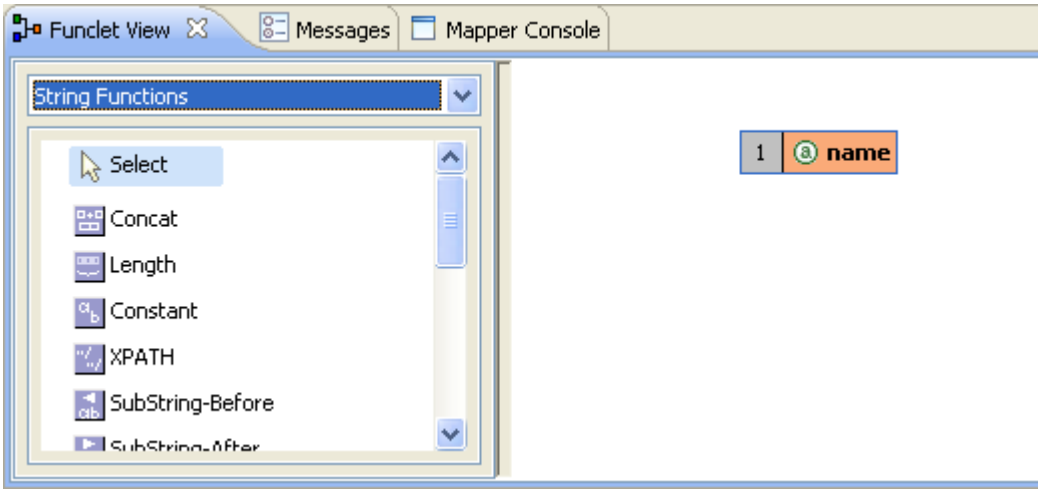


Figure 8.4.29: Adding Source node to Funclet easel

Add Function node to Funclet easel

1. Click the Function node on the Function palette that is to be placed on the Funclet easel, as shown in Figure 8.4.30.

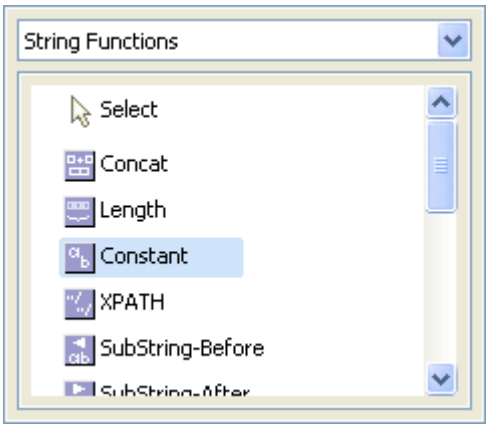


Figure 8.4.30: Selecting the Function node

2. Now move mouse into the Funclet easel. This changes the mouse to a?+? '+' sign, representing that the corresponding function node is selected.
3. Now click on the Funclet easel.
4. This places the corresponding function node building block on the Funclet easel.

Alternatively,

1. Drag-and-Drop the function node from Function palette to the Funclet easel, as shown in Figure 8.4.31.

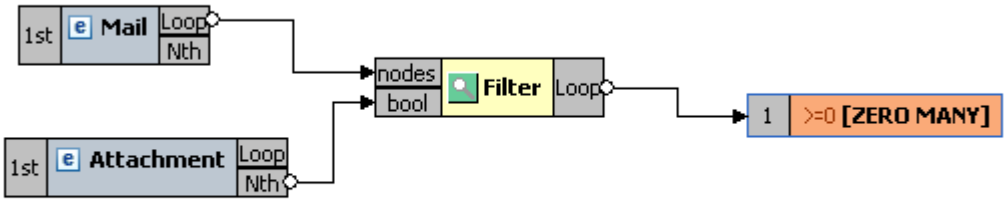


Figure 8.4.31: Funclet easel

Add Link between two nodes

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

1. Click on the gray box on the source building block. A small circle appears, as shown in Figure 8.4.32. This represents the starting point of the link and the output box of the building block.

Figure 8.4.32: Source node

2. Now drag-and-drop the mouse to the destination node's input point, which is again represented by a gray box. A big circle appears on the destination node, as shown in Figure 8.4.33.

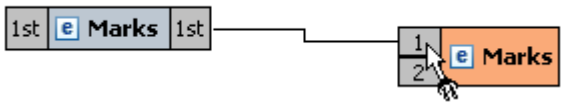


Figure 8.4.33: Linking the Source and the destination node

3. Release the mouse. A link between the two nodes is created, as shown in Figure 8.4.34.

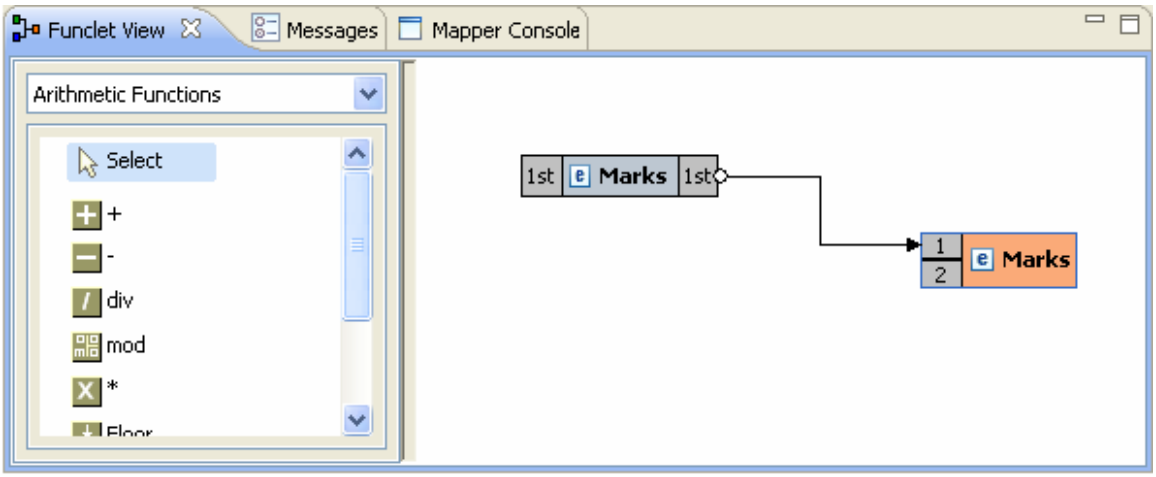


Figure 8.4.34: Linking Source and Destination nodes

Delete link between two nodes

To delete a link between two building blocks,

1. Click on the pointing arrow (ending point) of the link and drag it to an empty area in the Funclet easel, as shown in Figure 8.4.35.

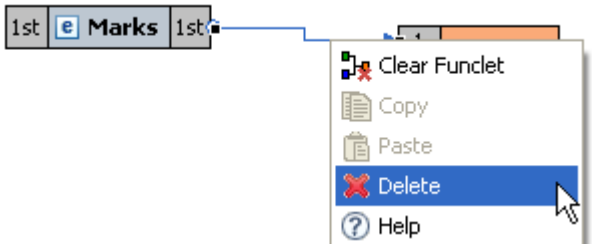


Figure 8.4.35: Deleting link

2. 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 8.4.36.

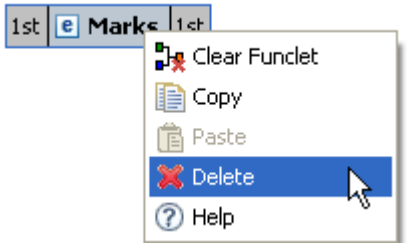


Figure 8.4.36: Pop-up menu

- 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.

8.5 Creating Mappings

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

8.5.1 Understanding Types of Nodes

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

1. **Element Node:** This type of node contains an XML element.
2. **Text Node:** This type of node contains an XML element only.
3. **Attribute Node:** This type of node contains an attribute of the XML element that contains it.

4. **Control Node:** The control node is a pseudo node that depicts the cardinality of the elements in an XML structure. The Control node is displayed in red color, and is surrounded by square brackets.

The control node serves as a useful indicator while creating mappings between the Input and Output Structures. For example, an Output structure node that has a cardinality of one or more requires that at least one element should be added to that XML structure.

1. **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 8.5.1 the Mail-List element can contain zero or many Mail nodes.

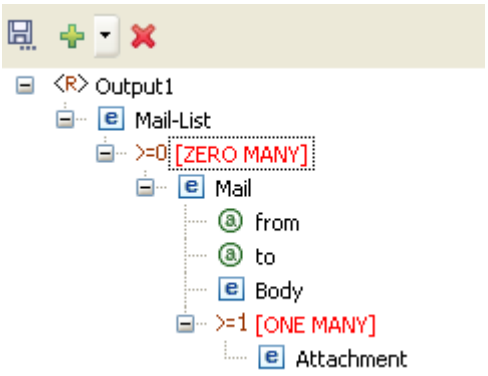


Figure 8.5.1: Example of Zero to Many control node

2. **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 8.5.2 the Mail node can contain one or many occurrences of the Attachment node.

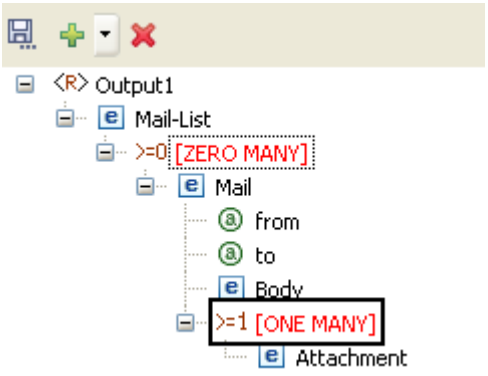


Figure 8.5.2: Example of One to Many control node

3. **Control Node [Choice]:** This Control node specifies that only one of the descendant nodes can exist in the parent node. For example in Figure 8.5.3 TifosiService node can have either Java node or Win32 node, but not both.

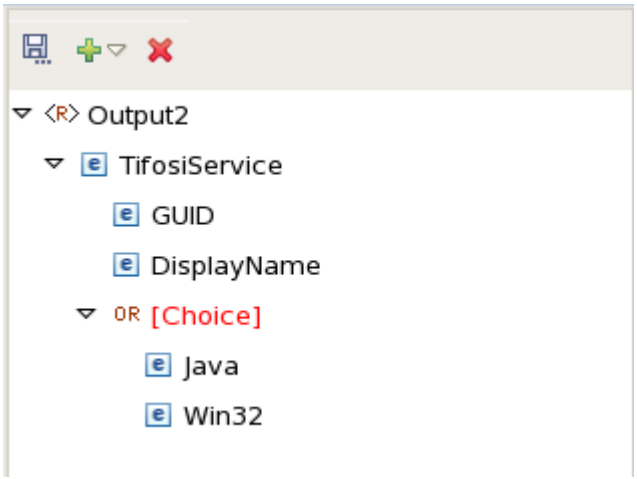


Figure 8.5.3: Example of Choice Control Node

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 8.5.4, Student node can have zero or on occurrence of the Nick-Name node. An Optional node (element/attribute) is displayed in green color.

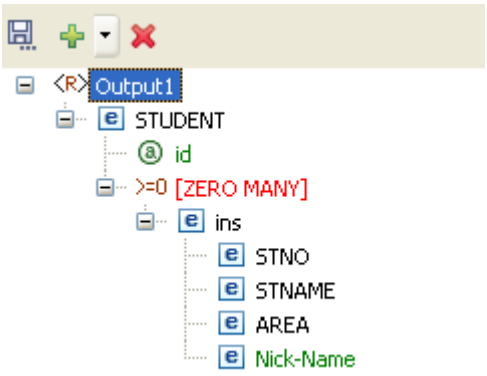


Figure 8.5.4: Example of Optional Node

8.5.2 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.

8.5.2.1 Name-to-Name Mapping

Now create mapping from Name-to-Name, as shown in Figure 8.5.6.

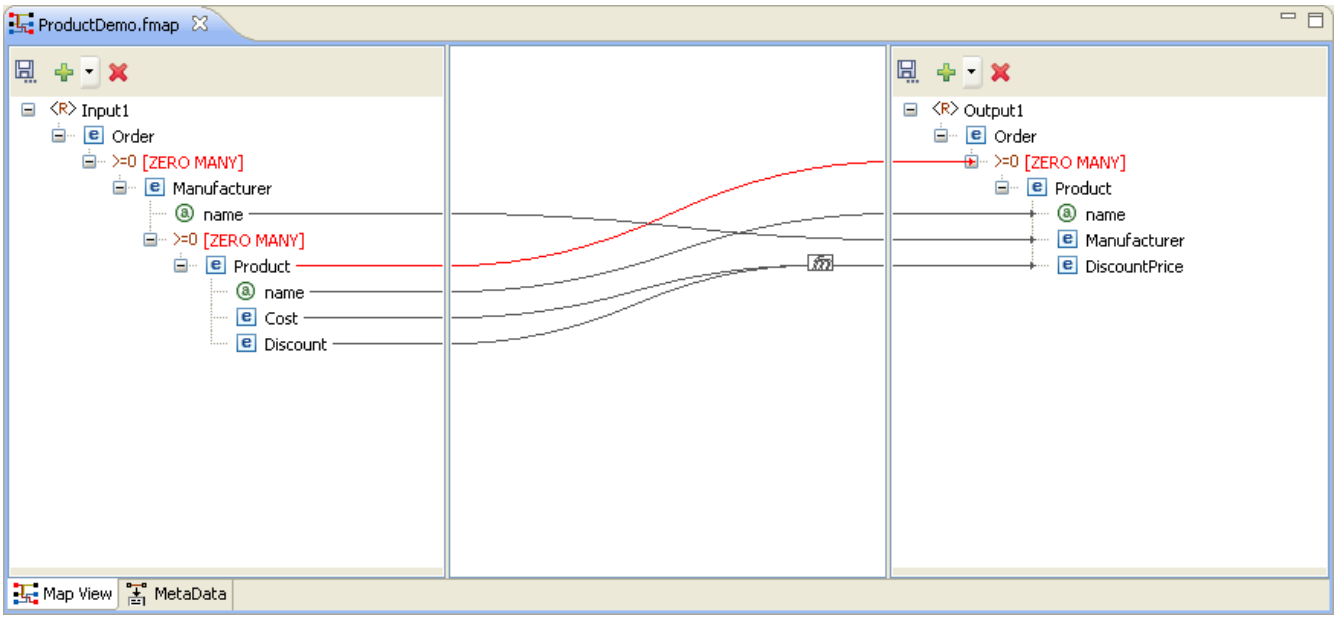


Figure 8.5.6: Name-to-name Mapping

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.

8.5.2.2 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 8.5.7, shows an instance of a For-Each mapping.

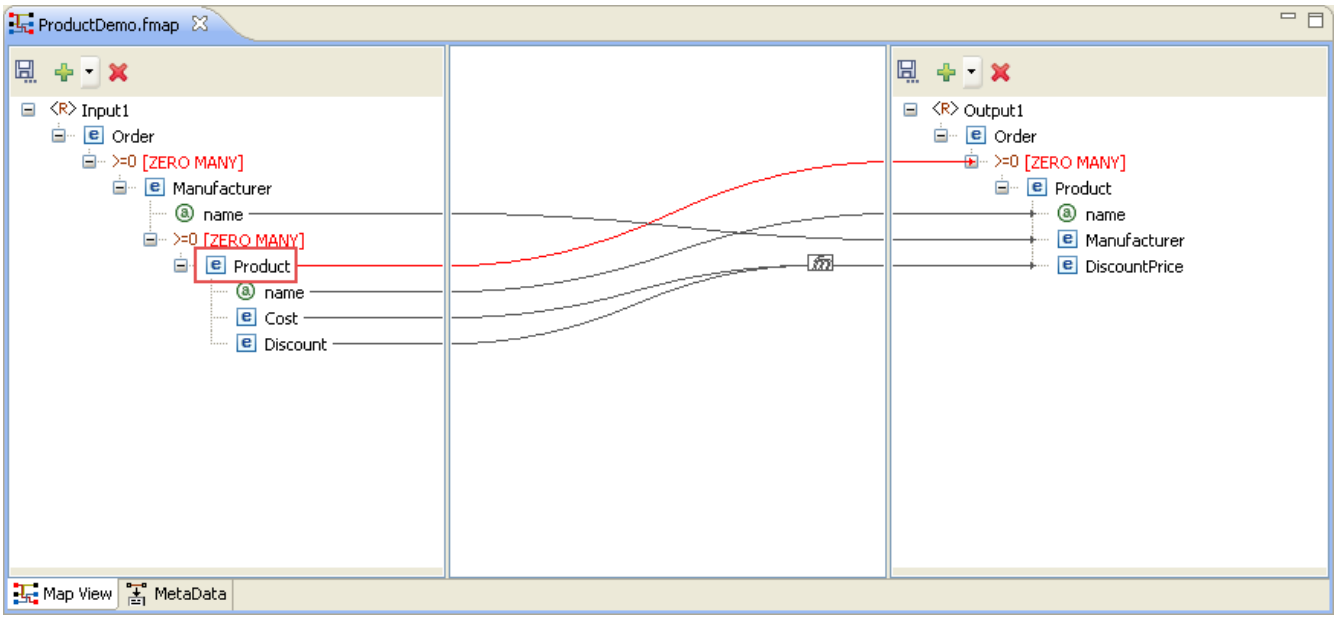


Figure 8.5.7: For-Each Mapping

This mapping specifies that for each Product element in the input XML, the output XML contains a Product element. For-each mapping can be applied only to [ZERO-MANY] or [ONE-MANY] control nodes in the Output Structure.

To create a For-each mapping in the Funclet Wizard, you need to link the Loop output label of the Input Structure node to a [ZERO MANY] or [ONE-MANY] control node in the Output Structure. These control nodes signify the cardinality of contained elements and attributes.

All value mappings for the attributes and child elements of a [ZERO MANY] or [ONE-MANY] node with For-Each mapping, are carried out within this For-Each context.

So, in Figure 8.5.7 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.

8.5.3 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 student element in the output structure DTD, as shown in Figure 8.5.8.

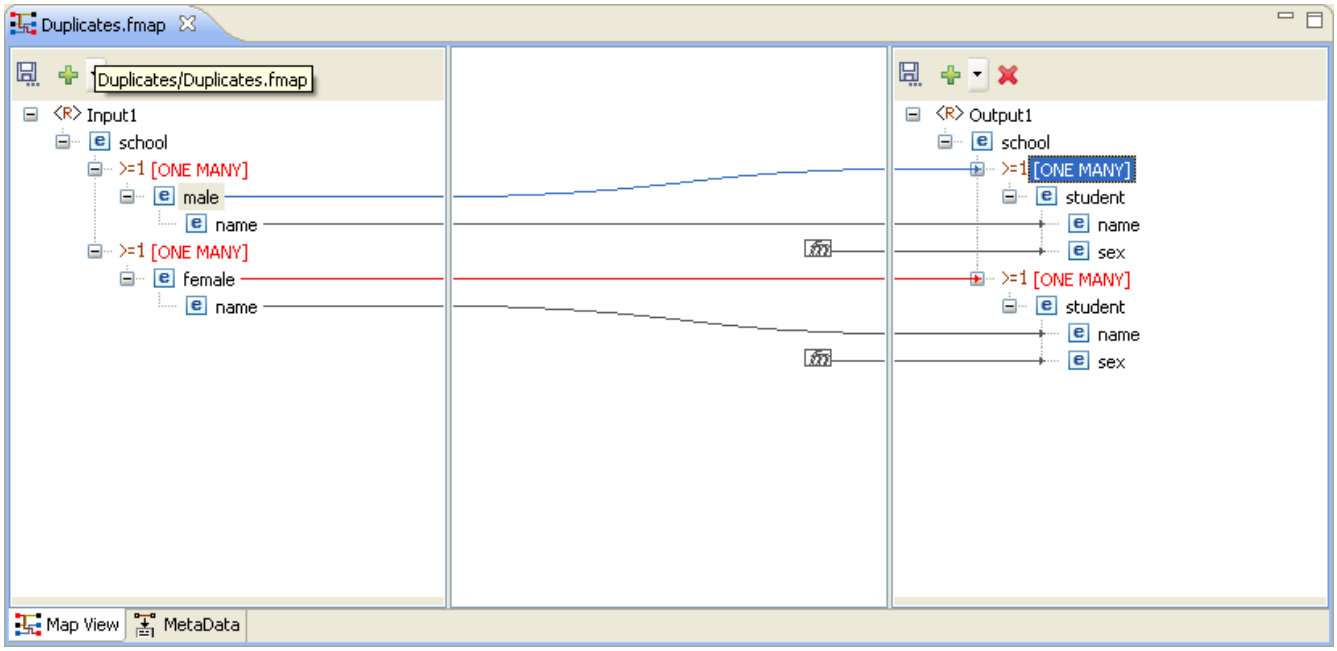


Figure 8.5.8: Mapping a node to One Many control node

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 is displayed as shown in Figure 8.5.8.

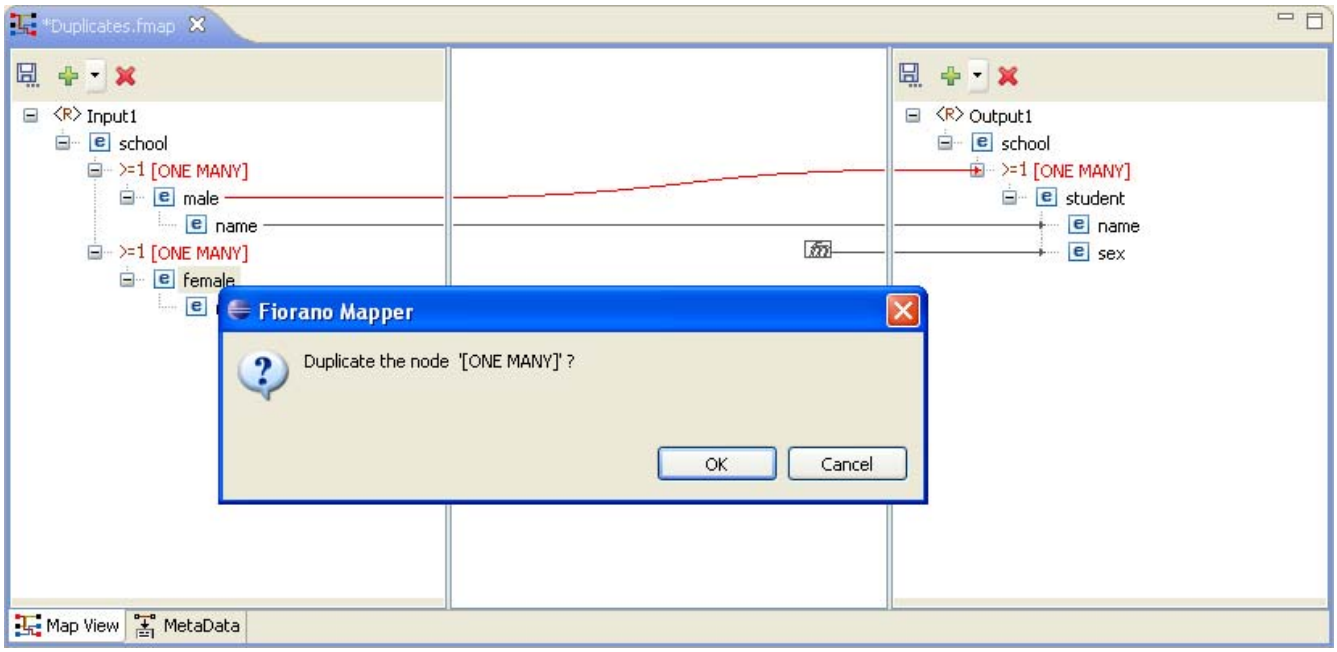


Figure 8.5.9: A shortcut menu prompts you to duplicate the node

Click OK in the message dialog to create a duplicate node. A mapping is created as shown in Figure 8.5.10.

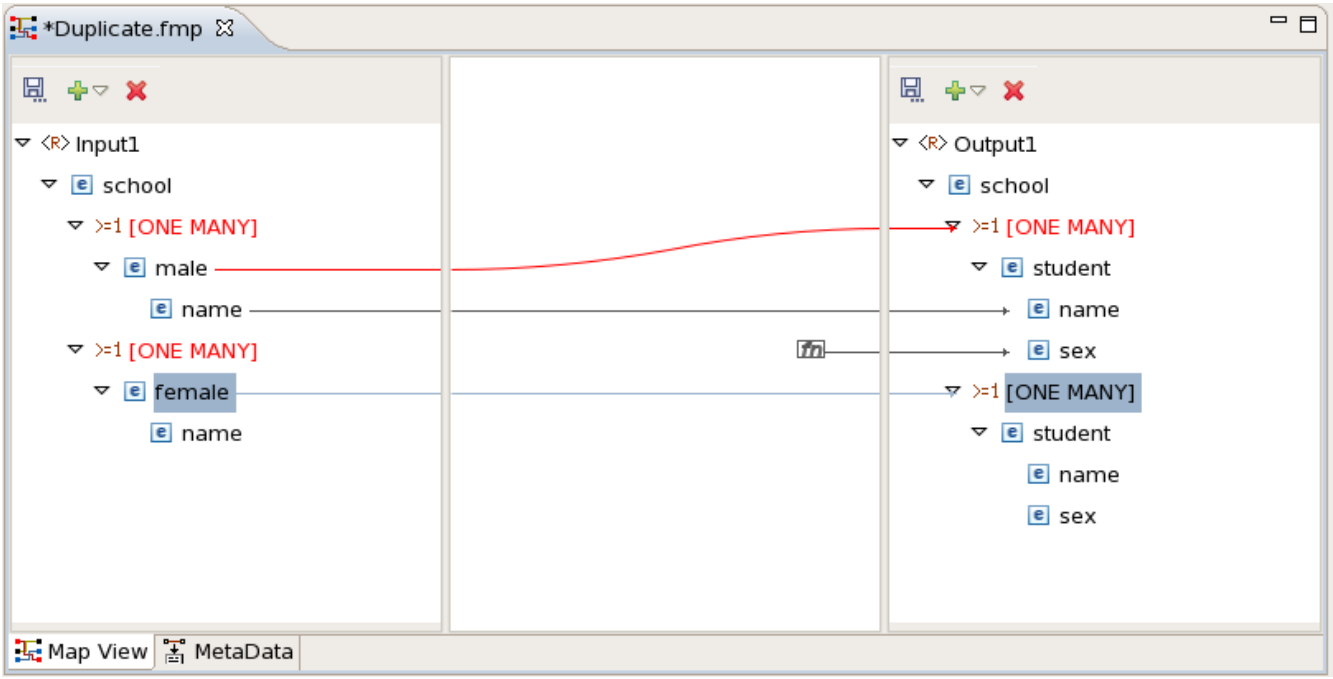


Figure 8.5.10: The One Many Node is Duplicated

8.5.4 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

8.5.4.1 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 8.5.11.

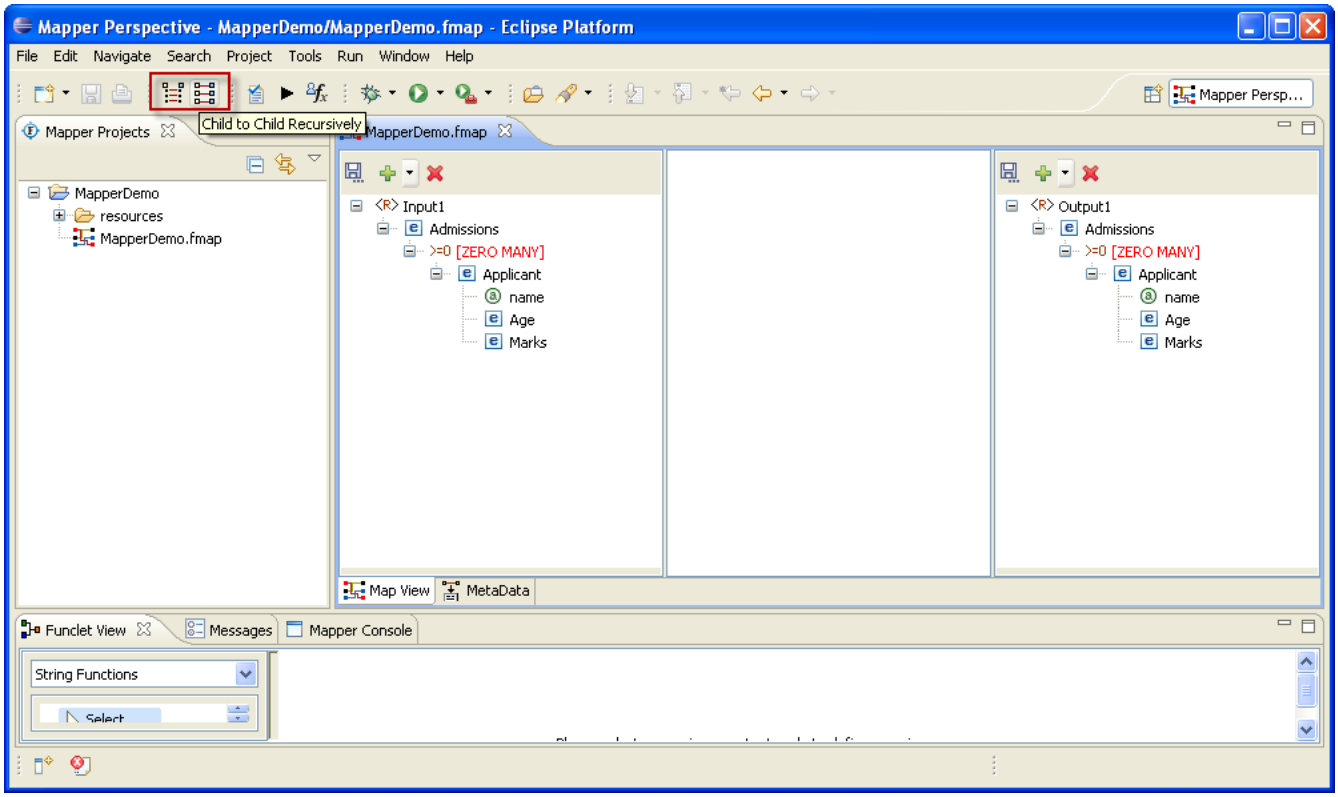


Figure 8.5.11: Creating Automatic Mapping between child nodes

8.5.4.2 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 and the Funclet View of the eMapper Perspective is displayed as shown in Figure 8.5.12.

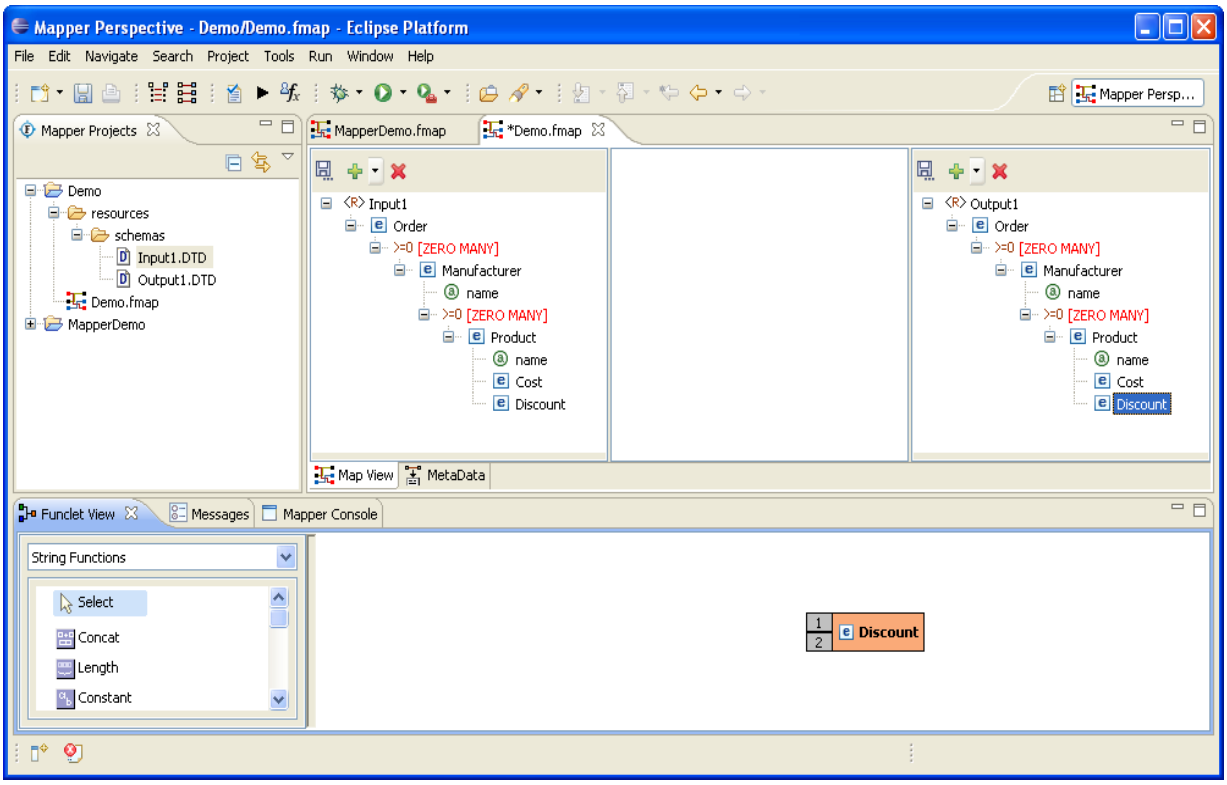


Figure 8.5.12: Selecting the Output Node for Mapping

2. The selected Output node is automatically displayed in the Function easel, as shown in the Figure 8.5.12. 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 8.5.13.

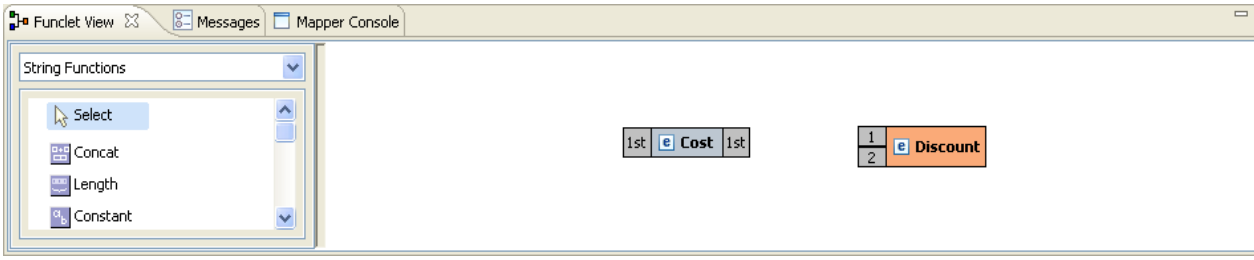


Figure 8.5.13: Dragging an Input node

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 8.5.14.

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 8.5.15.

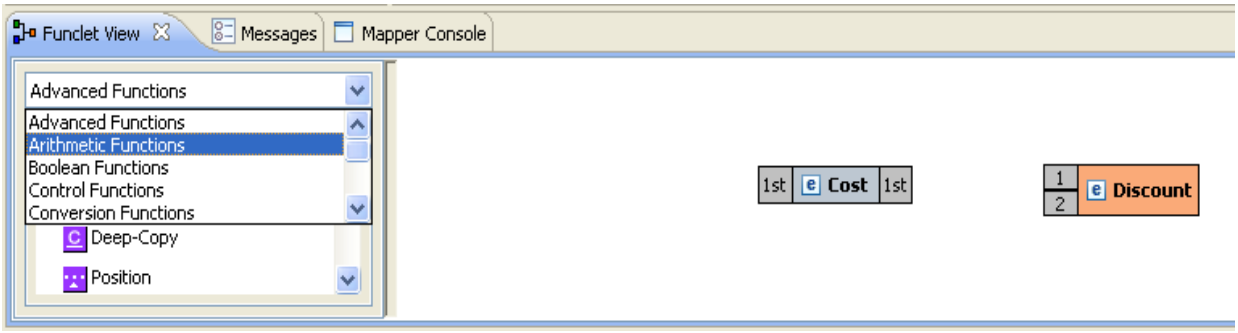


Figure 8.5.14: Selecting the Arithmetic Function Category in the Funclet palette

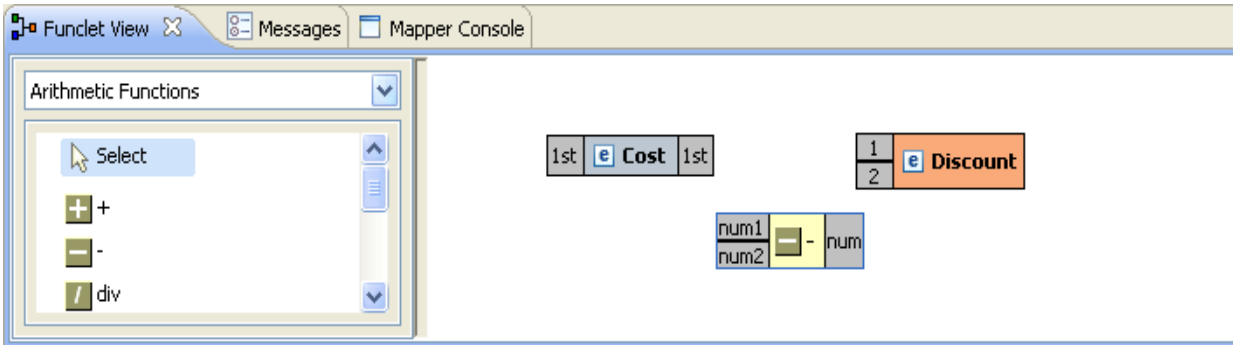


Figure 8.5.15: Adding the Subtract function

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

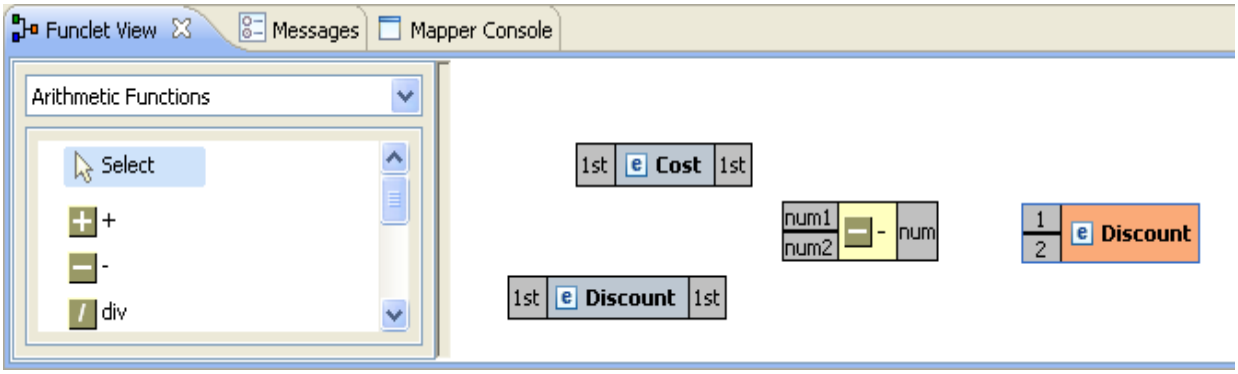


Figure 8.5.16: Adding another input node

6. To define a mapping links should be defined between these node. 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.

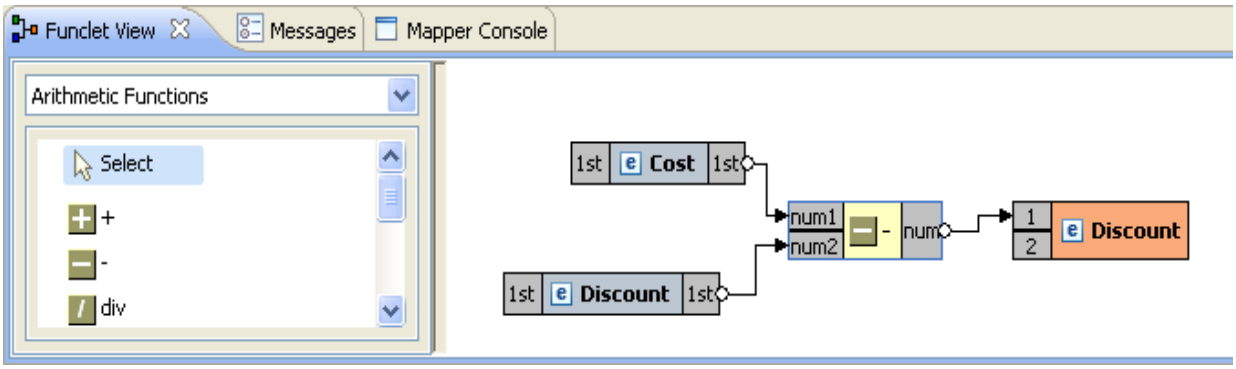


Figure 8.5.19: The final mapping is defined

7. The required mapping is defined as shown in Figure 8.5.18.

8.5.5 Mapping XML Formats

Mapping one XML format to another is a common requirement. The steps for mapping XML, formats to each other are as follows:

1. Load the XML, DTD, or XSD input structure or structures.
2. Load an XML, DTD, or XSD output structure.
3. Link the Input XML Structure node or nodes to the Output XML Structure node.

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

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

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

8.6 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

1. Right-click the <unmapped element> in the output structure and select the **User XSL** option from the shortcut menu as shown in Figure 8.6.1.

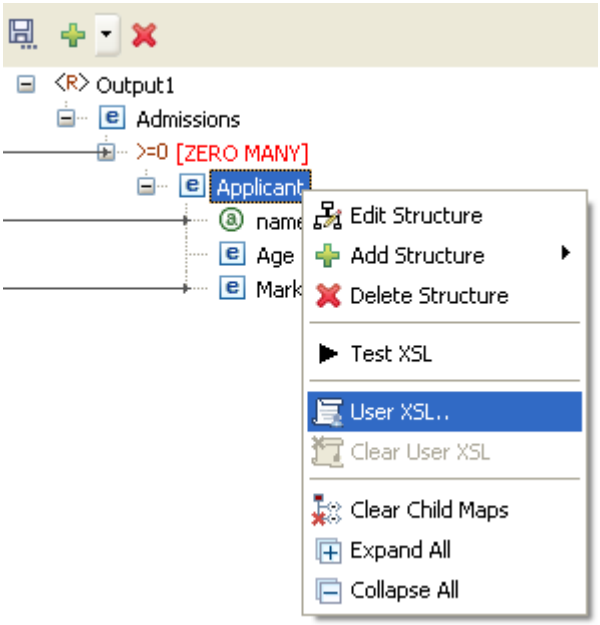


Figure 8.6.1: Selecting the User XSL option from shortcut menu

2. A dialog box appears which contains the xslt script. The xslt script displayed in this dialog box is partially editable. The editable regions, as shown in Figure 8.6.2, are marked by comments *<!--User code starts here-->* and *<!--User code ends here-->* at the beginning and ending respectively.

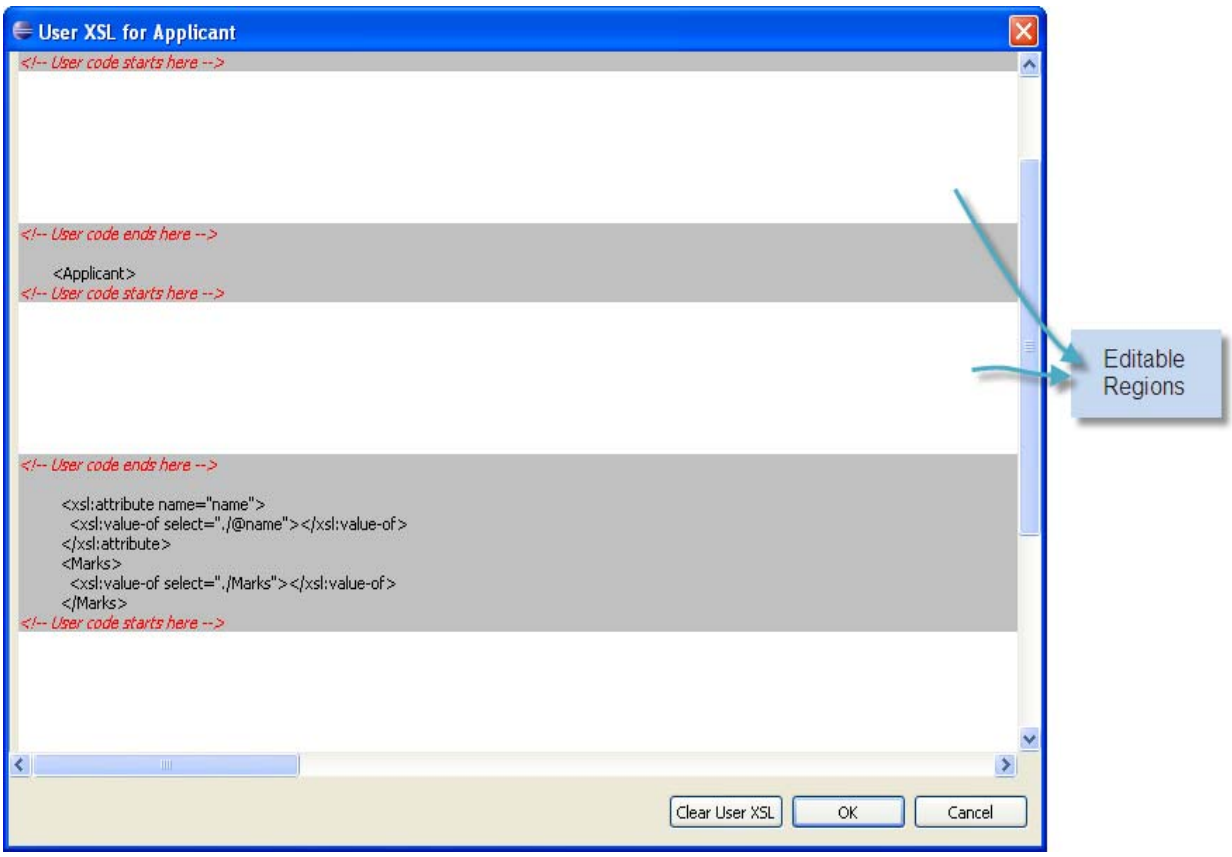


Figure 8.6.2: Editing the user xsl

As shown in Figure 8.6.2, XSL snippets can be added in the following four places:

- just above <element>
- just below <element>
- just above </element>
- just below </element>

3. Add the required if code snippet in these regions.
4. Click the **OK** button and 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 8.6.3.

Figure 8.6.3: Node with User XSL defined

5. The XSL can be tested it using **Test** option as described in the section 8.8 Testing the Transformation

8.7 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 8.7.1.

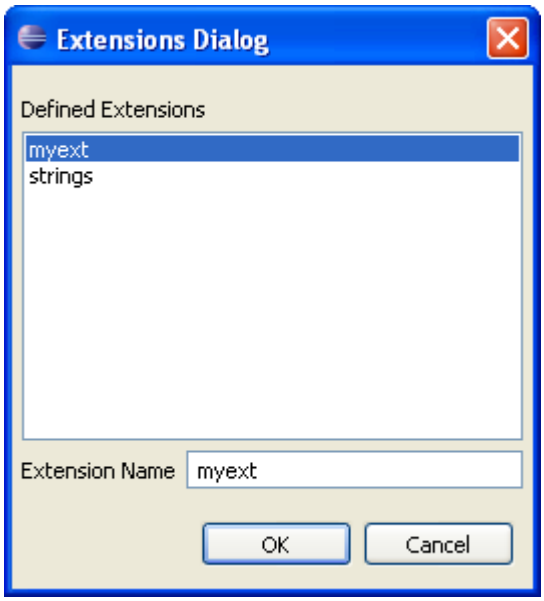


Figure 8.7.1 Extensions Dialog

- 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 8.7.2. 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.

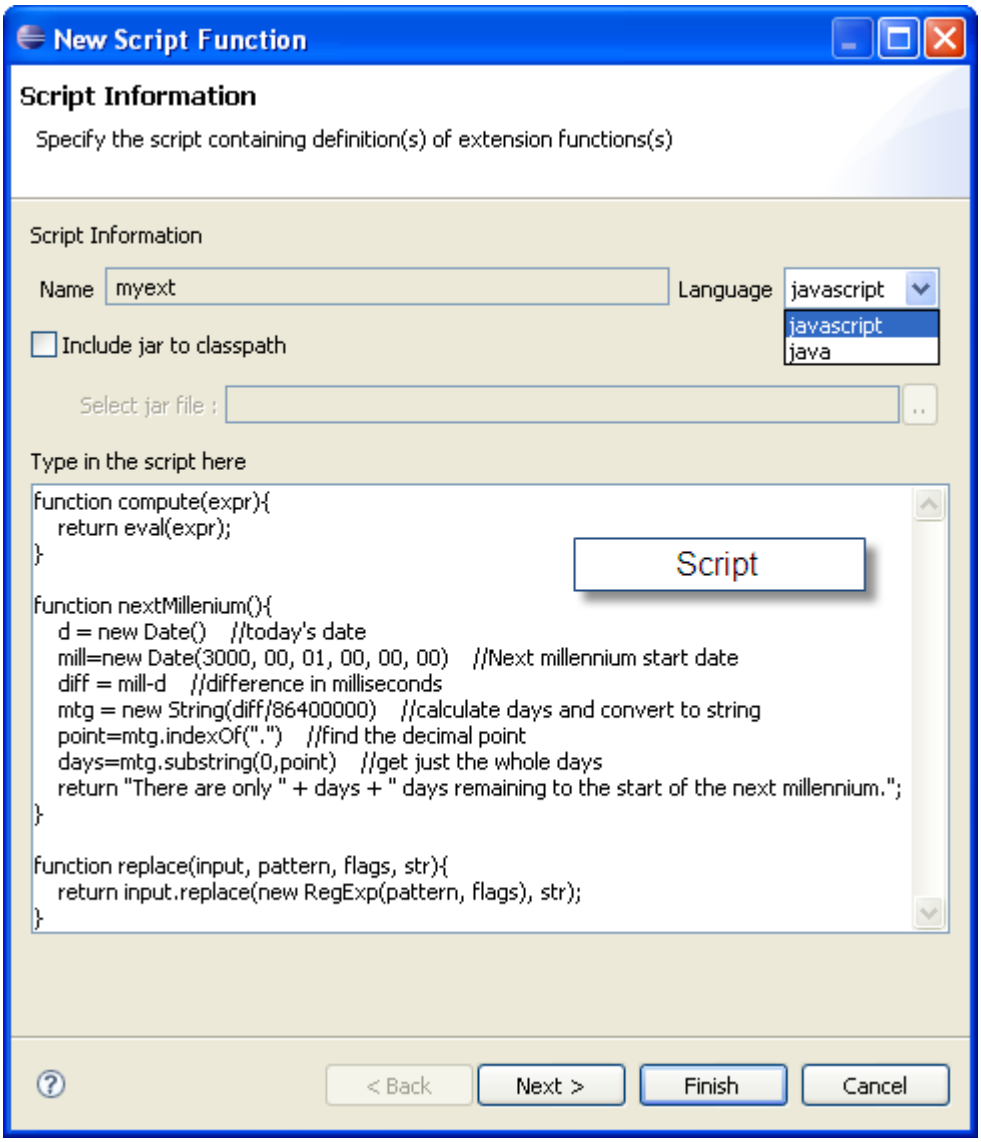


Figure 8.7.2 Script Information Page

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.

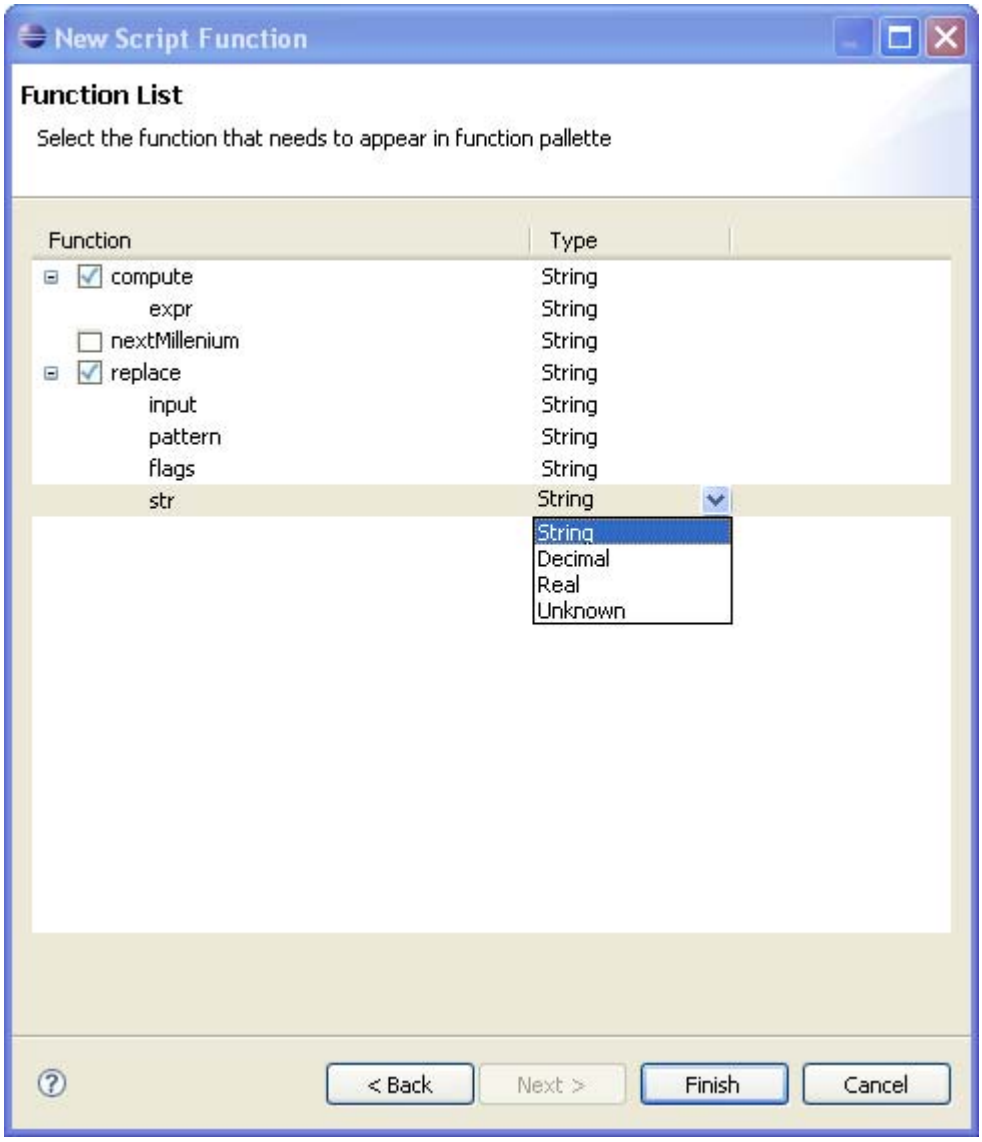


Figure 8.7.3 Function Page

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.

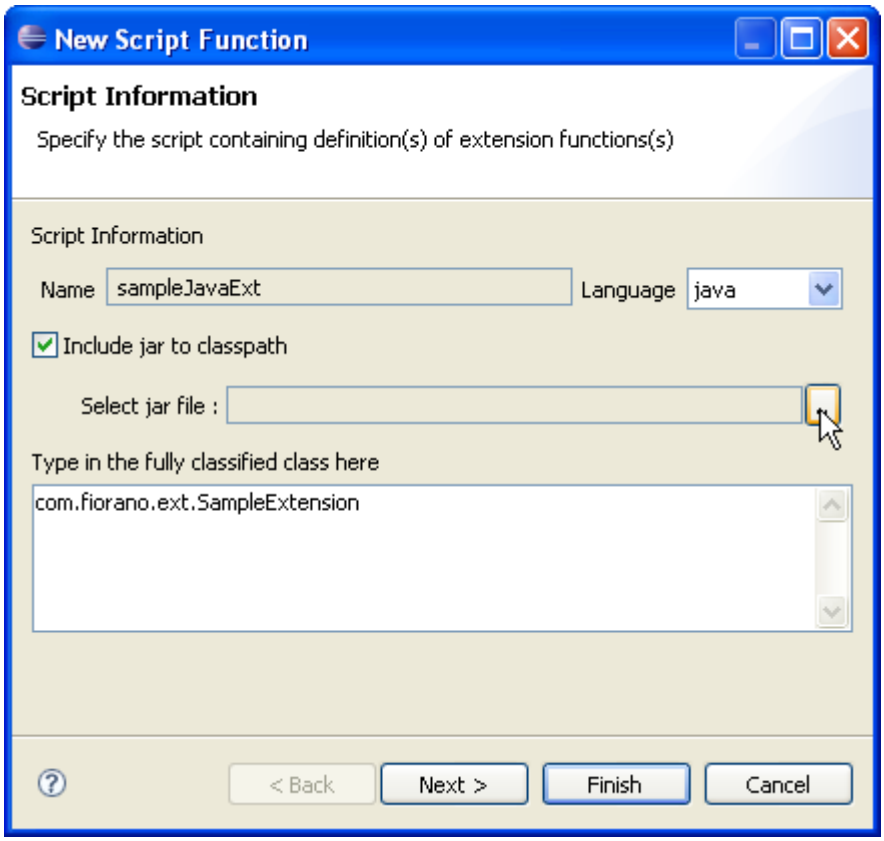


Figure 8.7.4 Adding Jar to classpath

8.8 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 8.8.1 or click the **Test** button in the tool bar

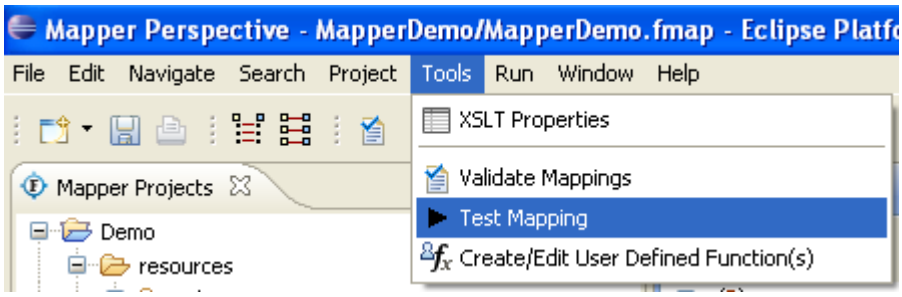


Figure 8.8.1: Invoking the Test option

The **Test XSL** wizard is displayed, as shown in Figure 8.8.2. The Transformation can be tested by following these steps:

1. 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.

2. 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.

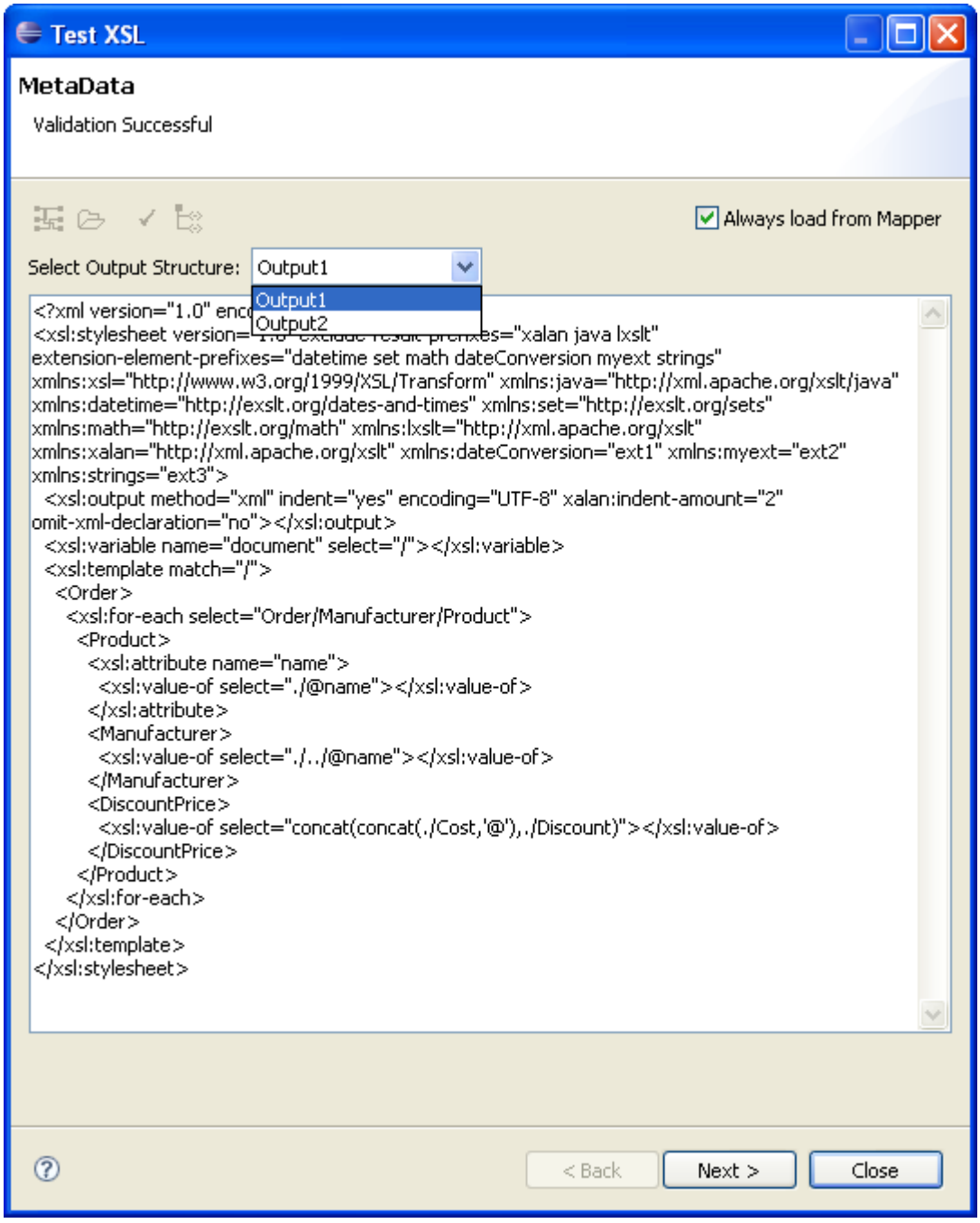


Figure 8.8.2: Metadata Page

- The **Generate Sample XML** dialog box is displayed, as shown in Figure 8.8.3. The default values are appropriate in most situations. Provide the desired values and click OK to generate a sample XML.

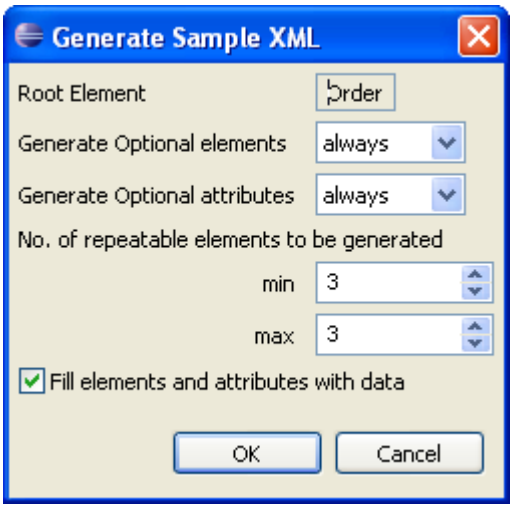


Figure 8.8.3: Selecting the sample Input XML generation options

- The sample XML is generated in the Input XML tab as shown in Figure 8.8.4.

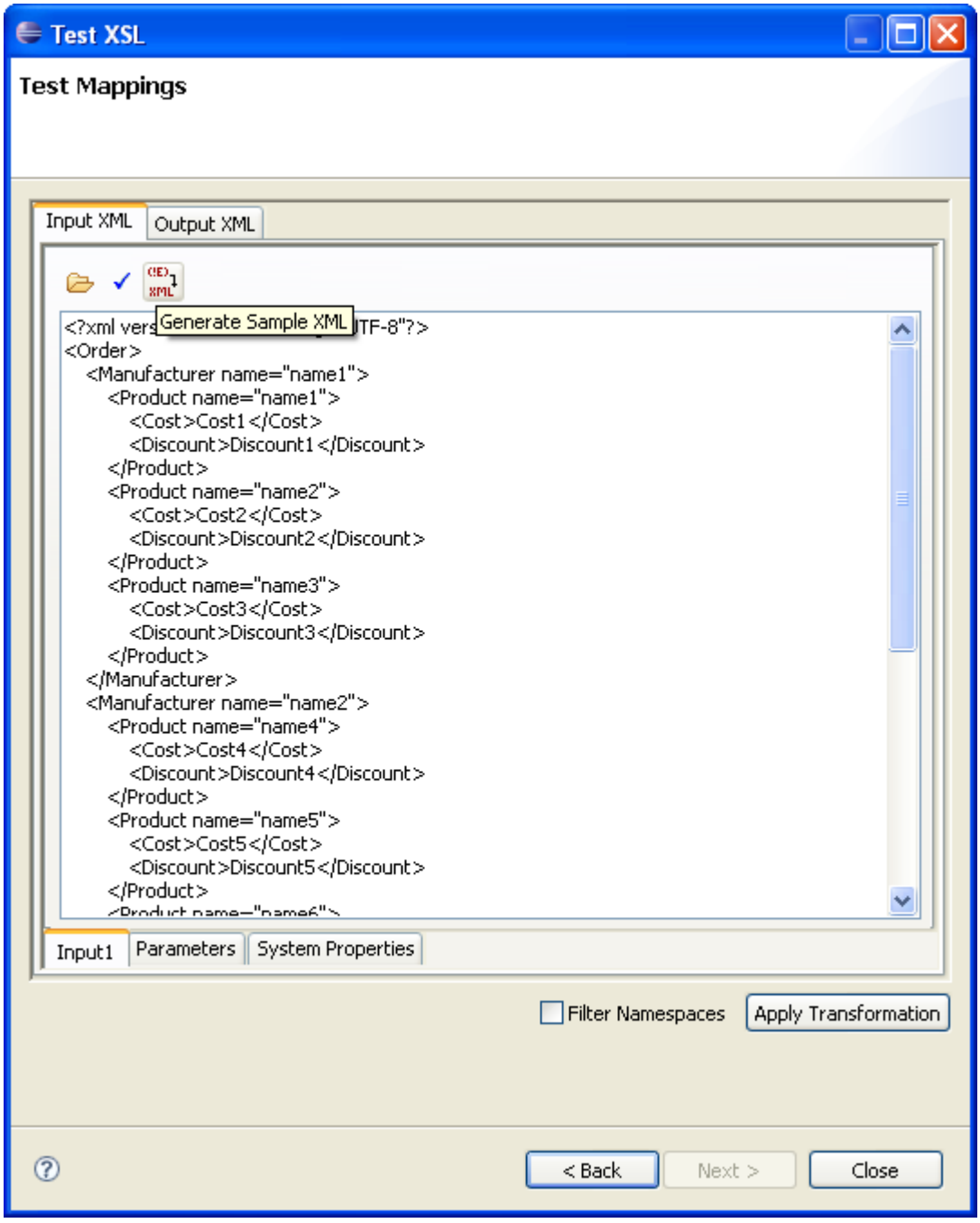


Figure 8.8.4: A Sample Input XML

- 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 defined pointing to the db.properties file which holds data for oracle data base

3. 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 8.8.6.

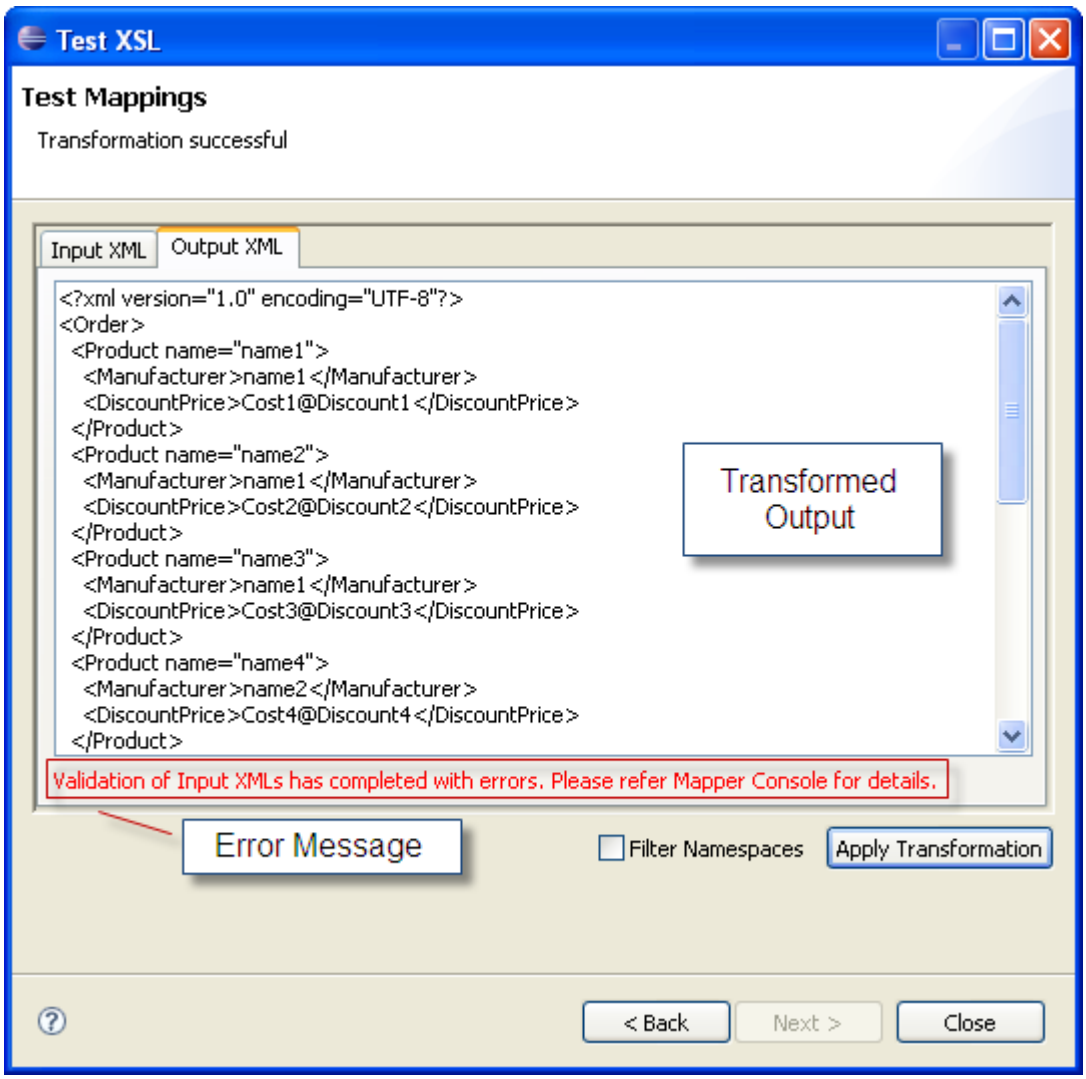


Figure 8.8.6: The Output XML resulting from the Transformation

8.9 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.

8.9.1 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

8.9.2 Importing Project from the File

A eMapper project an 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.

8.9.3 Copying functions in a Mapping

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

1. Select the function in the funclet view and click **Copy** from the right-click menu as shown below in Figure 8.9.1.

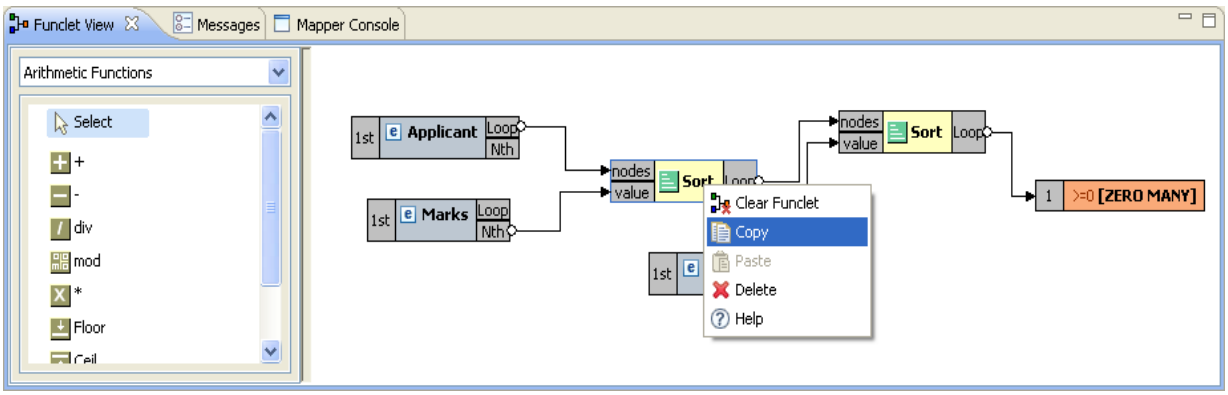


Figure 8.9.1: Copying a function

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

8.9.4 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.

8.9.5 Managing XSLT Properties

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

Click **Tools> XSLT Properties**. The XSLT Properties dialog box is displayed as shown in Figure 8.9.3.

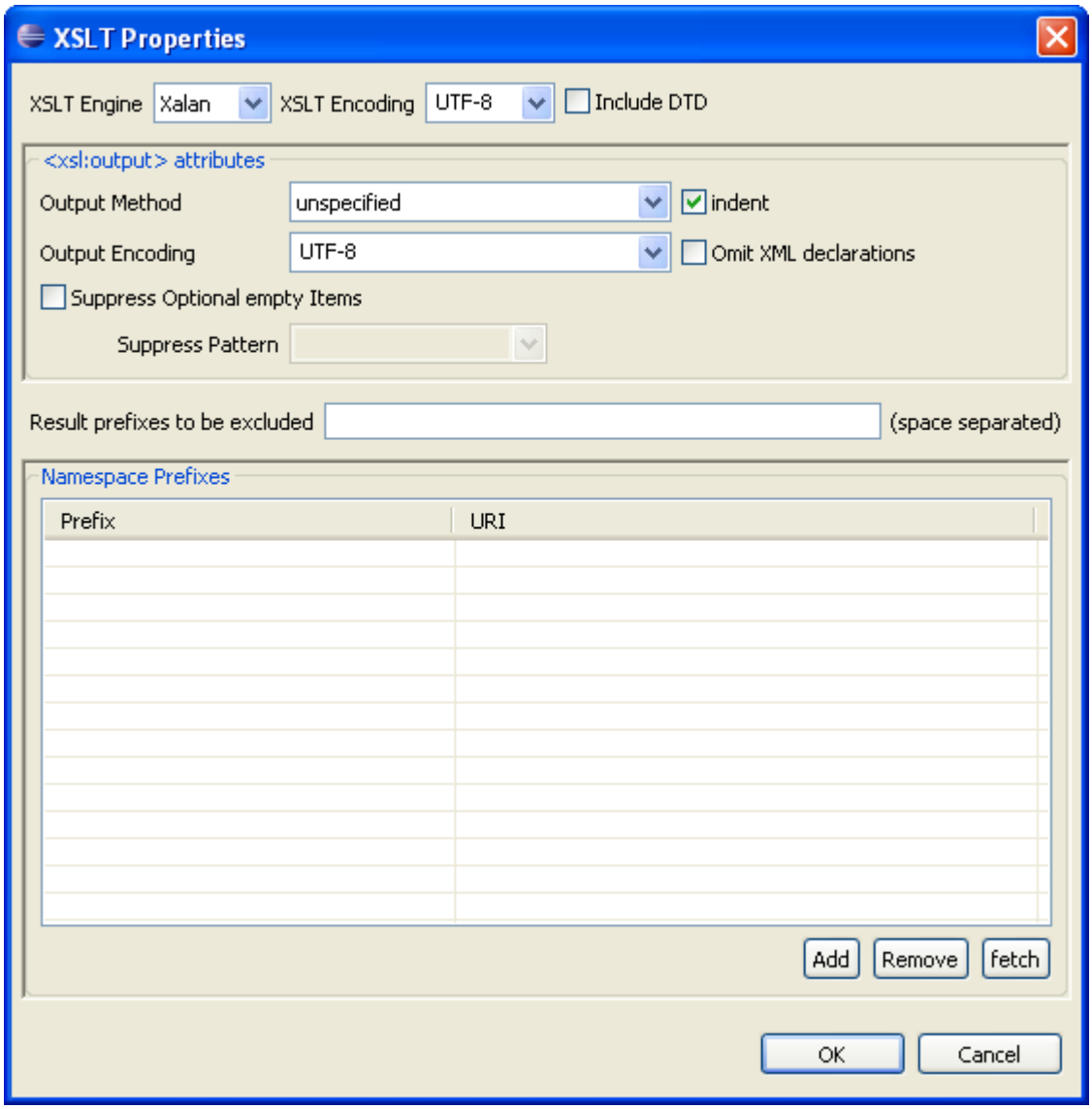


Figure 8.9.3: Viewing XSLT Properties

This dialog box contains the following components:

5. **XSLT Encoding;** Specifies the encoding of the generated XSL.
6. **Include DTD:** Select this option to include the internal specified DTD in the transformation output. By default, this option is disabled.

- **<xsl output: attributes>**

a. **Output Method:** Select the method of output after transformation from the dropdown list. The method of output can be HTML, XML, or text.

b. **Indent:** Select this option to indent the output XSLT.

c. **Output Encoding:** Specifies the encoding of the generated output XSL

- **Omit-xml-declaration:** Specifies whether the output XML generated should contain XML declaration or not.
- **Suppress optional empty items:** Select this option for defining a mapping to an output node, always generate the output node in output xml, event input xml has no matching node. It is some times desirable not to generate optional output nodes if no input matching node is found in input xml. This requires using a conditional mapping. You can specify such conditional mapping by using "User XSL" feature. eMapper can generate such conditions automatically for optional elements if this option is selected.

Chapter 9: Working With Multiple Servers And Perspective

9.1 Active Server Node

In eStudio Online Event Process Development perspective, user can add as many Enterprise Servers and Login to them and create, deploy, and run applications on them.

Active Enterprise Server in context of eStudio means states of applications and other repositories will be shown corresponding to this particular server. This will be decided by the current selection in Server Explorer view. An Enterprise Server is said to be active if the selection is on Enterprise Server node or one of its descendant nodes. For instance, in the figure 9.1.1 the **Enterprise Server_1** is an Active Enterprise Server. All other views will be in accordance to the Active Enterprise Server. For example, Service repository and Service palette will show the services present in the Active Enterprise Server.

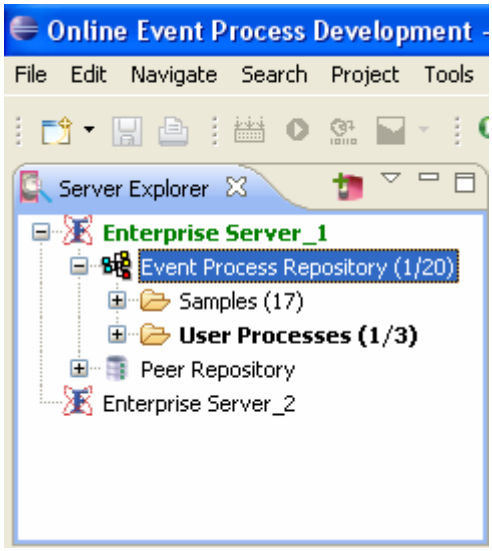


Figure 9.1.1: An Active Enterprise Server

9.2 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 user selects any node in that server).

Steps to make a server active:

1. Add two or more Enterprise servers by clicking **Add Enterprise Server** icon on the toolbar of **Server Explorer View**. For example, in figure 9.2.1 the Enterprise Server **Enterprise Server_1** and **Enterprise Server_2** are added.

2. Login in to the two servers, the Service Palette and Service Repository shows the services present in the **Enterprise Server_1**. Create some event processes in **Enterprise Server_1**, and keep the created event process editors open.

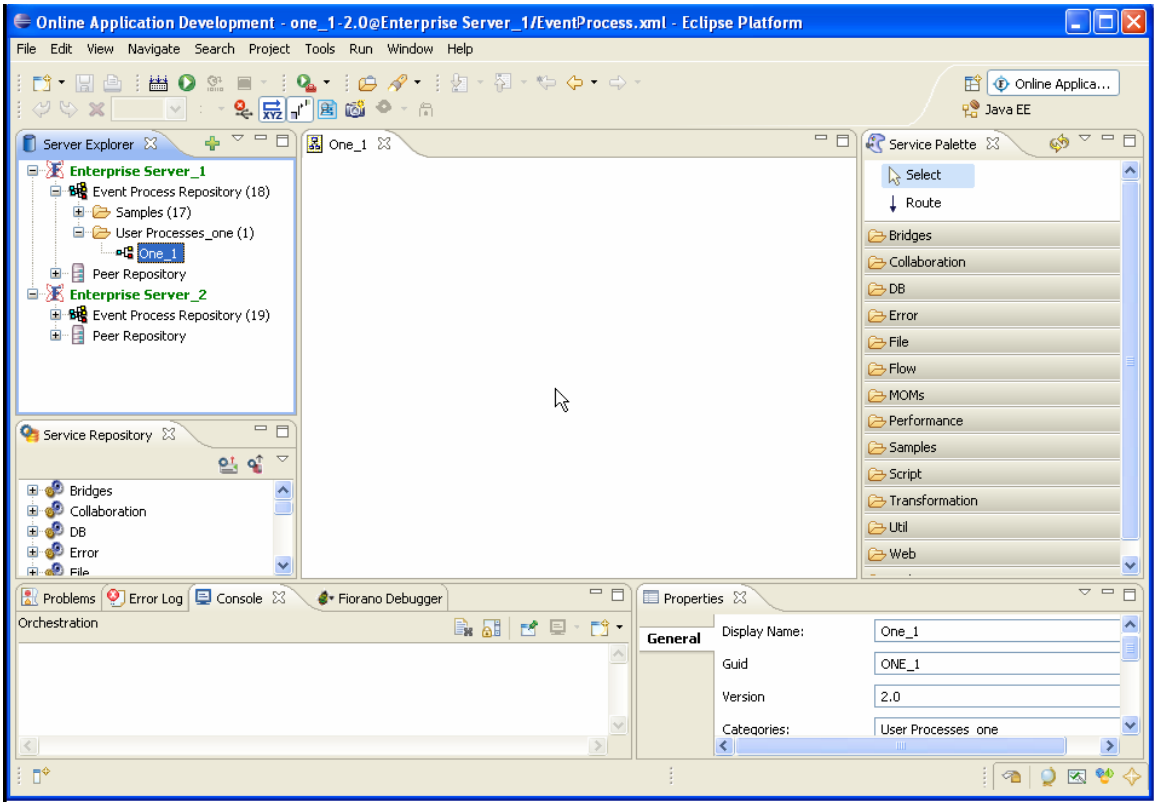


Figure 9.2.1: Two Active Enterprise Servers

3. Now click on any node inside **Enterprise Server_2** node in the **Sever Explorer** view. The editors related to **Enterprise Server_1** will be closed and service palette and service repository shows the services present in the **Enterprise Server_2**.

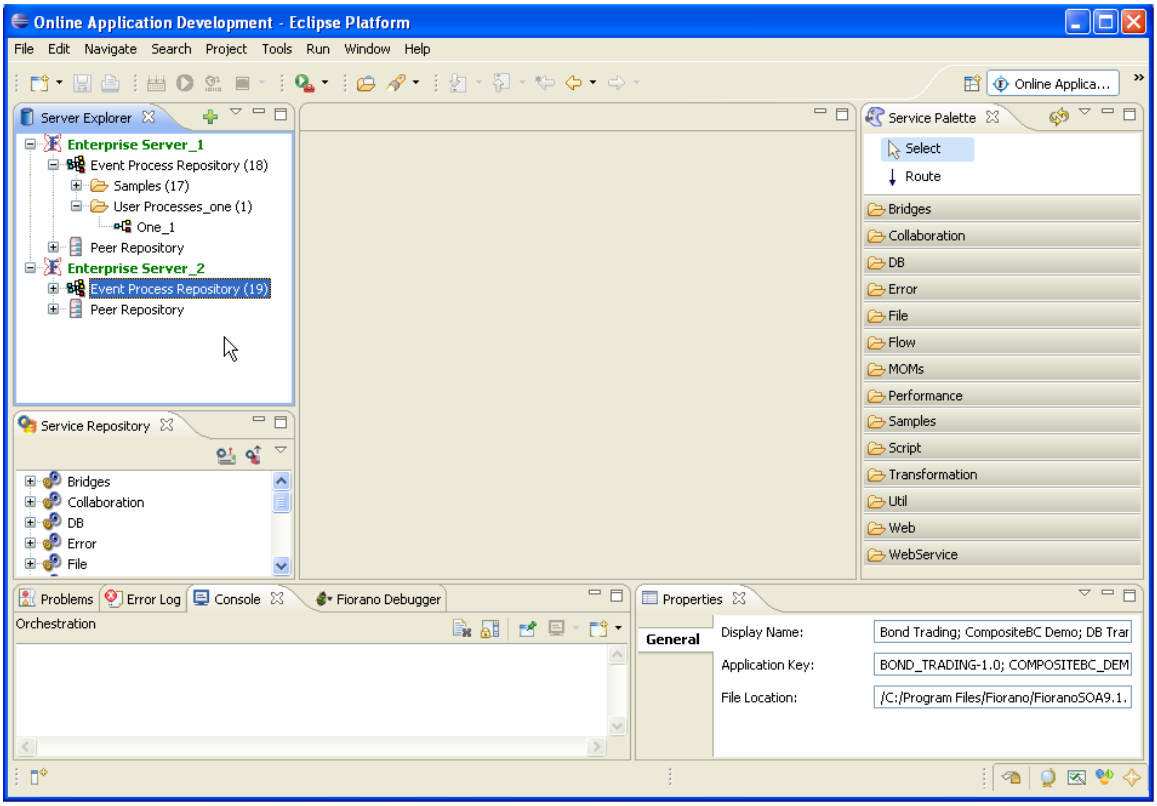


Figure 9.2.2: Services present in Enterprise Server_2

4. To switch back to **Enterprise Server_1**, click on any node inside **Enterprise Server_1** node. All the editors which are opened previously are restored.

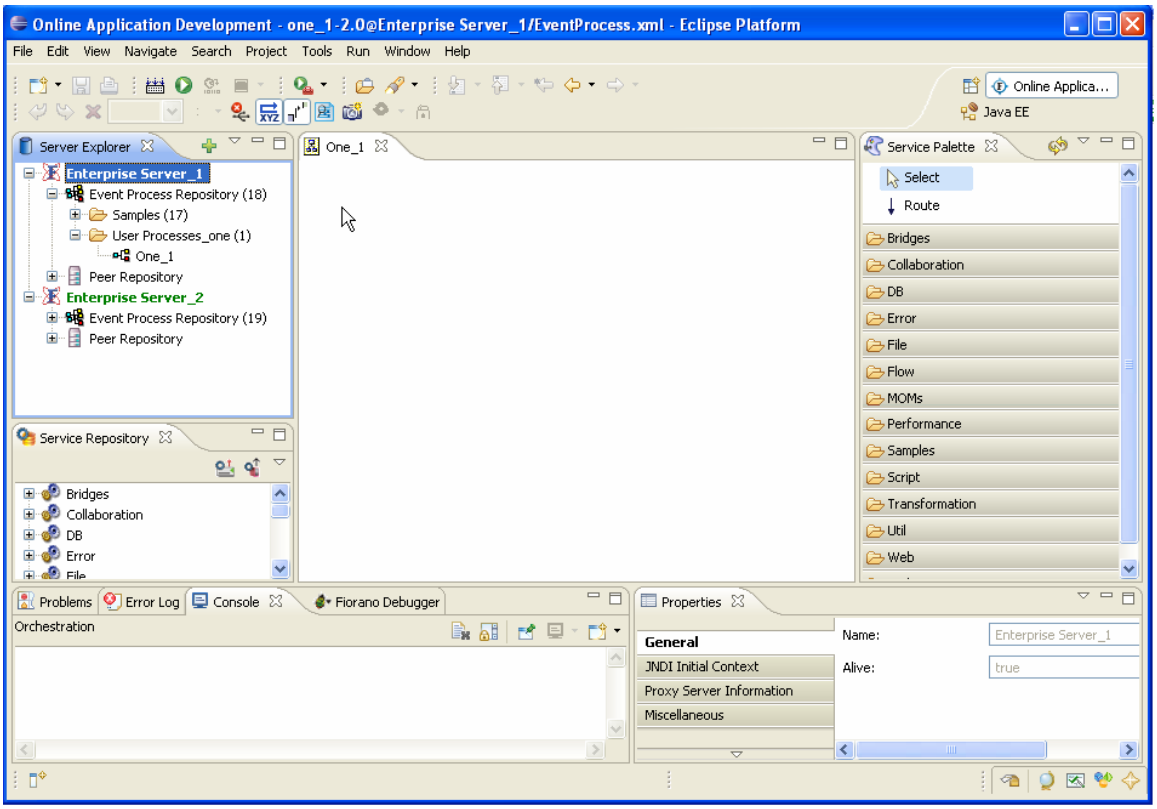


Figure 9.2.3: Services restored in Enterprise Server_1

9.3 Switching Between Perspectives

eStudio has two developing perspectives; Offline Event Process development and Online Event Process development

To change the perspective, perform the following steps:

1. Click the **Open Perspective** button from the shortcut bar on the right-hand side of the Workbench window.
2. Select **Other...** from the drop-down menu.

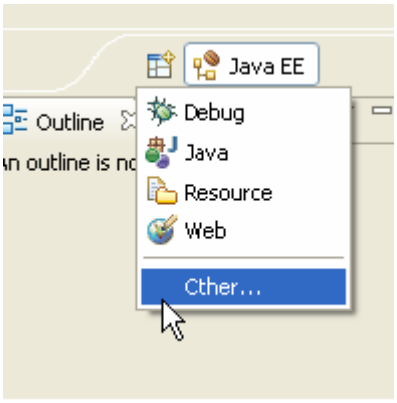


Figure 9.3.1: Selecting the Other.. option from perspective button

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

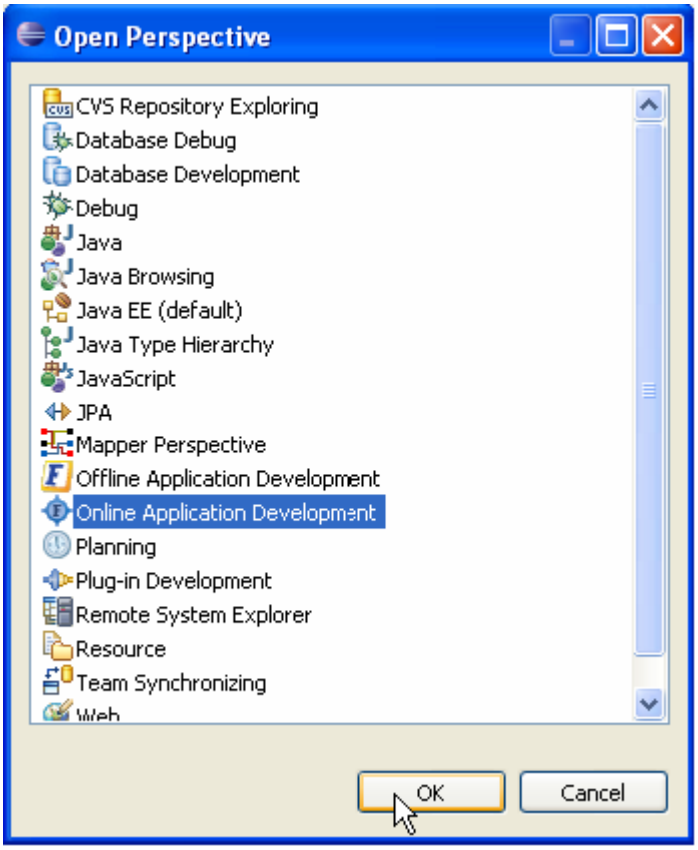


Figure 9.3.2: Selecting Online Application Development perspective

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.

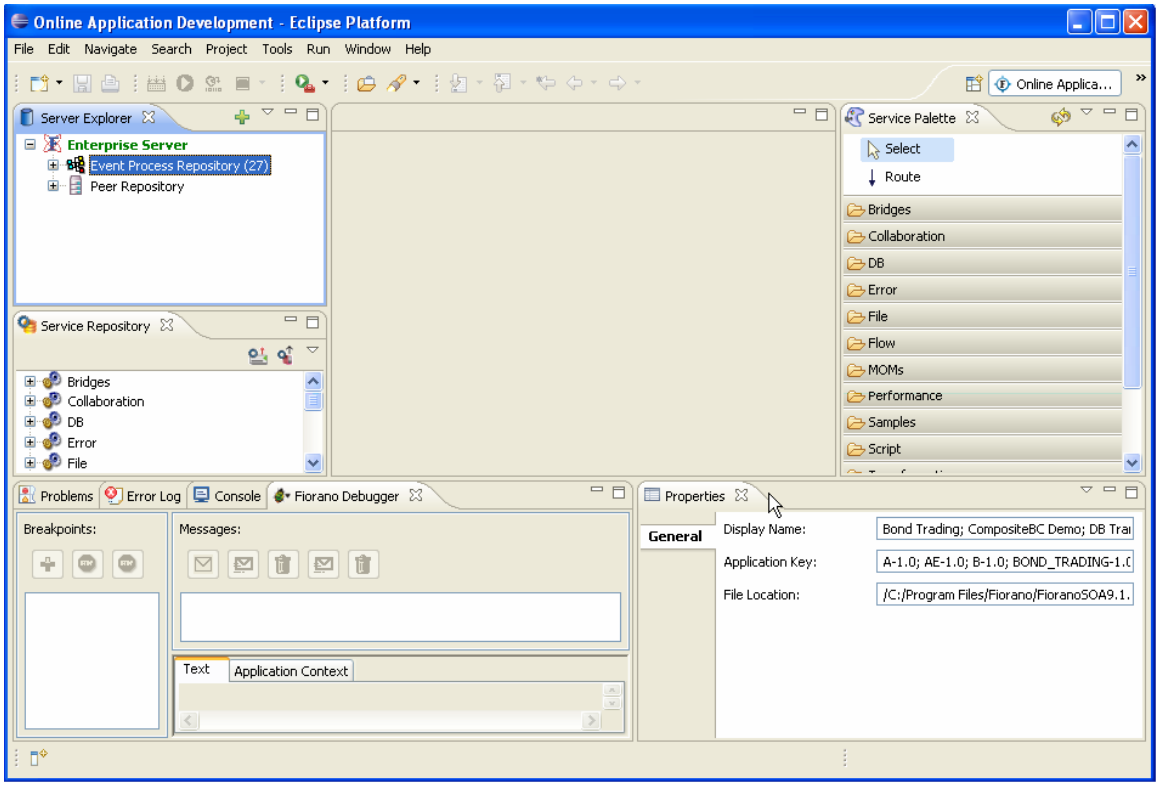


Figure 9.3.3: Online Application Development Perspective

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 9.3.4. During offline application development, there will not be any interaction with the server.

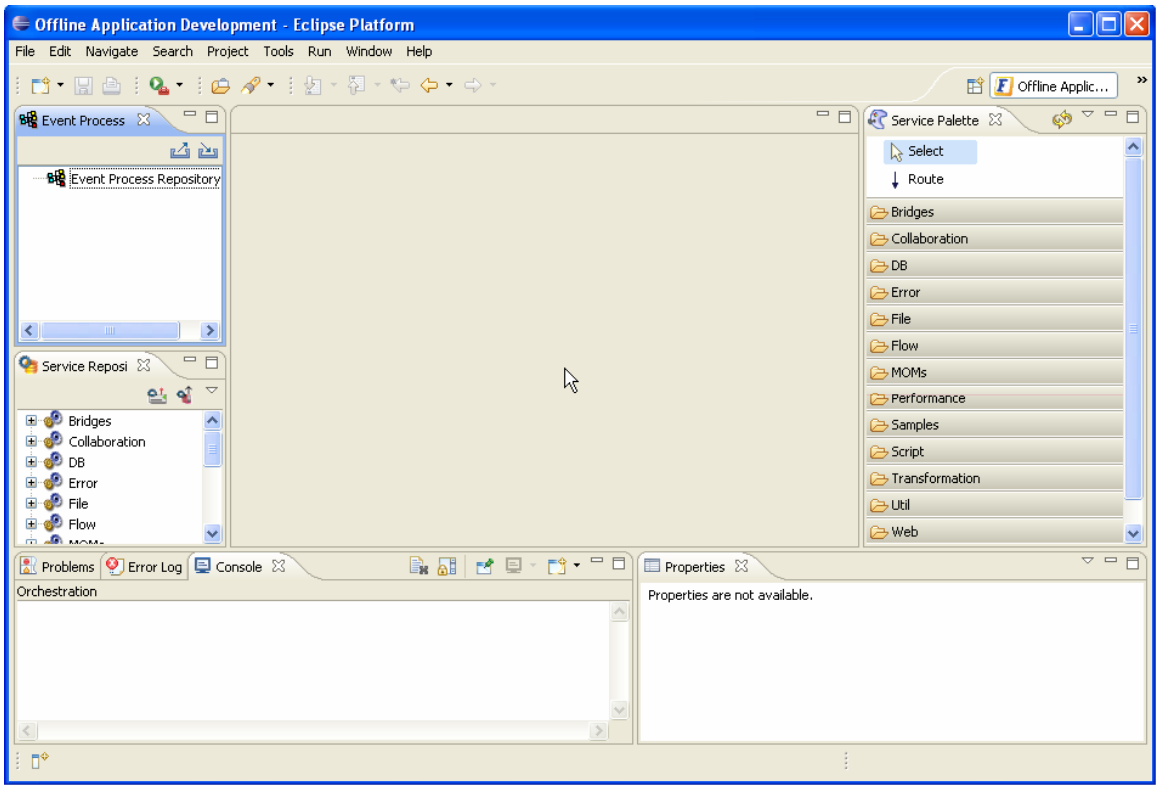


Figure 9.3.4: Offline Application Development Perspective

Chapter 10: Custom Preferences

Fiorano Preferences are available under Window > Preferences -> Fiorano.

10.1 Event Process Preferences

Event Process Preferences can be found under the category Fiorano in Preferences window. The color of the error routes/ports can be changed from the color dialog provided for the error port color preference. Similarly the color of request reply port can also be changed. The preference page for Event Process is shown below:

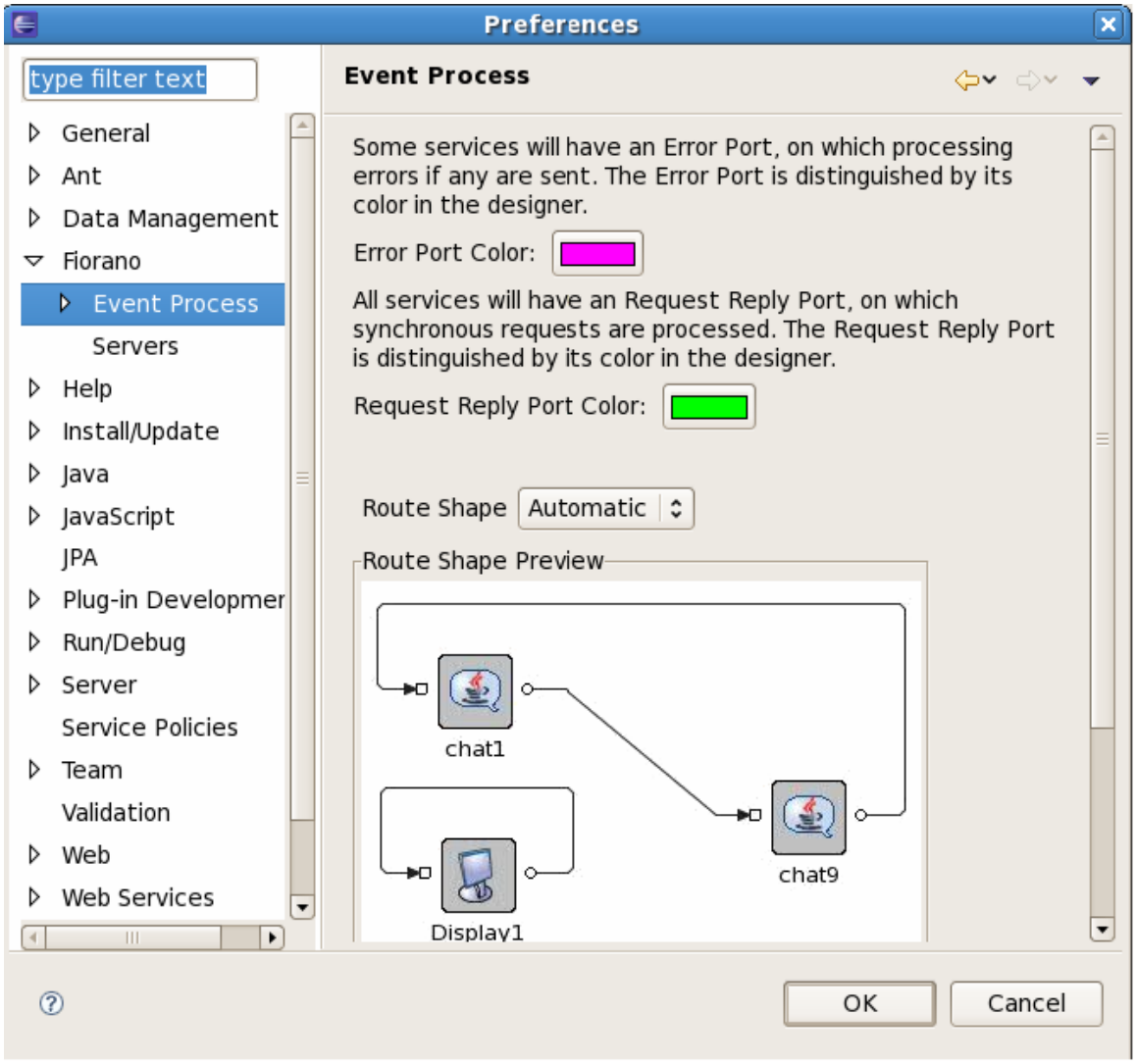


Figure 10.1: Event Process Preferences dialog box

10.1.1 Service Instance Execution Status Option

This option is available under the Event Process Preference Category. The color of the Service Instance at different execution status can be configured from here, that is, when the Service Instance is running, stopped and so on.

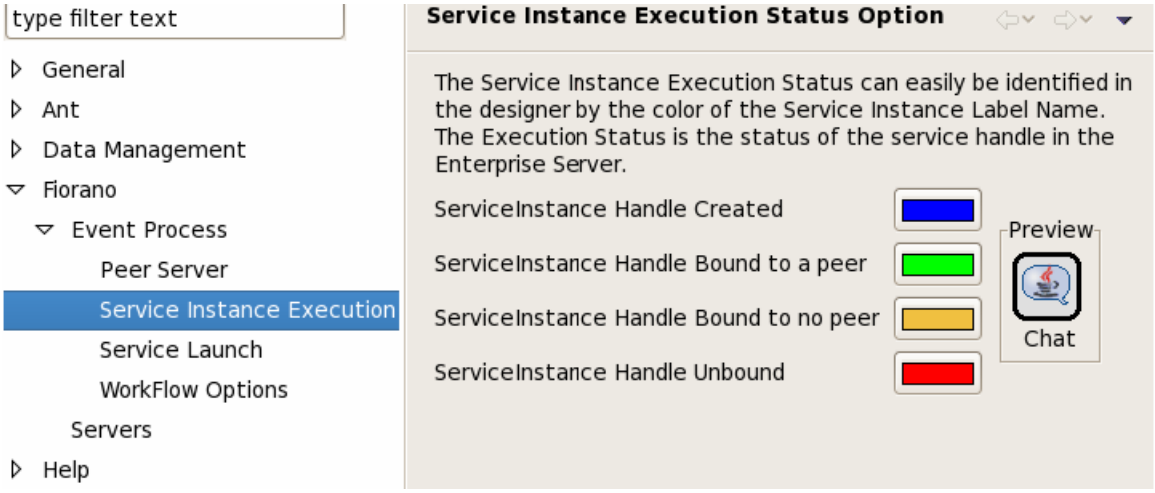


Figure 10.1.1: Service Instance Execution Status Option

10.1.2 Work Flow Options

This option is available under the Event Process Preference Category. The color of the ports for work flow item and work flow end can be configured from here.

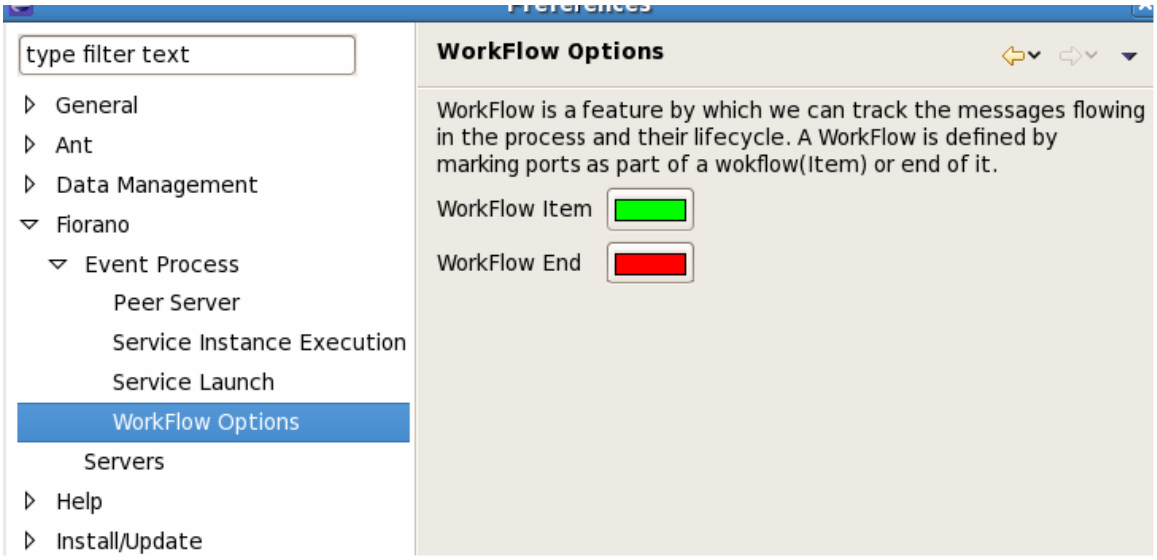


Figure 10.1.2: Work Flow Options

10.2 Server Preferences

Server Preferences can be found under the category Fiorano in Preferences window. User can set the server details such as IP of the Enterprise Server and security credentials.

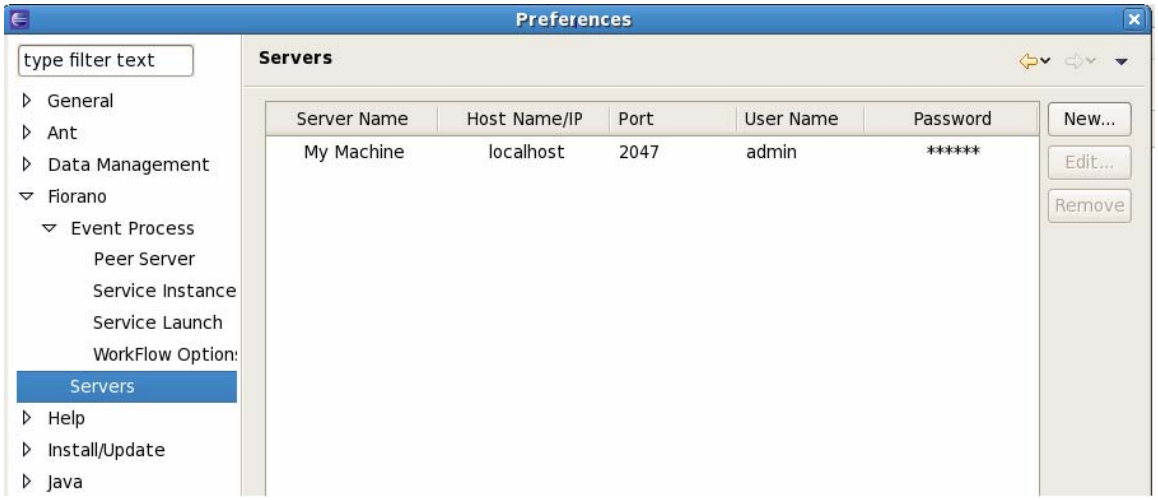


Figure 10.2: Server Preferences dialog box

10.3 Key Board Short Cut Preferences

The option to edit key board shortcuts is 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. For example the shortcut to add an application can be changed from Ctrl+I to Ctrl+J.

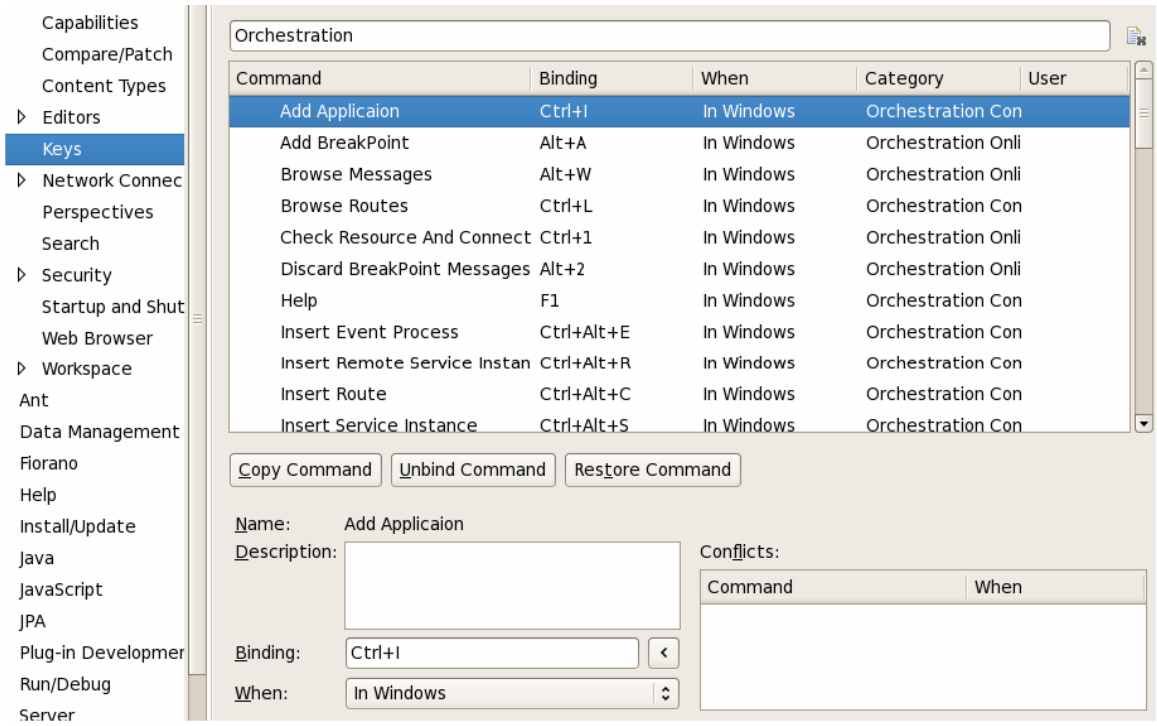


Figure 10.3: Keys Preferences dialog box