Oracle® Communications MetaSolv Solution

XML API Developer's Reference
Release 6.2.1

October 2013

ORACLE

Oracle Communications MetaSolv Solution XML API Developer's Reference, Release 6.2.1

Contents

Preface

This document explains how to use the Oracle Communications MetaSolv Solution Integration and Portal Toolkit to integrate MetaSolv Solution with other MetaSolv Products and with external applications. The toolkit provides a workspace and other tools for the integration development and testing.

Audience

This document is for individuals who are responsible for using the MetaSolv Integration and Portal Toolkit in a development environment and developing software to integrate an external application or another MetaSolv product with MetaSolv Solution. This document assumes the reader has a working knowledge of Oracle 11g, Windows XP Professional, Oracle WebLogic 10.3.1, and Java JEE.

Related documentation

For more information, see the following documents in Oracle Communications MetaSolv Solution 6.2.1 documentation set:

- *MSS Planning Guide:* Describes information you need to consider in planning your MetaSolv Solution environment prior to installation.
- *MSS Installation Guide:* Describes system requirements and installation procedures for installing MetaSolv Solution.
- *MSS System Administrator's Guide:* Describes post-installation tasks and administrative tasks such as maintaining user security.
- *MSS Database Change Reference:* Provides information on the database changes for the MetaSolv Solution 6.2.1 release. Database changes for subsequent maintenance releases will be added to this guide as they are released.
- *MSS Network Grooming User's Guide:* Provides information about the MSS Network Grooming tool.
- *MSS Technology Module Guide:* Describes each of the MetaSolv Solution technology modules.
- *MSS Data Selection Tool How-to Guide:* Provides an overview of the Data Selection Tool, and procedures on how it used to migrate the product catalog, equipment specifications, and provisioning plans from one release of your environment to another.

- *MSS Operational Reports:* Provides an overview of using Operational Reports and Business Objects with MSS, and procedures for running reports, updating universes, and simple maintenance.
- *MSS CORBA API Developer's Reference:* Describes how MetaSolv Solution APIs work, high-level information about each API, and instructions for using the APIs to perform specific tasks.
- *MSS Custom Extensions Developer's Reference:* Describes how to extend the MetaSolv Solution business logic with custom business logic through the use of custom extensions.
- *MSS Flow-through Packages Guide:* Describes information and procedures you need to install and work with the flow-through packages provided by Oracle as an example of how to integrate MetaSolv Solution with ASAP for flow-through activation.

For step-by-step instructions for tasks you perform in MetaSolv Solution, log into the application to see the online Help.

Additional documentation resources

You can obtain additional information about the XML APIs used to integrate MetaSolv Solution with other applications from the following resources:

- **XML Schema**—The XML schema used in integration for MetaSolv Solution have documentation included directly in the schema.
- **Sample code**—The sample code is installed with the MetaSolv Solution installation on the workstation if you have the XML API option.

1

Setting Up

This chapter contains general information on getting ready to develop an integration application. It does not contain installation instructions for MetaSolv Solution.

Technical requirements and installation instructions

- See the technical requirements for the MetaSolv Solution application and client in the *MetaSolv Solution Planning Guide*.
- To find complete installation instructions for the MetaSolv Solution Integration and Portal Toolkit, see the chapter entitled "Installing MetaSolv Solution with the XML API option" in the *MetaSolv Solution Installation Guide XML API Option*. Regarding the installation, note the following:
  - Single server installation is required for development.

    Clustered server installation is not available for development. The installation program for MetaSolv Solution is the same for production and development. The WebLogic configuration is different only in the selection of Production or Development mode.
  - The connection pool MSLVwliPool must be established.

    Connection pooling is a technique used for sharing server resources among requesting clients. This allows for multiple clients to share a cached set of connection objects that provide access to a database. The MSLVwliPool is used by calls generated from the XML APIs and is mapped to the username APP_INT. This means that any records created or updated in the MetaSolv Solution database that resulted from a XML API call will have the last_modified_userid field set to APP_INT.

About the development database

WebLogic allows you to accept a default PointBase database when you configure the domain. Oracle recommends that you use a test Oracle database. A tool is provided that allows you to create an SQL script file that can be run against your production database to re-create any new tables created in the development database. For information on the tool, see “Updating the production database” on page 79.

Recommended deployment configurations

All of the development components can be installed and run on a single machine. Components of the development environment include:

- **WebLogic Integration**—WebLogic Server with the WebLogic Integration extensions included.
- **WebLogic Workshop IDE**—An integrated development environment used to develop and test customer integration and Web GUI applications.
- **MetaSolv Solution with the XML API option**—This application has a minimum deployment of MetaSolv Solution and its client. The client is used to set up gateway events in the MetaSolv Solution application, and it requires a Windows environment.

The following figure shows a Windows development environment.

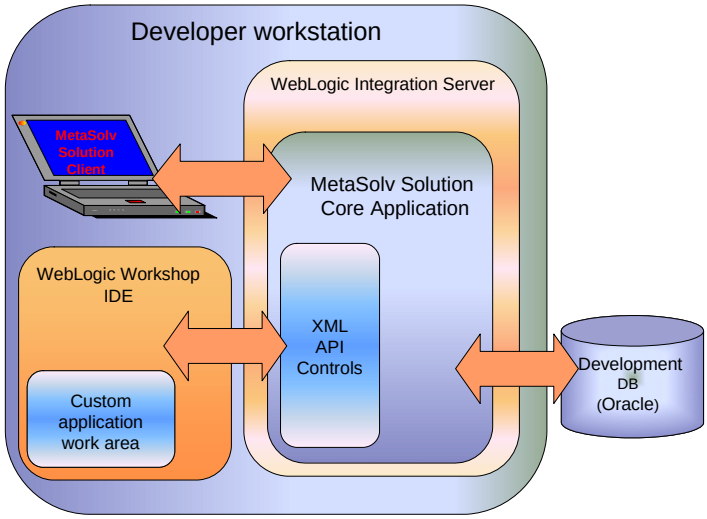


Figure 1: Windows developer workstation environment

In a Windows environment, all WebLogic components and the MetaSolv Solution core and client can be installed on the same developer workstation. The XML API controls are included in the MetaSolv Solution installation.

The following figure shows a UNIX development environment.

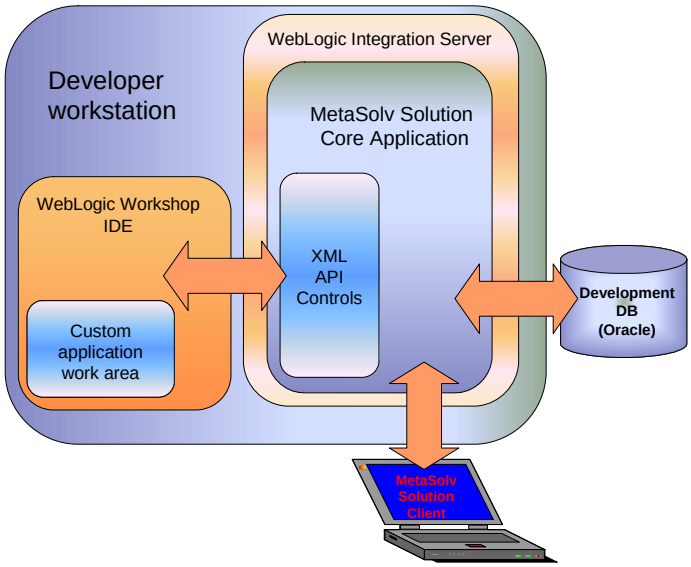


Figure 2: UNIX developer workstation environment

The MetaSolv Solution client requires a Windows environment so it must be loaded onto a separate machine in a UNIX development environment.

JMS messaging requirements

The MetaSolv Solution installation program sets up a paging store for each JMS server. The paging store is used exclusively for paging out non-persistent messages for the JMS server and its destinations.

The installation program also sets the **Store Enabled** check box to *selected*. This setting is for persistent messages, which are necessary for a guaranteed message delivery system. If you create additional JMS destinations (queues/topics) after installation, you must select the **Store Enabled** check box for these destinations manually.

In the WebLogic Server Administration Console, the **Store Enabled** check box is located on the **General** tab of a JMS Server.

To access the Stored Enabled check box:

1. Open the Weblogic Server Administration Console by entering the following in your internet browser:

   http://*host_admin*:*port*/console

   where:

   host_admin is the name of the administration server

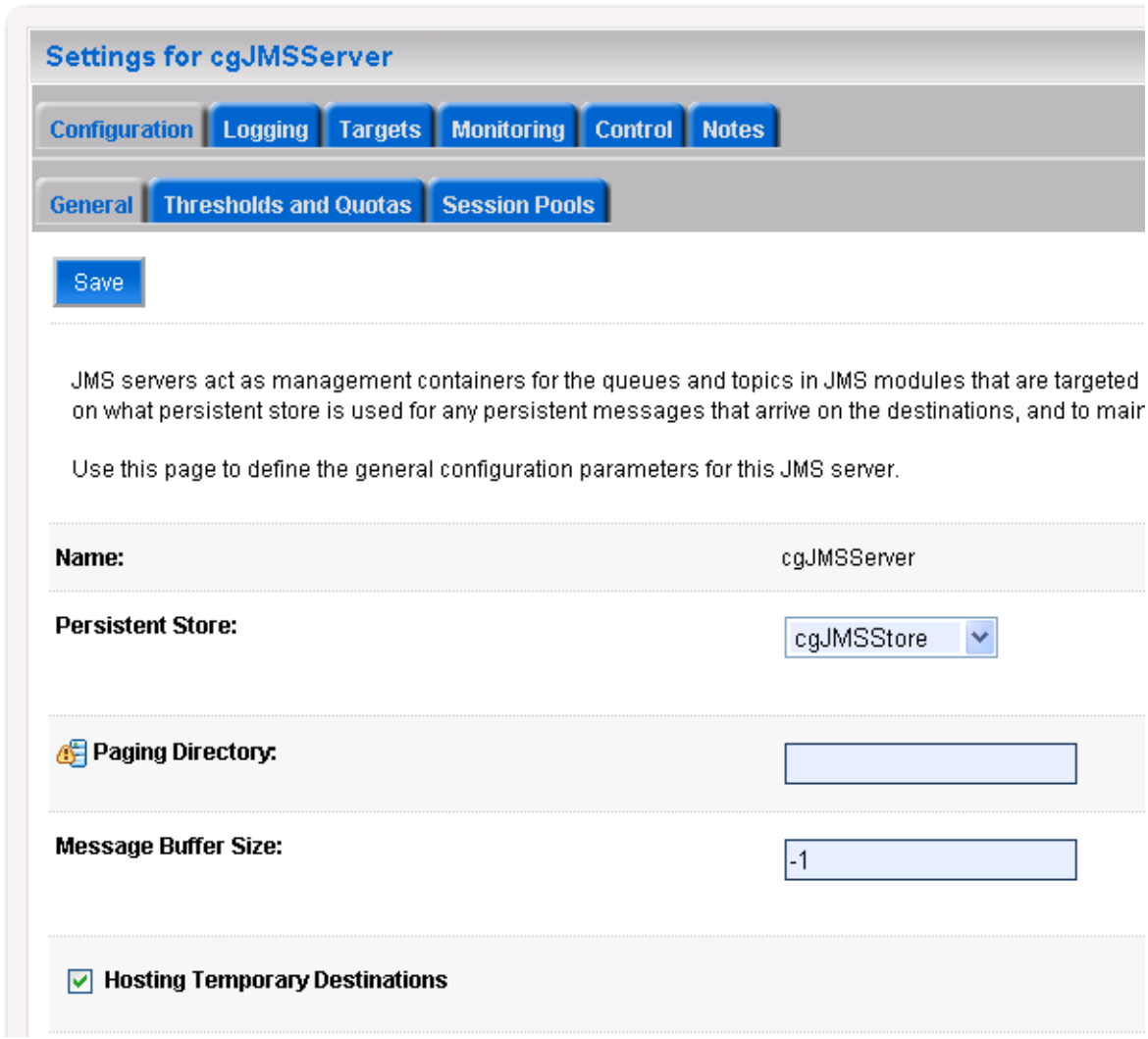
   port is the administration server port number

2. When the Login window appears, enter your user name and password and click **Login**.
3. On the Home page, under Domain Configurations, under Services, and under Messaging, click the **JMS Servers** link.

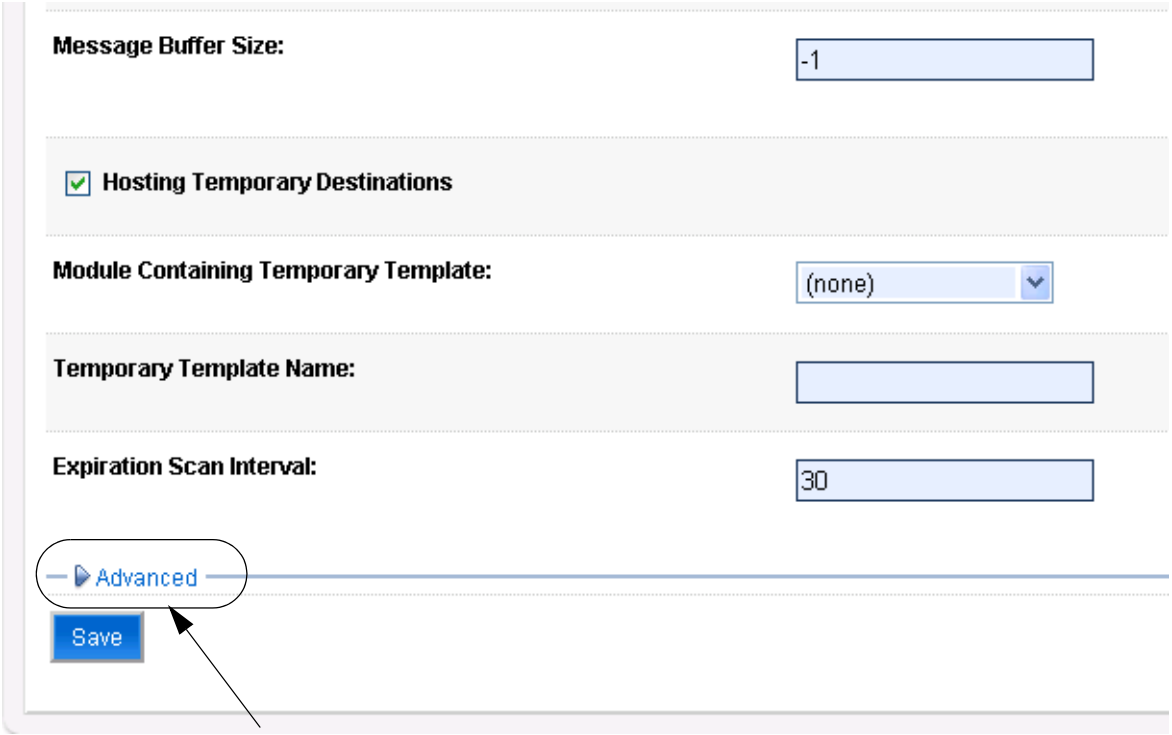
   The Summary of JMS Servers page displays.

4. Select a JMS server from the list, such as cgJMSServer.

   The Settings page for the JMS server that you selected displays:

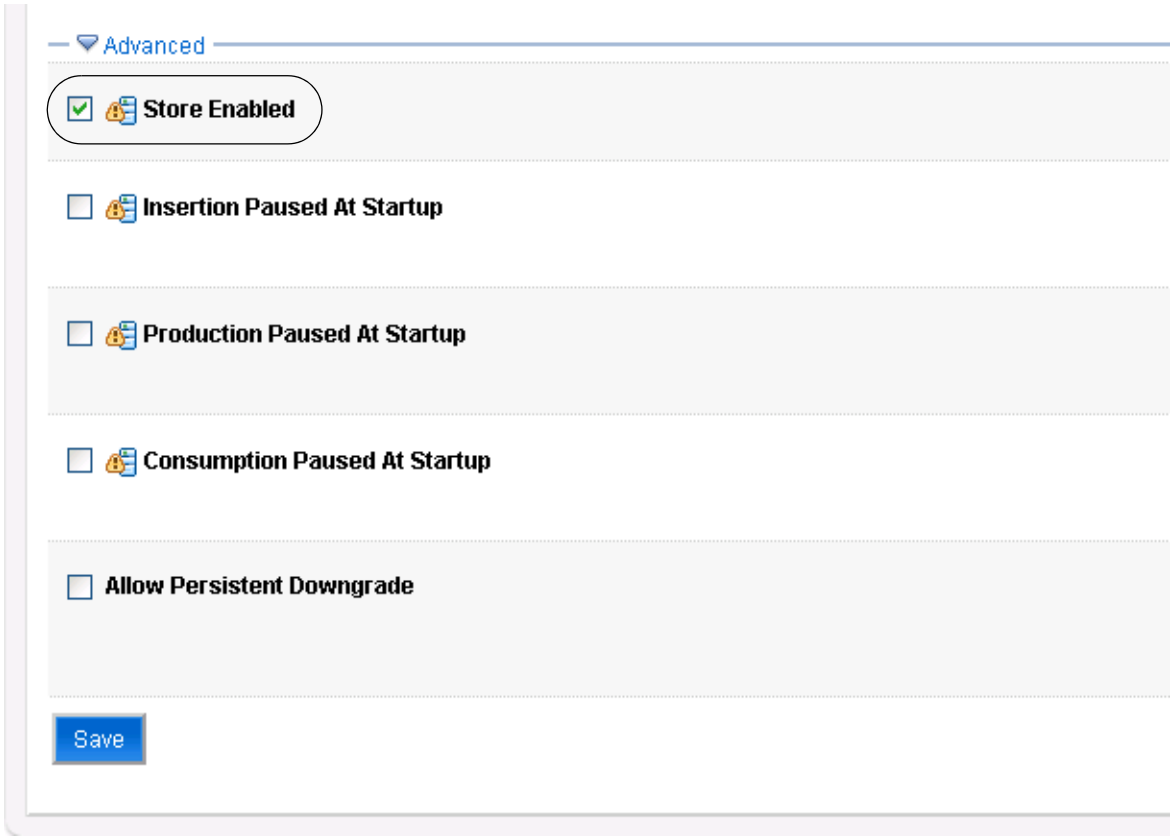


5. Scroll down.



6. Click **Advanced** to expand the General tab to include additional advanced fields.

   The **Stored Enabled** check box is now visible on the General tab of the JMS server.


Advanced
Store Enabled
Insertion Paused At Startup
Production Paused At Startup
Consumption Paused At Startup
Allow Persistent Downgrade
Save

2

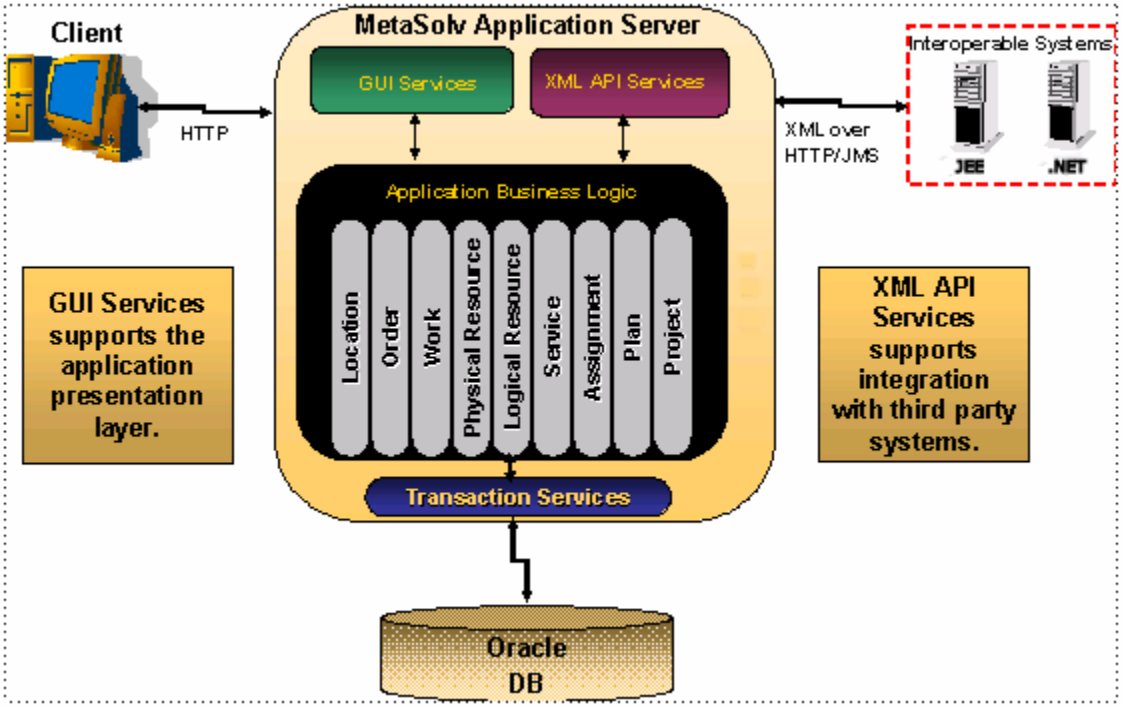
Integration Overview

This chapter provides basic information about the MetaSolv Solution Integration and Portal Toolkit and how you can use it to integrate with MetaSolv Solution.

High Level Overview

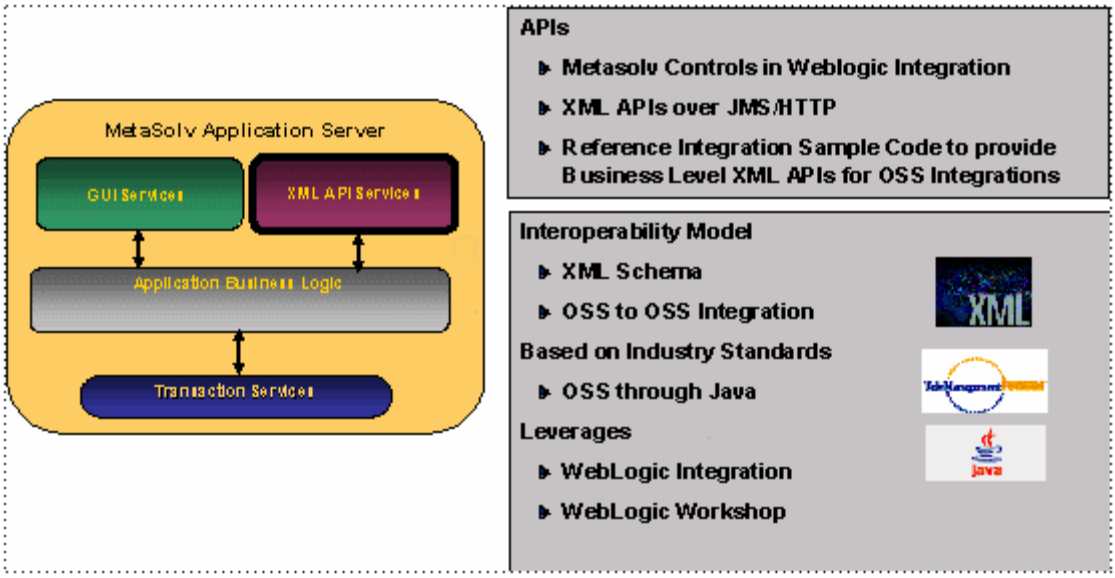
The figure below represents a high level overview of the MetaSolv Solution Integration. At it's core is the Application Business Logic, which is grouped by functional area such as Location, Order, Work, etc. The Application Business Logic is used by both the GUI Services and the XML API Services. GUI Services supports the application presentation layer to the client over HTTP. The XML API Services supports integration with third party systems using XML over HTTP or JMS. JMS is the recommended choice because it is a more reliable messaging service. Collectively, the MetaSolv Solution Application Server runs on a WebLogic server.

Figure 3: MetaSolv Solution Integration Overview



The figure below includes information regarding tools utilized by the MetaSolv Solution Integration, and standards that it followed.

Figure 4: MetaSolv Solution Integration Tools and Standards



About the MetaSolv Solution Integration and Portal Toolkit

The MetaSolv Solution Integration and Portal Toolkit is a package that allows you to integrate a third-party software product with MetaSolv Solution using XML APIs. The toolkit includes:

- Controls
- Schema
- WebLogic 10.3.1

  This is a third-party software product that can be purchased as an option with MetaSolv Solution. It provides an integration environment, transformation mapping, and a high-level interface that represents code elements visually in the work area in WebLogic Workshop.

The following sections describe the components of the Integration and Portal Toolkit.

Controls

A Java control is code that forms a reusable component that can be used anywhere within a platform application. The MetaSolv Solution controls referred to in this document are individual applications that expose XML APIs for integration purposes.

WebLogic Workshop, a component of WebLogic, provides Java controls that you can use to encapsulate business logic and to access enterprise resources such as databases, legacy applications, and web services.

WebLogic controls

WebLogic allows the use of three different types of Java controls:

- **Built-in controls**

  Built-in controls, included in the WebLogic Workshop, provide easy access to enterprise resources. For example, the Database control makes it easy to connect to a database and perform operations on the data using simple SQL statements, while the EJB control lets you easily access an EJB. Built-in controls provide simple properties and methods for customizing their behavior, and in many cases you can add methods and callbacks to further customize a control.

  For more information on WebLogic built-in controls, see the Oracle WebLogic documentation at:

  http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html

- **Portal controls**

  A portal control is a type of built-in Java control designed for the portal environment. If you are building a portal, you can use portal controls to expose tracking and personalization functions in multi-page portlets.

- **Custom controls**

  You can also build a custom Java control from scratch. A custom control can act as the nerve center of a piece of functionality, implementing the desired overall behavior and delegating subtasks to built-in Java controls (and/or other custom Java controls). This use of a custom Java control ensures modularity and encapsulation. Web services, JSP pages, or other custom Java controls can simply use the custom Java control to obtain the desired functionality, and changes that may become necessary can be implemented in one software component instead of many.

  The MetaSolv Solution XML API controls were developed in WebLogic's integration environment as custom controls. The controls contain code that transforms the XML input into the proper format for MetaSolv Solution and transforms the response that returns from MetaSolv Solution into the proper format for the third-party application.

MetaSolv Solution controls

Each control works with a specific portion of the MetaSolv Solution functionality. The MetaSolv Solution controls in the Integration and Portal Toolkit include:

- Customer Management
- Order Management
- Inventory Management
- Network Resource Management
- Service Order Activation
- LSR Management
- Event Management

The controls correspond to XML APIs. When you add a control into WebLogic Workshop to begin integration development, the methods under each control become available to use on the Workshop work area. You can drag the methods to the Workshop work area to create nodes in the workflow. You can then define the necessary values for sending and receiving data using the method.

The following XML API methods are exposed by each control. The methods are listed in the order that appear in java file that defines the controls.

Customer Management API

- importCustomerAccount
- getCustomerAccountByKey
- deleteCustomerRequest

Order Management API

- queryOrderManagementRequest
- startOrderByKeyRequest
- updateOrderManagementRequest
- getOrderByKeyRequest
- createOrderByValueRequest
- assignProvisionPlanProcedureRequest
- getActivationDataByKeyRequest
- transferTaskRequest
- updateE911DataRequest
- getE911DataRequest
- updateEstimationCompletedDateRequest
- addTaskJeopardyRequest
- getTaskDetailRequest

- taskJeopardyRequest
- getPSROrderByTN
- processSuppOrder
- getCNAMDataRequest
- getLIBDDataRequest
- updateCNAMDataRequest
- updateLIBDDataRequest
- reopenTaskRequest
- createAttachment
- createOrderRelationshipRequest

Inventory Management API

- createEntityByValueRequest
- getServiceRequestDLRsValue
- getEntityByKeyRequest
- updateEntityByValueRequest
- queryInventoryManagementRequest
- updateTNRequest
- tnRecall
- tnValidationRequest
- auditTrailRecording
- getNetworkAreasByGeoAreaRequest
- getNetworkComponentRequest
- getIpAddressesRequest
- inventoryAssociationRequest
- createNewInventoryItemRequest
- queryNetworkLocation
- queryEndUserLocation
- getLocationRequest
- deleteLocationRequest
- updateLocationRequest
- createLocationRequest

Network Resource Management API

- getAvailablePhysicalPortsRequest

Service Order Activation API

- createSOAMessageRequest
- getSoaTnsForOrderRequest
- getSoaDefaultsRequest
- getSoaInformationRequest
- getSoaMessageToSendRequest
- setTnSoaCompleteRequest

LSR Management API

- getLRByKeyRequest
- getDLByKeyRequest
- getLSRByKeyRequest
- getLSRCMByKeyRequest
- createDSCNByValueRequest
- createDSREDByValueRequest
- createLRByValueRequest
- createLSRCMByValueRequest
- createNPLSRByValueRequest
- queryCCNARequest
- queryLSRRequest
- queryLSRForPONCCNAVERRequest
- queryPONSForCCNARequest
- createLSROrderByValueRequest

Event Management API

- updateInboundEventStatus
- getEventStatus
- updateOutboundEventStatus

Viewing controls in WebLogic Workshop

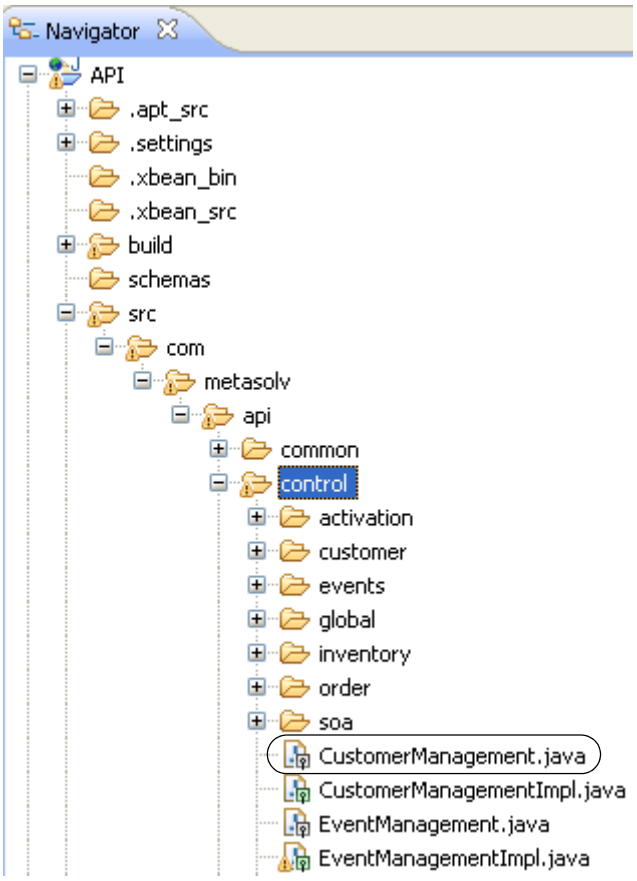
The following figures show how a control and the methods that the control exposes for the corresponding XML API display in WebLogic Workshop. The CustomerManagement control is used as an example, so the figures show the CustomerManagement control, and the methods that the CustomManagement control exposes for the corresponding XML API.

To view a control in WebLogic Workshop:

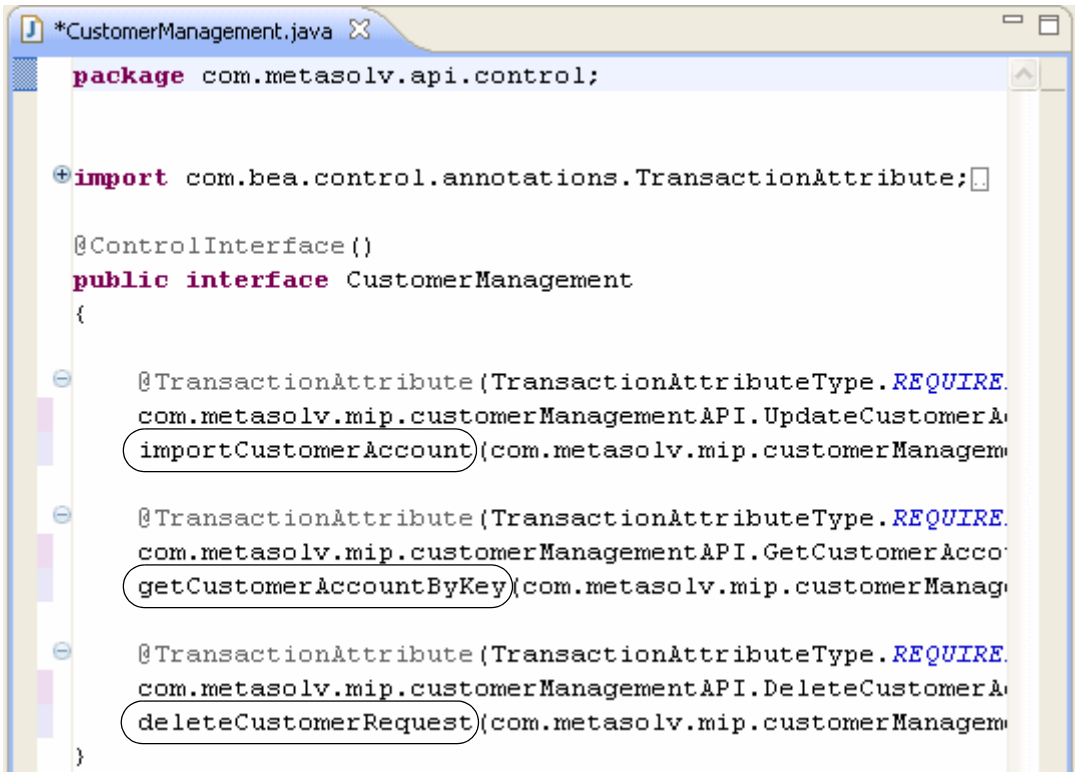
1. In the Navigator view, navigate to the **control** directory.

   In this example, the **control** directory is located in the **API/src/com/metasolv/api/control** directory path.

2. Open a file in the **control** directory, such as **CustomerMangagement.java**.



CustomerManagement.java defines three controls: importCustomerAccount, getCustomerAccountByKey, and deleteCustomerRequest, as shown below.



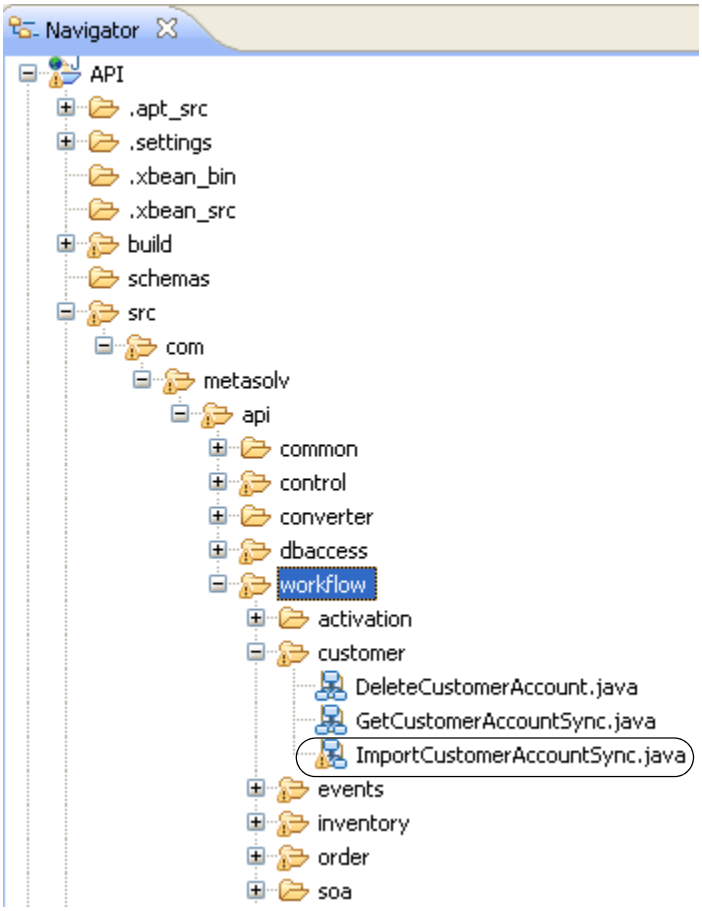
To view the methods that a control exposes for the corresponding XML API:

1. In the Navigator view, navigate to the **workflow** directory.

   In this example, the **workflow** directory is located in the **API/src/com/metasolv/api/workflow** directory path.

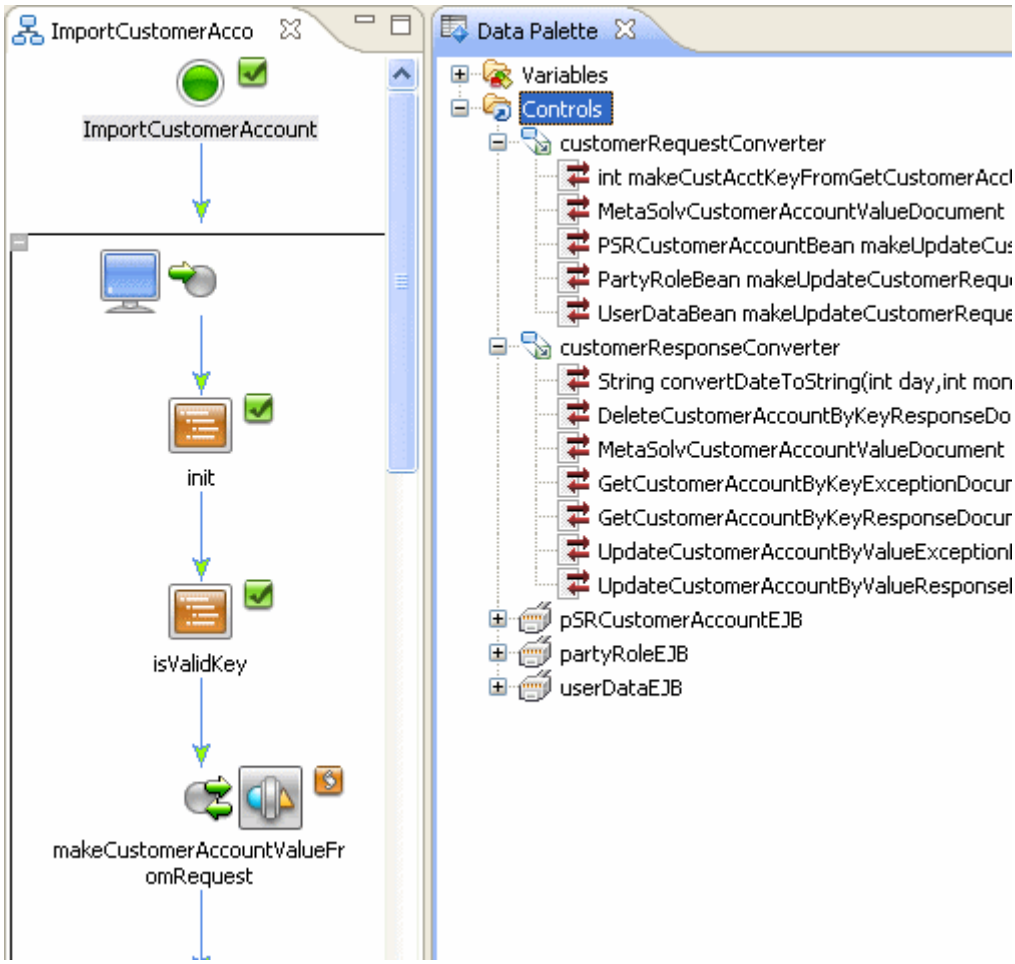
2. Open a file in the **workflow** directory, such as **ImportCustomerAccountSync.java**, which is located in the customer subdirectory.

   Notice that the customer workflow names correspond to the CustomerManagement control names of import, get, and delete.



3. After you have opened the workflow, open the **Data Palette** view by selecting **Window**, then **Show View**, then **Data Palette**.

   The **Data Palette** view displays the XML API methods that work with the control:



Each control has corresponding XSDs that define the XML format that can be received by MetaSolv Solution and show the data fields.The XSDs are constructed using the MetaSolv Solution Information Model (MIM), a dictionary of terms used to standardize data being imported to or exported from MetaSolv Solution using XML.

For a description of the MetaSolv Solution XML API methods, see “Appendix C: XML API Methods” on page 115.

MetaSolv Solution schema

Schema, also known as XSDs, are documents that define how XML must be formatted when it is sent to MetaSolv Solution as data input. Where applicable, it also defines how XML will be formatted when MetaSolv Solution returns data. MetaSolv Solution XSDs contain documentation that explains what values are expected, where a value appears in the MetaSolv Solution user interface (where applicable), and the purpose of fields included in the XSDs.

The schema files are housed in two JAR files:

- MetaSolvSolutionUtility.jar

  This file contains the XSD files that define the MetaSolv Solution schema. This file must be pulled into your workspace, as described in "Adding the MetaSolv Solution controls to Workshop". This file is located in the *<INSTALLATION_DIRECTORY>***/mss_samples/EAR_content/APP-INF/lib** directory.

- mss_xml_api_schemas.jar

  This file contains the annotated versions of the XSDs that exist in the **MetaSolvSolutionUtility.jar** file. This file is used for reference purpose only. Previous versions of MetsSolv Solution contained the **MetaSolvSolutionUtil.jar** and **MetaSolvSolutionSchemas.jar** files.

  The **MetaSolvSolutionUtility.jar** file is a combined packaging of the **MetaSolvSolutionUtil.jar** and **MetaSolvSolutionSchemas.jar** files.

The MetaSolv Solution schema files are grouped by function. For example, Customer Management, Order Management, Inventory Management, etc. Each functional group defines three separate XSD files. For example, the Customer Management XML API defines the following three files:

- XmlMetaSolvCustomerManagementAPI.xsd
- XmlMetaSolvCustomerManagementEntities.xsd
- XmlMetaSolvCustomerManagementData.xsd

Similarly, the Order Management XML API defines the following three files:

- XmlMetaSolvOrderManagementAPI.xsd
- XmlMetaSolvOrderManagementEntities.xsd
- XmlMetaSolvOrderManagementData.xsd

Each of the files define a specific type of information:

- The *API.xsd files define the requests and responses that correspond to the defined control methods covered in the previous section of this chapter.

- The *Entities.xsd files define the data structures that are input to requests or output from responses.
- The *Data.xsd files define various data structures that group data together so the data can be referenced as a complex group, rather than by each individual data element. These data structures are commonly referenced from within an entity data structure. Think of the data structures defined in these files as sub-structures; specifically, the data structures defined in these files are not input to requests or output from responses.

For detailed information on navigating the XSD files, refer to "Appendix B: Navigating The XSD".

The following figure shows the Customer Management schema displayed in an XML editor. Notice the listing of elements in the window.

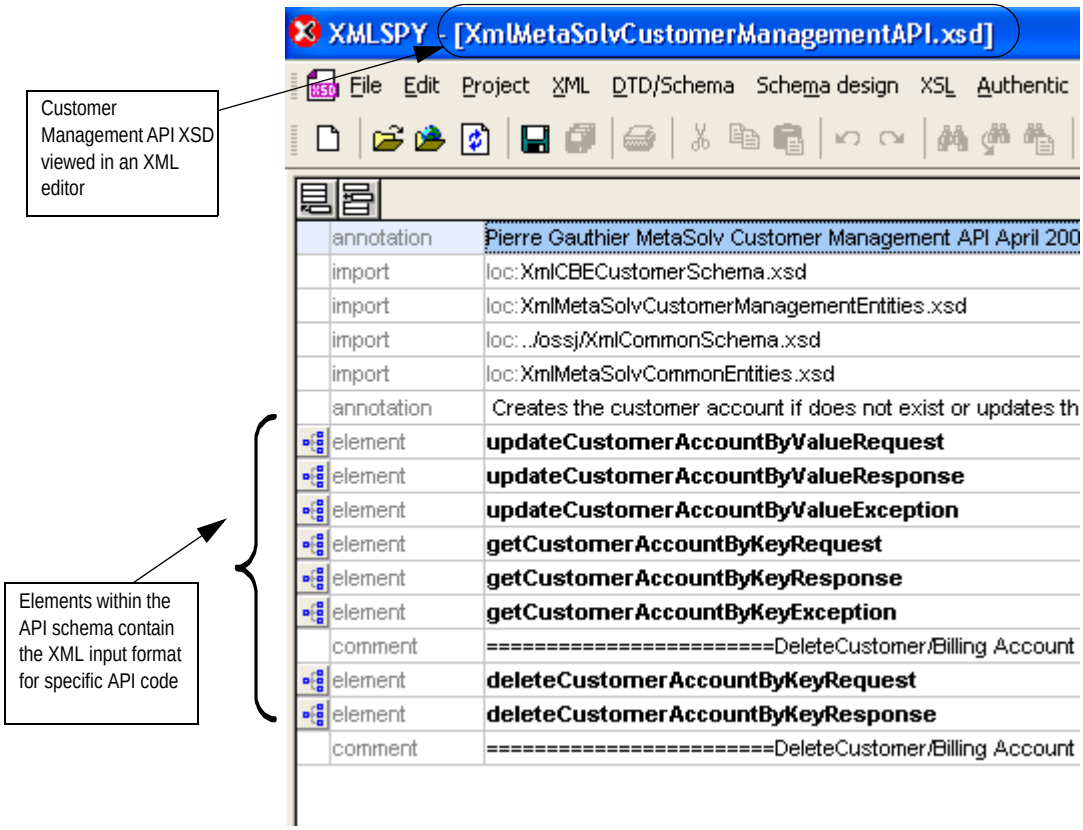


Figure 5: Customer Management API methods displayed in an XML editor (XMLSpy)

The following figure shows *getCustomerAccountByKeyRequest* opened and displayed on the screen.

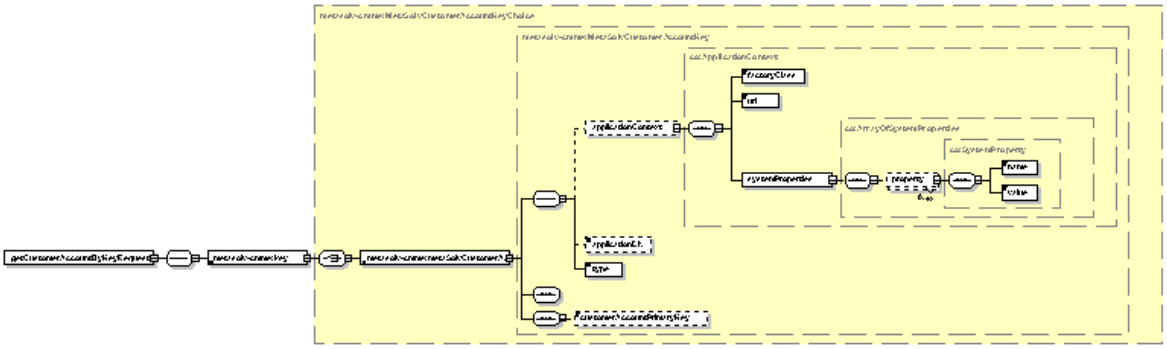


Figure 6: Graphical view of the schema for a method in XMLSpy

The schema include documentation on the information they contain. The following figure shows the graphical representation of an element's schema with documentation highlighted.

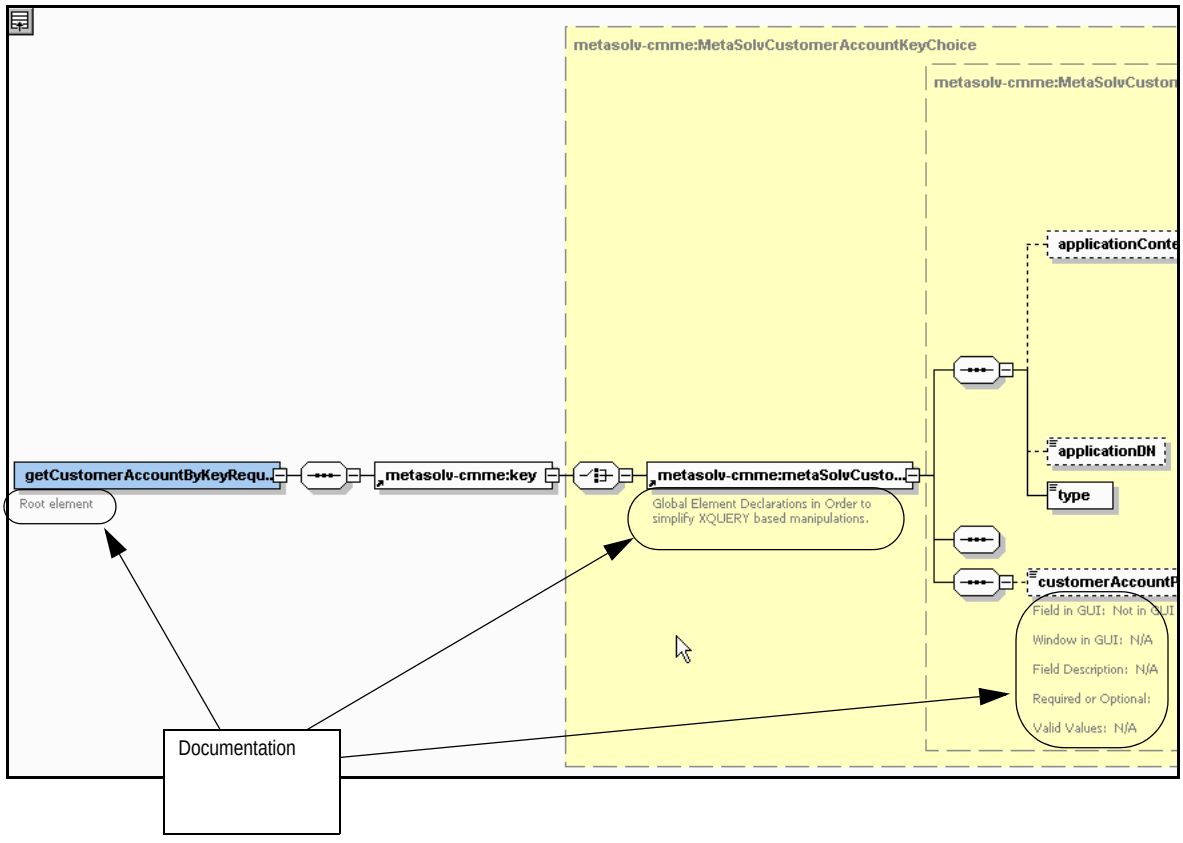


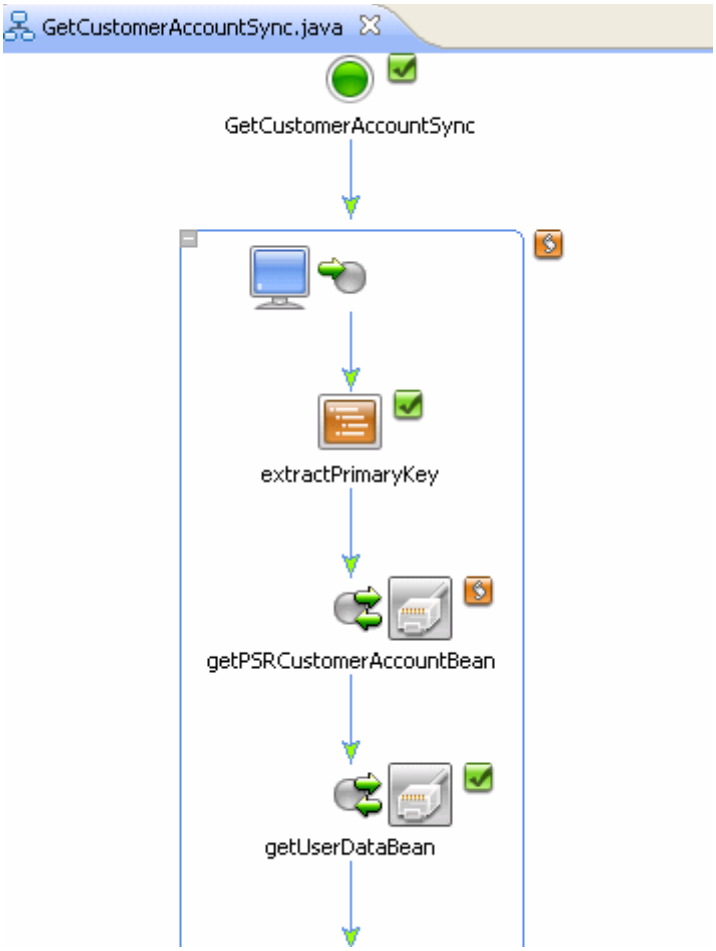
Figure 7: Schema documentation shown in a graphical view in XMLSpy

Oracle WebLogic 10.3.1

Oracle WebLogic 10.3.1 provides the environment for the MetaSolv Solution integration process. WebLogic is a powerful software application that has many uses beyond the scope of this document. This document documents only those portions of WebLogic and its software components that relate directly to accomplishing a task in the integration of MetaSolv Solution. See the Oracle WebLogic documentation at:

http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html

WebLogic contains a full-featured integrated development environment (IDE) that you can use to create and debug your application. WebLogic Workshop provides the tools to automate much of the coding that is required for development. The following figure shows a Workshop workflow, which is the graphical representation of steps that combine methods and data transformations to accomplish a task in MetaSolv Solution.



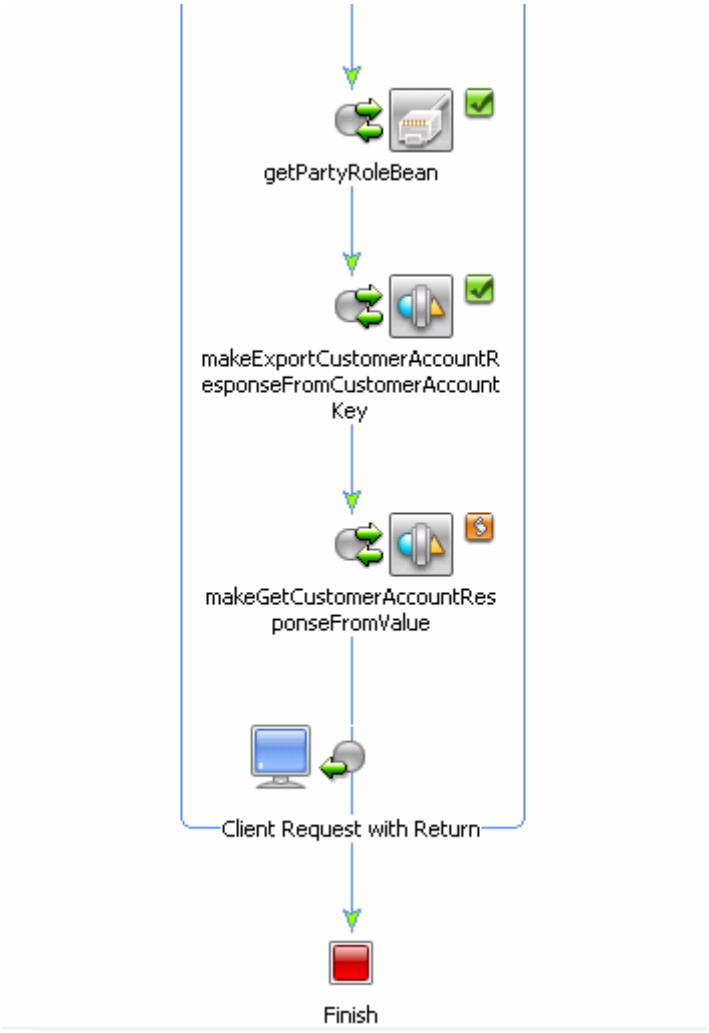


Figure 8: A WebLogic Workshop workflow for getting customer information

To integrate MetaSolv Solution with another application, you must use the following WebLogic components:

- **WebLogic Workshop**

  This component provides an integrated development, deployment, and run-time environment for building applications. The procedures for developing a workflow and the MetaSolv Solution samples will be described using Workshop in this document.

- **WebLogic Integration**

  This component provides a framework for developing and integrating applications and business processes from within and across an enterprise.

Basic integration steps

The following steps show the high level process for integrating MetaSolv Solution with a third-party application. Some steps can be performed in a different order, but the order shown here is recommended by Oracle as a best practice.

1. Layout the steps in your project to determine any data mapping that must be done between your schemas and the MetaSolv Solution schemas.

   This means identifying the nodes, or building blocks, in what will become the workflow for the application.

2. Identify the sources and targets for all data mapping that you identify between the schemas.

   This step includes both requests into MetaSolv Solution and the responses that are returned to your external system.

3. Build the transformations in WebLogic Workshop.

4. Build the workflow for the application in WebLogic Workshop.

5. Build the framework.

   This step is optional. It includes setting up logging and error handling.

6. Test the workflow.

   Some basic test capabilities are included in WebLogic Workshop. You can also use the connector included in the mss_samples.jar file to simulate receiving and processing data from an external application in WebLogic Workshop.

7. Create an ear file that contains the application and deploy it with MetaSolv Solution.

Special characters

The XML API supports the special characters listed in the table below. The first five rows show special characters that are recognized by the API as an entity other than the special character itself. The remaining rows list special characters that are recognized by the API in the same manner as the GUI.

Table 1: Special characters supported by the XML API

Special character	API	GUI
’	&apos;	’
"	&quot;	The " must be last character.
&	&amp;	&
<	&lt;	<
>	&gt;	>
~	~	~
!	!	!
@	@	@
#	#	#
$	$	$
%	%	%
^	^	^
*	*	*
(	(	(
)	)	)
{	{	{
}	}	}
[	[	[
]	]	]

3

Developing An Integration Application

This chapter provides information to help get you started developing applications using the MetaSolv Solution Integration and Portal Toolkit. The information is presented through a simple example that shows how to use the tools in the toolkit. When you understand the tools, and you have a clear determination of what you want to accomplish, you are ready to begin developing applications.

This chapter also provides information on migrating from a previous release of MetaSolv Solution. The information is presented at end of this chapter. See "Migrating To MetaSolv Solution 6.2.1".

In this chapter, you will be shown how to accomplish the major steps in creating and deploying a MetaSolv Solution integration application using the *GetCustomerHttpSample* included in *mss_samples.jar*. The sections in this chapter follow the basic steps for creating an application for MetaSolv Solution. To create an application for MetaSolv Solution in WebLogic Workshop, you must:

1. Plan the application.
2. Create a new (empty) application in WebLogic Workshop.
3. Import the appropriate MetaSolv Solution controls into the new workflow in Workshop.
4. Create data transformation controls for your incoming XML data and for the outgoing data you expect to receive in response.
5. Create the workflow using MetaSolv Solution controls, the transformation controls you created, and generic controls inside Workshop.
6. Set up logging and exception handling for the application.
7. Test the workflow.
8. Create and deploy the application .ear file.

All of the steps listed are explained in the following sections. The sections describe the process for creating the *GetCustomerHttpSample* included in *mss_samples.jar*. For information on how to locate the samples file, see "Appendix A: XML API Sample Code" on page 89. The *GetCustomerHttpSample* is simple and easy to understand, but it contains processes that illustrate how controls are used to integrate MetaSolv Solution with another system.

The input for the *GetCustomerHttpSample* comes from the *Cim_customer.xsd*, which is also included in Samples.jar.The details of the example in this chapter may vary slightly from the *GetCustomerHttpSample* in the *mss_samples.jar* file. This is done to simplify the example for illustration purposes.

This figure shows where the **GetCustomerHttpSample.java** file is located inside the mss_samples.jar file after the contents are extracted and opened in a workspace. For more information on the samples in the **mss_samples.jar** file, see "Appendix A: XML API Sample Code" on page 89.

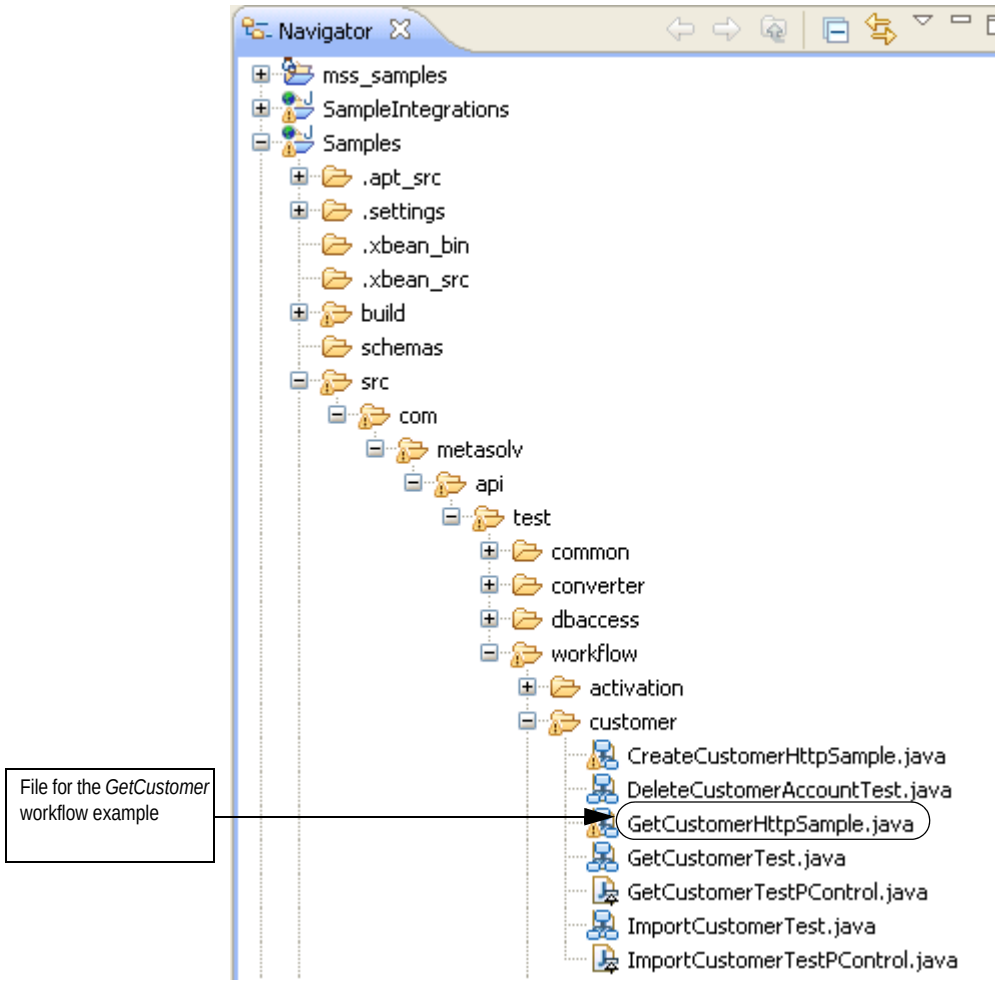


Figure 9: Directory structure for mss_samples

When you open GetCustomerHttpSample.java in Workshop, the following workflow displays.

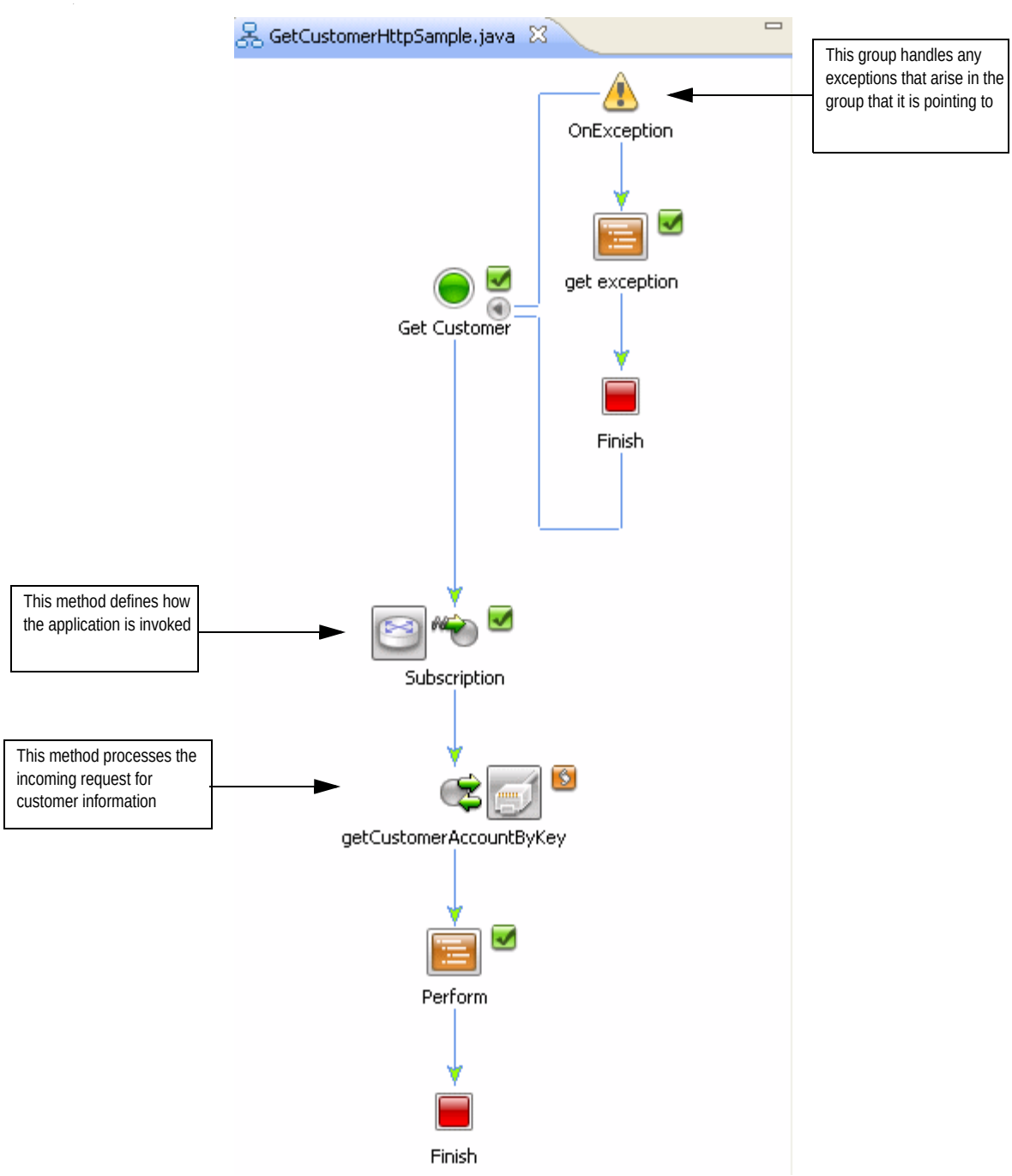


Figure 10: Workflow for getCustomerHttpSample.java

The following figure shows a graphical view of the schema for Cim_Customer, which provides the format for the incoming XML for the *GetCustomerHttpSample* shown in this chapter.

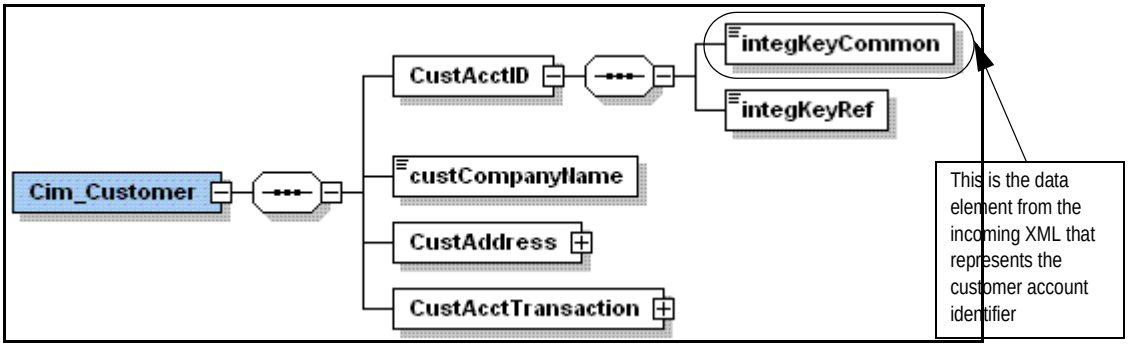


Figure 11: Schema for the incoming request XML

Planning the application

This section describes how to plan for your application. Oracle recommends as a best practice that you list out the information you need before you begin work on the application in Workshop. The information you need includes:

- The WebLogic domain to which the application will be assigned in Workshop. You can use an existing domain or create a new domain.
- MetaSolv Solution controls you will require.
- A list of the data that needs to be transformed from your XML format to the MetaSolv Solution MIM format.

Here is the information needed for the *GetCustomerHttpSample* used in this chapter.

MetaSolv Solution controls required:

MetaSolv Customer Management API

Transformations required:

Incoming XML: *customer account identifier*

Outgoing XML: *customer account identifier, customer's company name*

The output was limited to two data items to keep the example simple. Although this example uses simple inputs and outputs, the transformations for a normal integration effort can become complex and therefore requires careful planning. The problem of knowing which data from your XML maps into which data in the MetaSolv Solution schema is eased by the documentation in the MetaSolv Solution schema. See "MetaSolv Solution schema" on page 19 for more information about the MetaSolv Solution schemas.

Accessing WebLogic Workshop

The following procedures are all performed within WebLogic Workshop.

To access WebLogic Workshop:

1. Perform either step a or step b:
   a. For Windows:

      From the **Start** menu, select **Programs**, then **Oracle WebLogic**, then **Workshop for WebLogic 10gR3**.

   b. For UNIX:

      $BEA_HOME/workshop_10.3/Workshop.sh

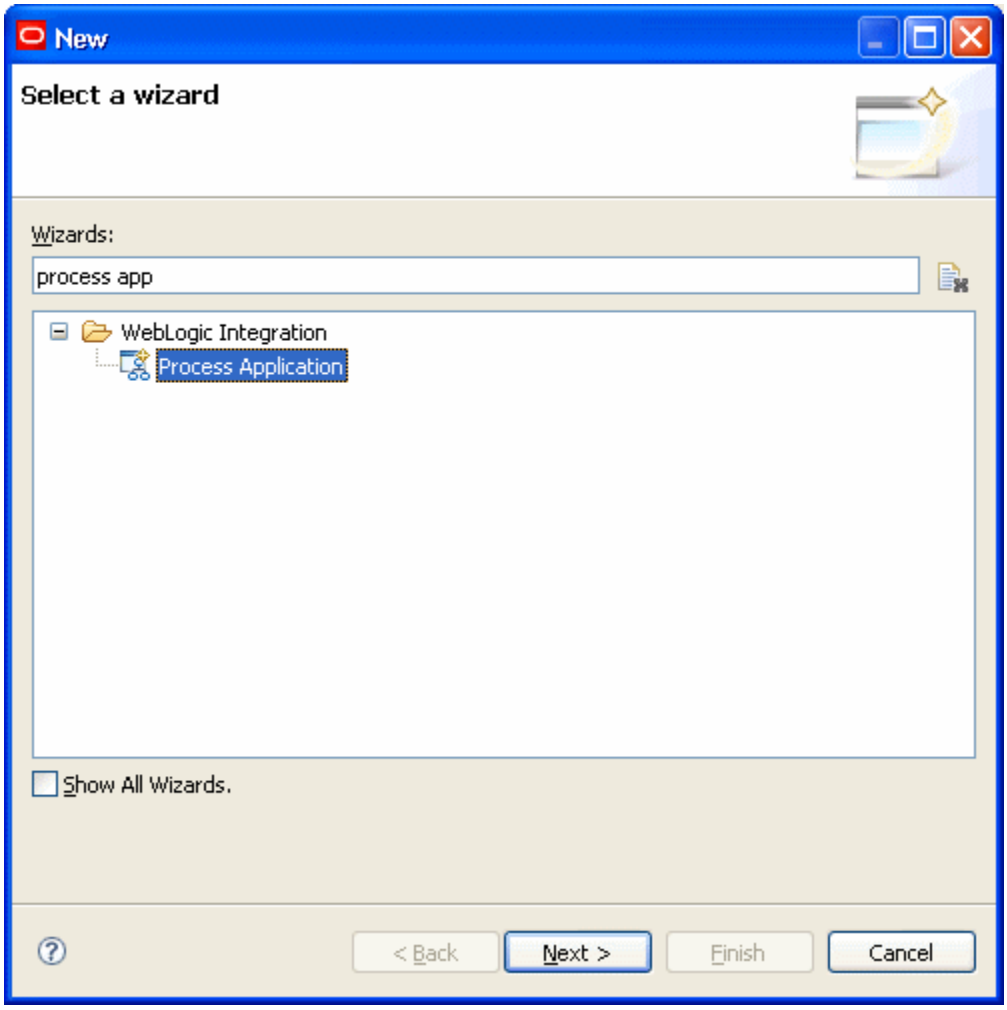
      If this is the first time starting WebLogic Workshop, or if Workshop was previously open on an application from another WebLogic domain, you might see a warning about fixing the domain. If so, click **Continue** to ignore the warning, then close the open application.

Creating a new application in Workshop

To create a new application in Workshop:

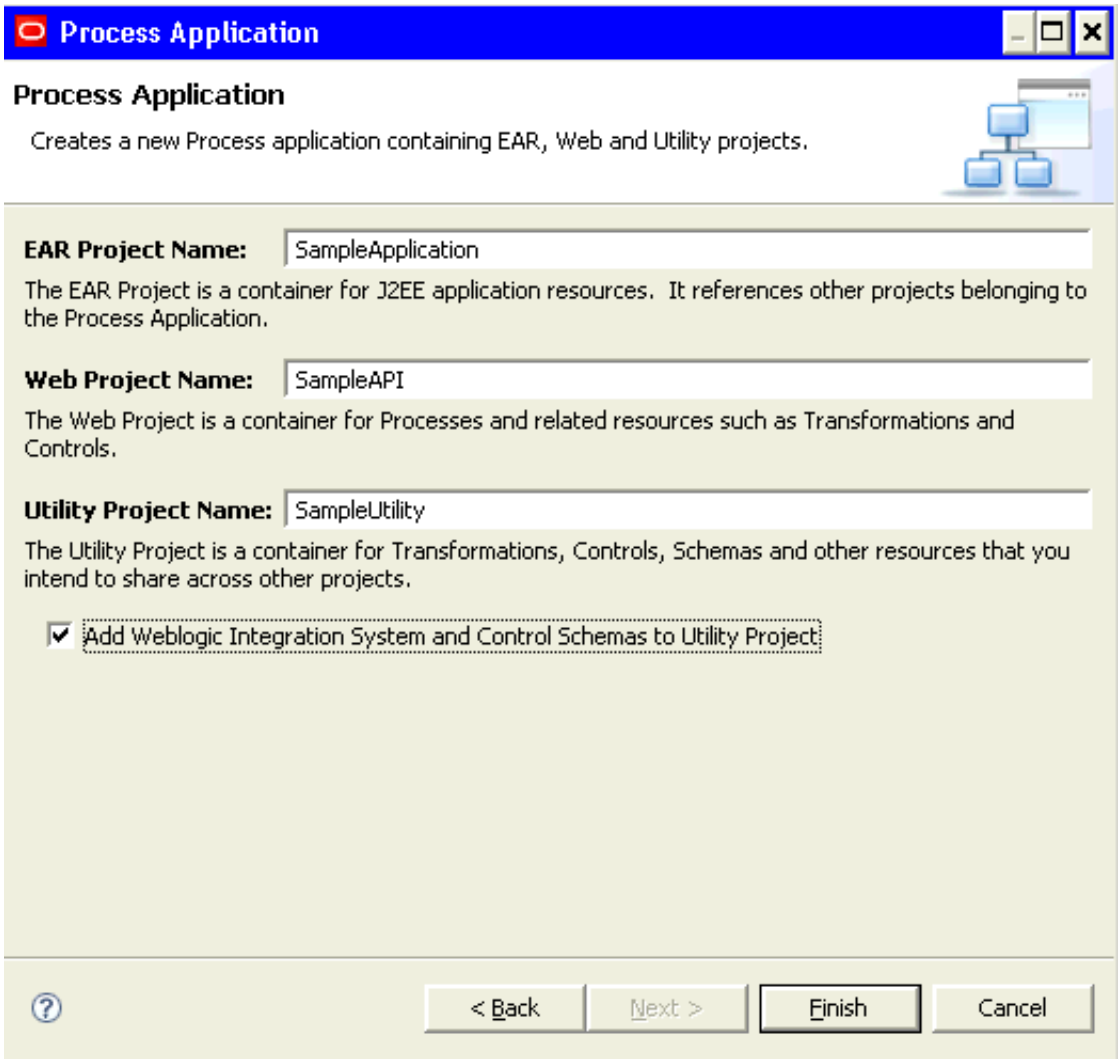
1. From the menu, select **File**, then **New**, then **Other**.

   The Select a Wizard window displays.



2. In the **Wizards** field, enter *process app*.
3. Select **Process Application** and click **Next**.

The Process Application window displays.



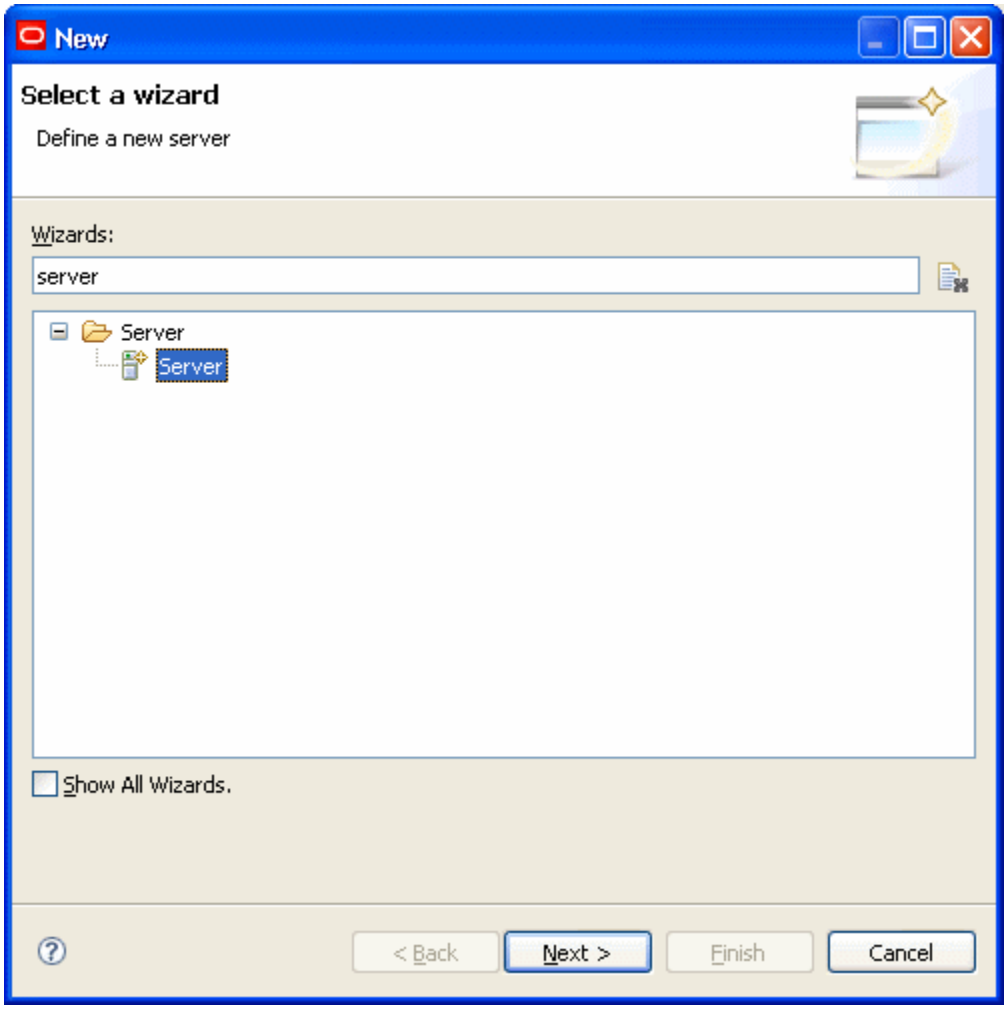
4. In **EAR Project Name**, enter an ear project name such as *SampleApplication*.
5. In **Web Project Name**, enter a Web project name such as *SampleAPI*.
6. In **Utility Project Name**, enter a utility project name such as *SampleUtility.*
7. Select the **Add WebLogic Integration System and Control Schemas to Utility Project** check box.
8. Click **Finish**.

Creating a new server in Workshop

To create a new server in Workshop:

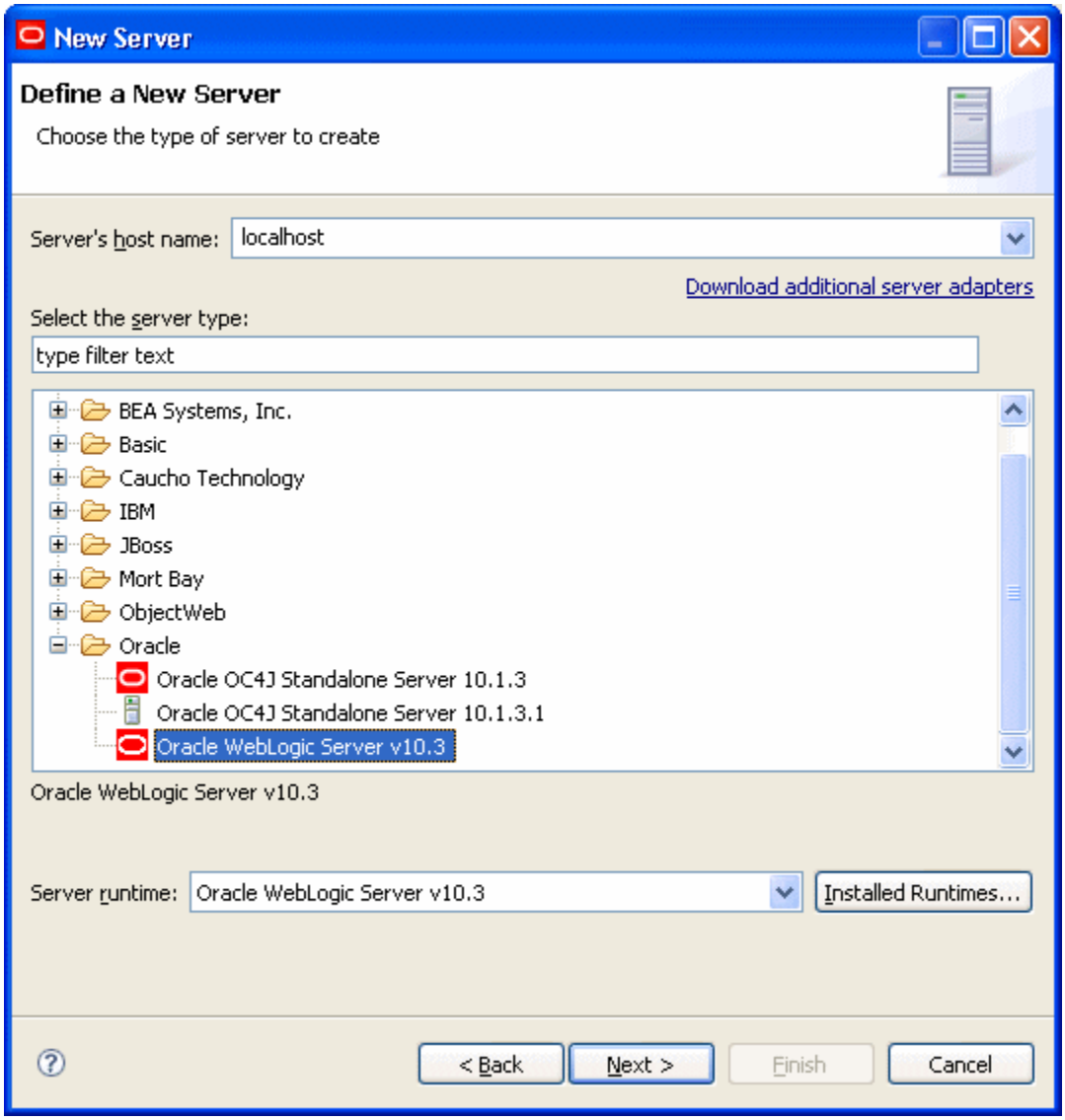
1. From the menu, select **File**, then **New**, then **Other**.

   The Select a Wizard window displays.



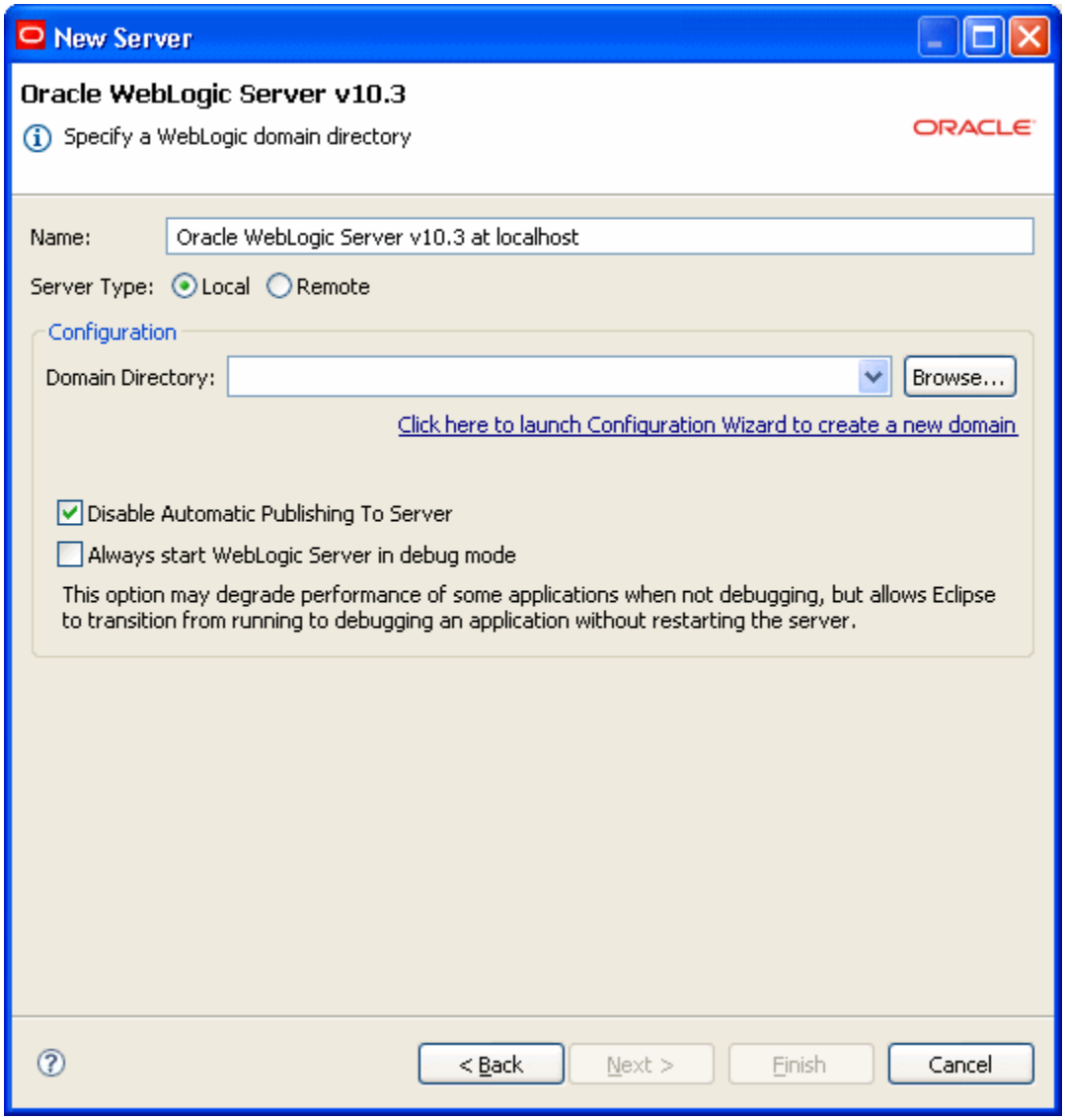
2. In the **Wizards** field, enter *server*.
3. Select **Server** and click **Next**.

The New Server window displays:



4. By default, Oracle WebLogic Server v10.3 is selected. Click **Next**.

The following window displays:



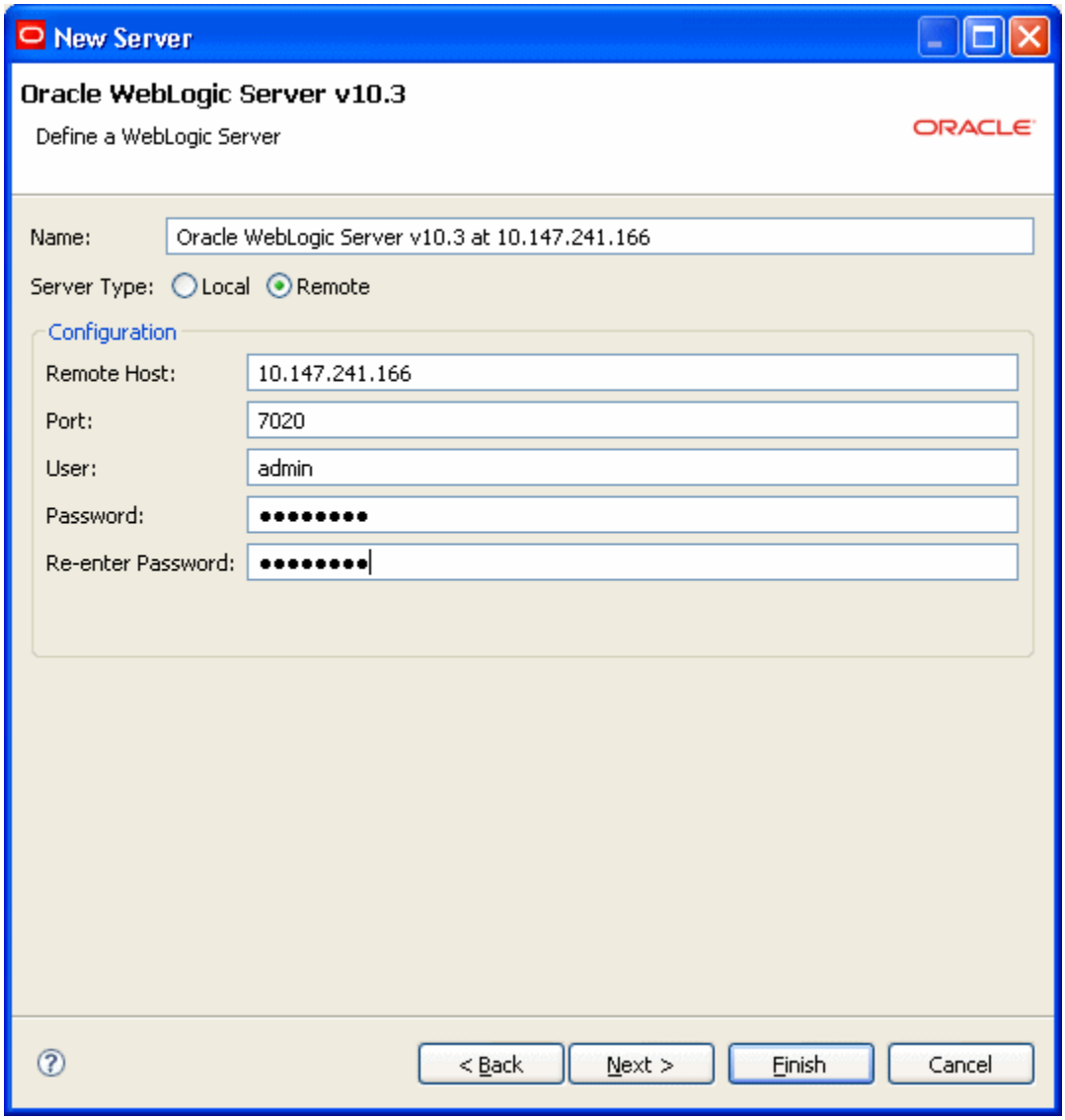
5. Perform either step a or step b, depending on your environment:

   a. For **Server Type**, choose **Local**.

      - In **Domain Directory**, enter the name of the domain directory that is specific to your installation of MetaSolv Solution by clicking **Browse** to navigate to the directory.

   b. For **Server Type**, choose **Remote**.

The fields on the window change based on your selection of **Remote**.



- In the **Configuration** fields, enter the information applicable to your environment.

6. Click **Finish**.

   The server displays in the Servers view. You can start and stop the server from within WebLogic Workshop by right-clicking on the server in the Servers view and selecting the appropriate option.

Adding the MetaSolv Solution controls to Workshop

The MetaSolv Solution controls are located in the **MetaSolvSolutionInterface.jar** file, which is located in the **mss_integration.ear** file.

To add the MetaSolv Solution controls:

1. Locate and open the **mss_integration.ear** file.
2. Extract the **MetaSolvSolutionInterface.jar** file to a local working directory.
3. Copy the **MetaSolvSolutionInterface.jar** file into your Workshop workspace. Specifically, copy the file into the *SampleApplication*/**EarContent/APP-INF/lib** directory path, where *SampleApplication* is the name you provided when creating your application by following the steps in "Creating a new application in Workshop".

Adding the MetaSolv Solution schemas to Workshop

The MetaSolv Solution controls are located in the **MetaSolvSolutionUtility.jar** file, which is located in the **mss_integration.ear** file.

To add the MetaSolv Solution controls:

1. Locate and open the **mss_integration.ear** file.
2. Extract the **MetaSolvSolutionUtility.jar** file to a local working directory.
3. Copy the **MetaSolvSolutionUtility.jar** file into your Workshop workspace. Specifically, copy the file into the *SampleApplication*/EarContent/APP-INF/lib directory path, where *SampleApplication* is the name you provided when creating your application by following the steps in "Creating a new application in Workshop".
4. Extract the **schemas** directory located in the **MetaSolvSolutionUtility.jar** file and add it to the **schemas** folder (**SampleUtility/schemas** directory) of the newly created Utility project.
5. Copy the channel file located in the **src** directory to the **SampleUtility\src** directory.

   Note: After the controls and schemas are added, clean and build the project.

Creating data transformations

The next step in creating an application is the creation of controls for transforming data. The *GetCustomerHttpSample* has data transformations for the request data (incoming) and the response data (outgoing). A transformation file for each set of data must be created. The following figure shows a simple example of the process.

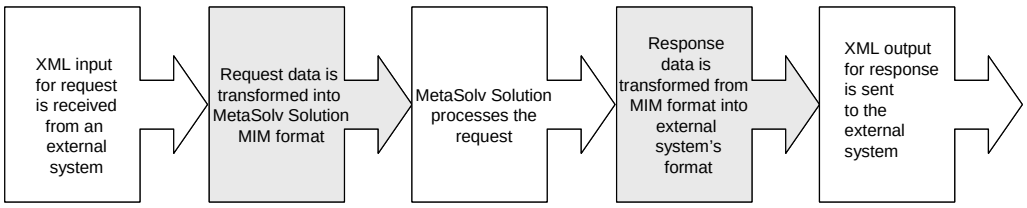


Figure 12: Transforming XML input and output files

The following procedure shows the steps used to create a transformation file for the Cim_Customer example XSD.

Request transformation control

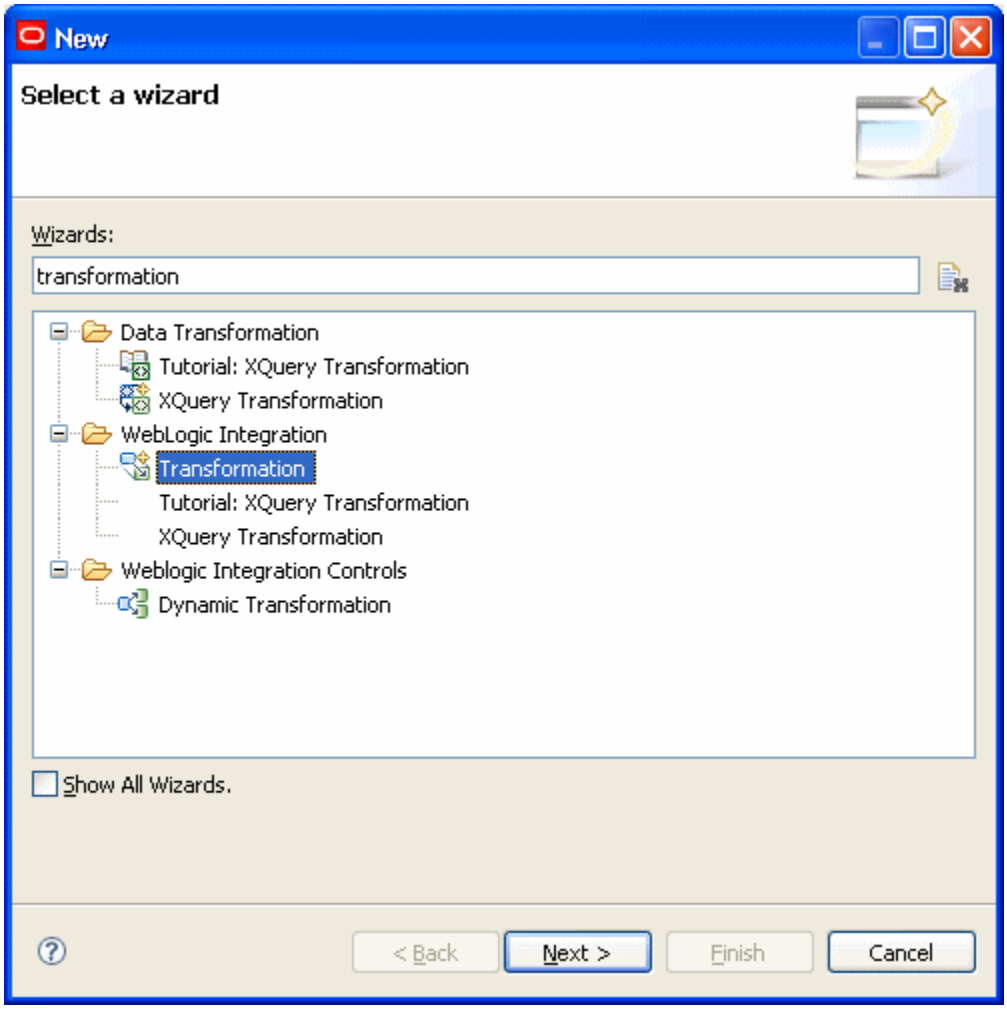
To create a transformation file for the incoming data:

1. In the Navigator view, select a directory for the location of the new transformation file.

   You can create a directory or place the file in an existing directory.

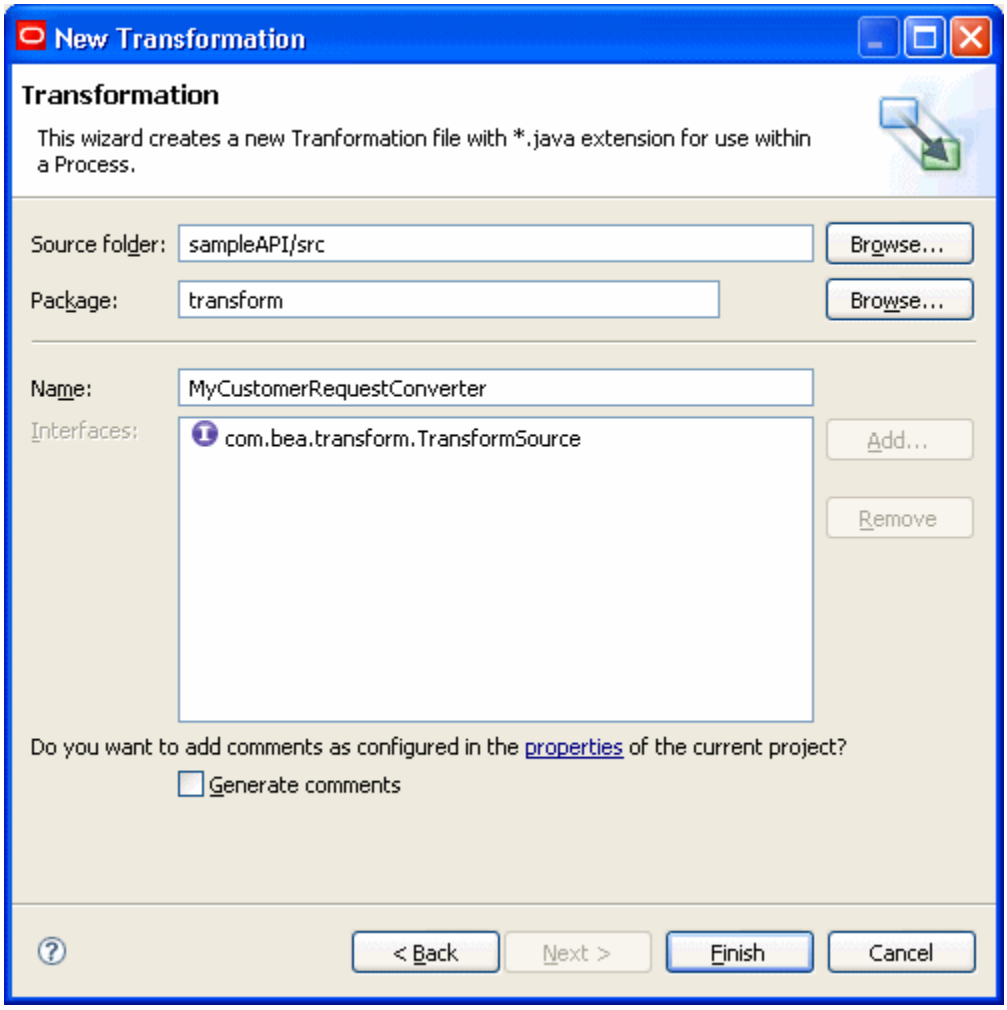
2. From the menu, select **File**, then **New**, then **Other**.

The Select a Wizard window displays.



3. In the **Wizards** field, enter *Transformation*.
4. WebLogic Integration, select **Transformation** and click **Next**.

The New Transformation window displays.

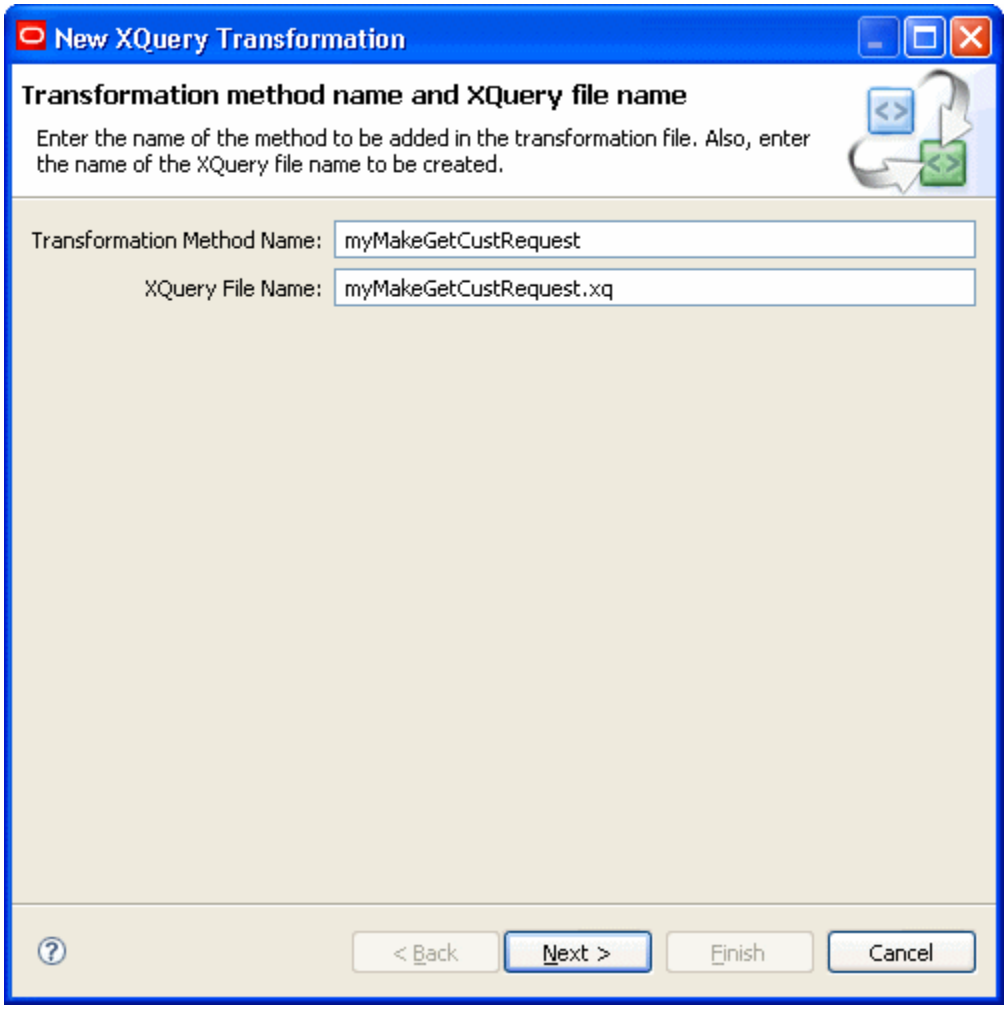


5. In **Name**, enter a name such as *MyCustomerRequestConverter* and click **Finish**.

   This creates a new Transformation file named MyCustomerRequestConverter.java.

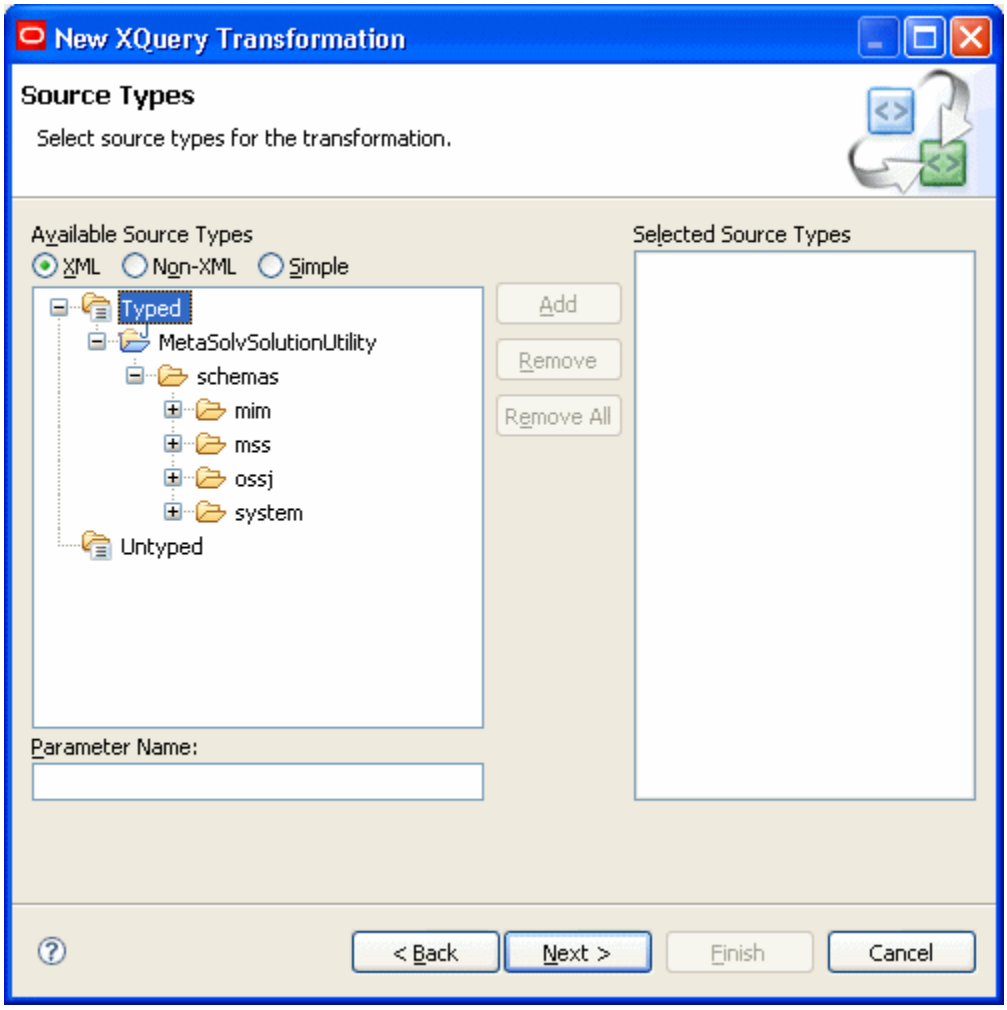
6. In the Package Explorer, right-click on the Transformer file you just created (MyCustomerRequestConverter.java), and select **Transformer**, then **Add**, then **XQueryTransformation Method**.

The New XQueryTransformation window displays.



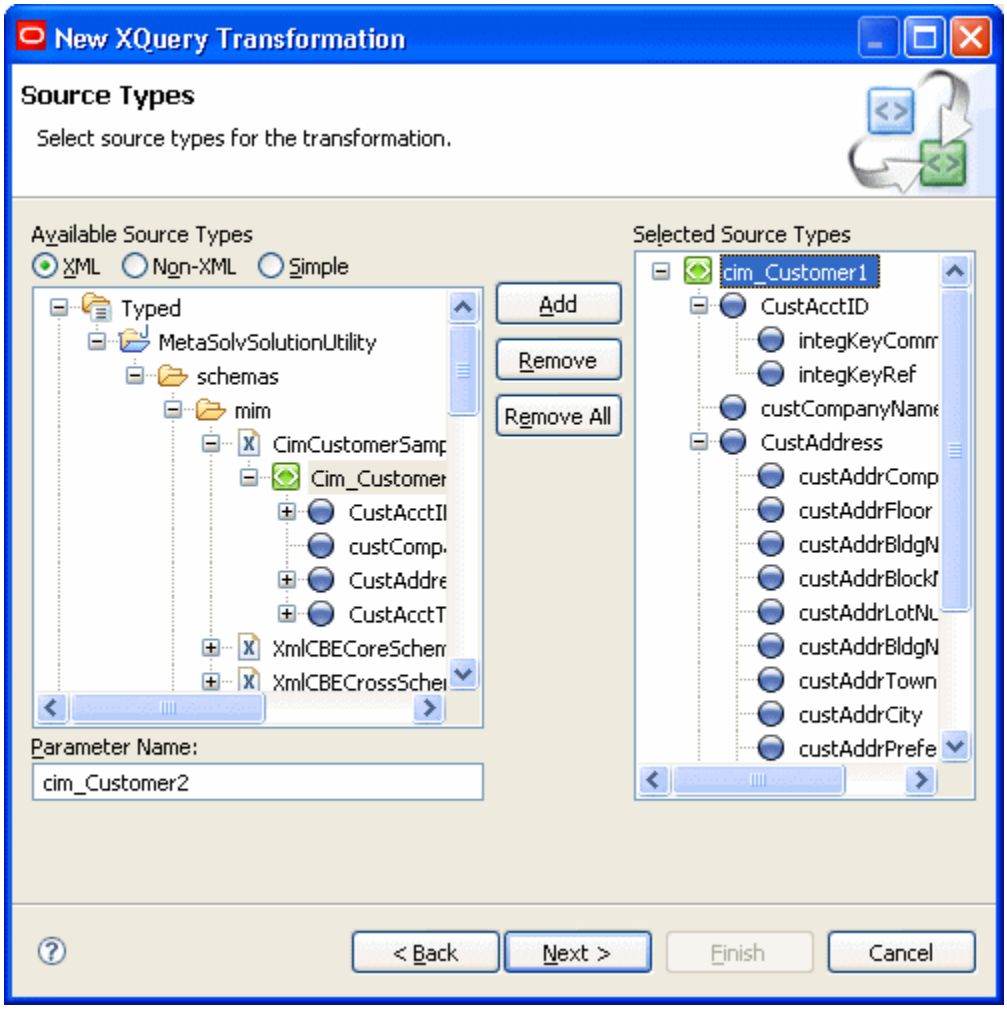
7. In **Transformation Method Name**, enter a name for your method, such as *myMakeGetCustRequest*.
8. In **XQuery File Name**, enter an XQuery file name that corresponds to the name of the method you entered, such as *myMakeGetCustRequest.xq*. (The names you enter should be the same, with the exception of the file extension.)
9. Click **Next**.

The New XQuery Transformation for Source Types window displays.



10. Choose **XML** for the **Available Source Types** option.
11. Expand the treeview to the **schemas** directory.
12. Locate the schema that defines your source data.

    This example shows the mapping of an inbound customer account ID to the customer account ID defined for getCustomerAccountByKeyRequest.



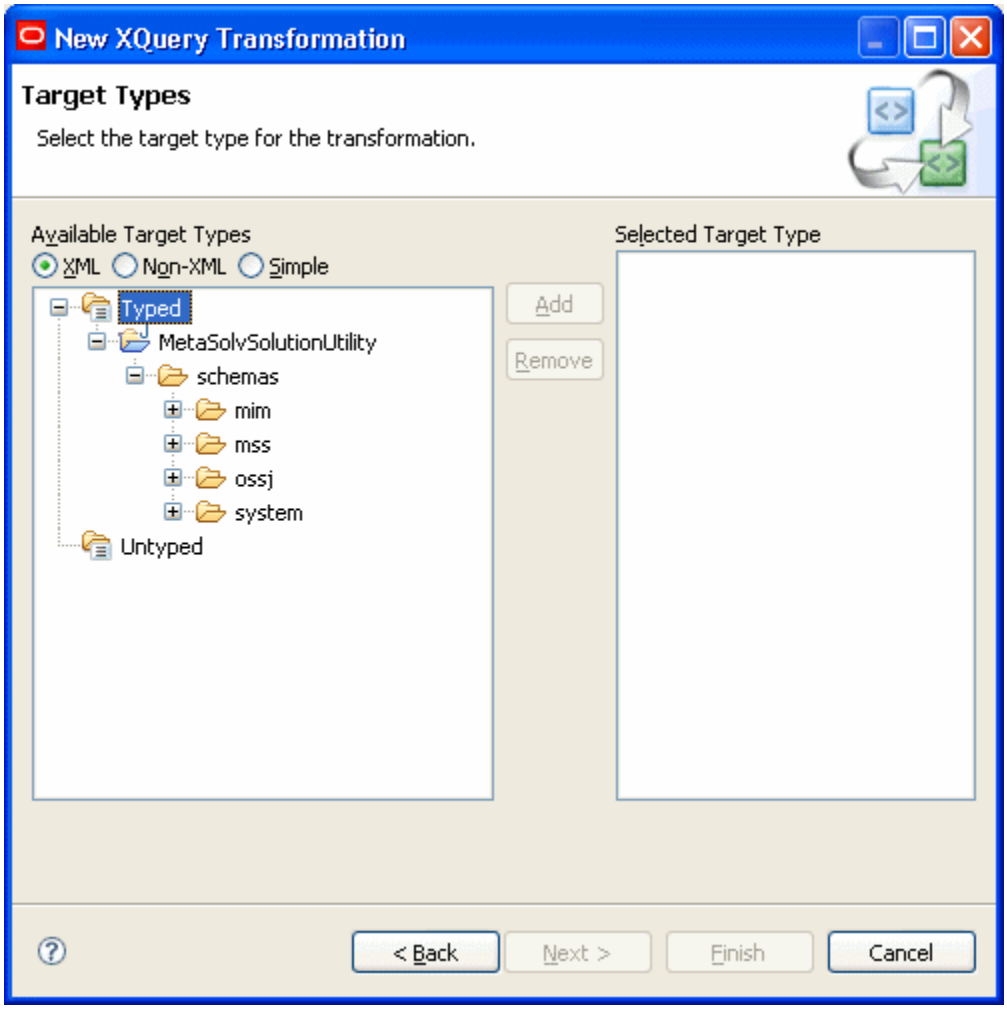
13. From within the schema, select an element, and click **Add**.

    The selected element displays in the Selected Source Types section.

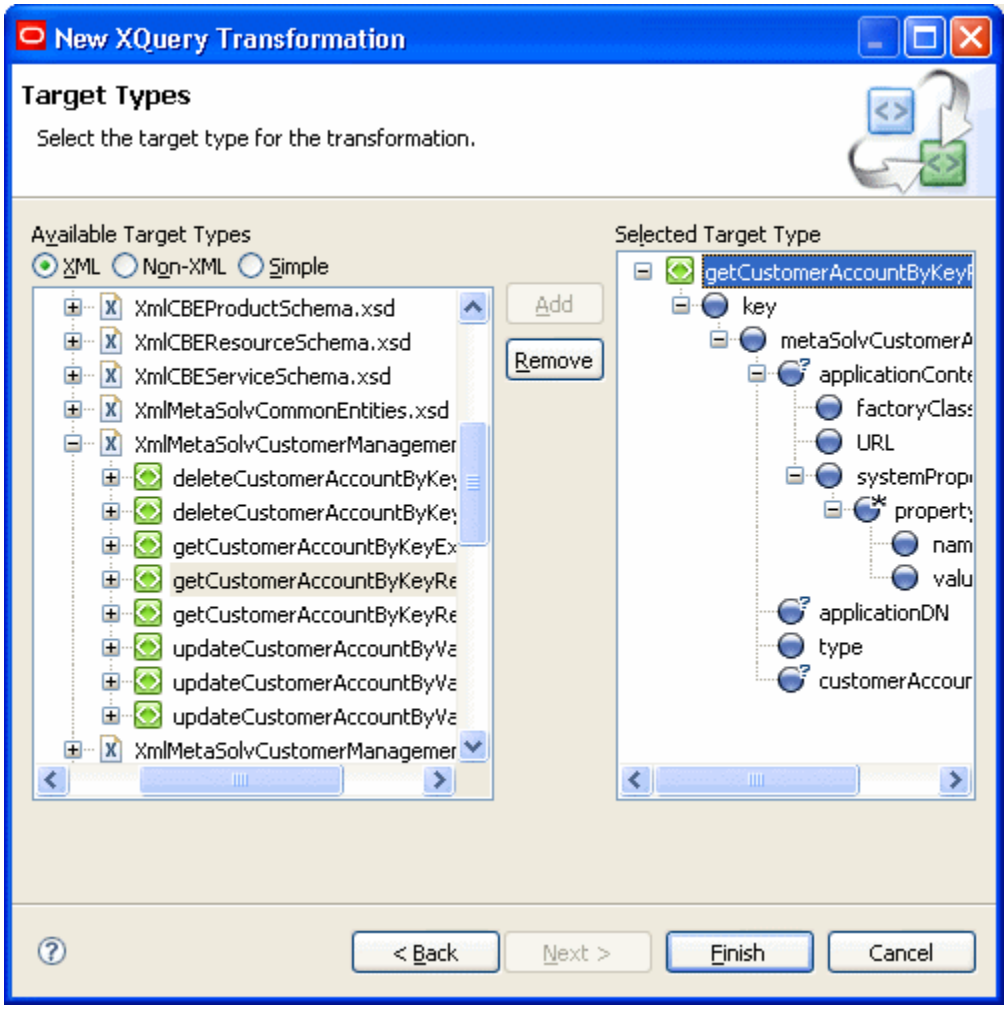
    This example shows the selection of the Cim_Customer1 element defined within the **CimCustomerSample.xsd** file.

14. Click **Next**.

The New XQuery Transformation for Target Types window displays.



15. Locate the schema that defines your target data.



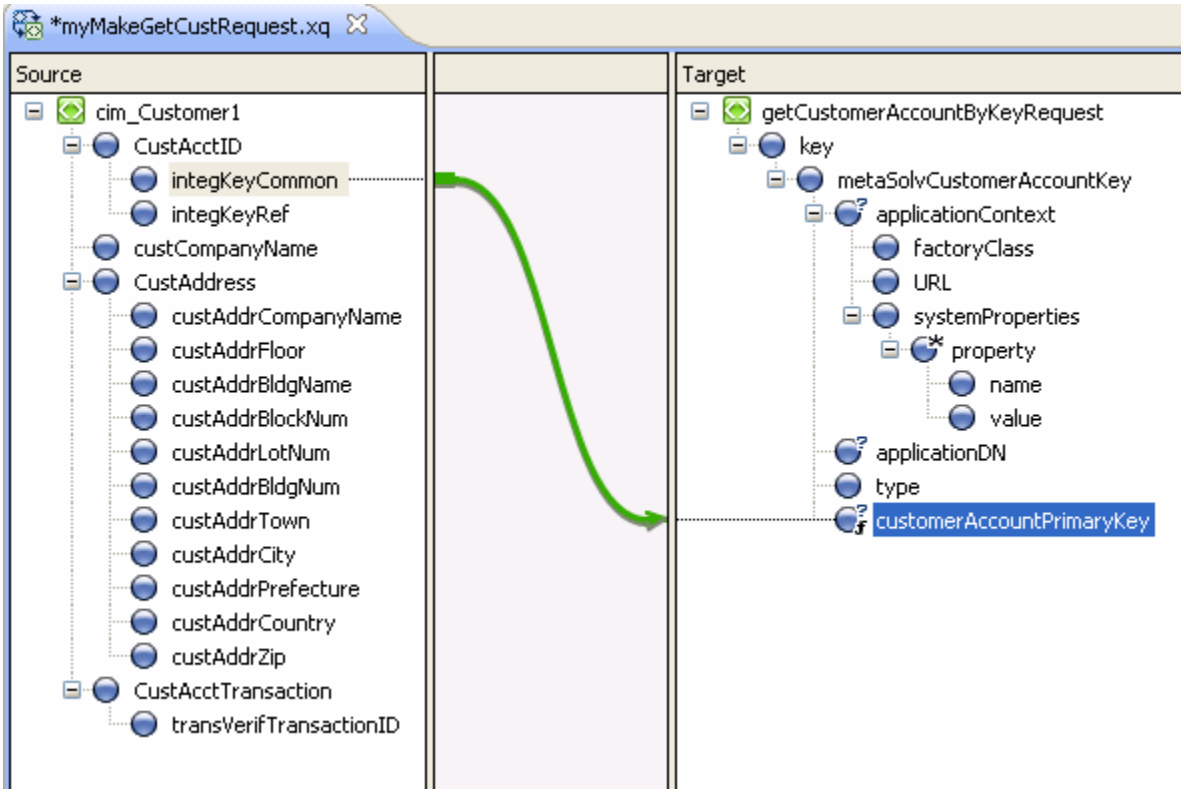
16. From within the schema, select an element and click **Add**.

    The selected element displays in the Selected Target Type section.

    This example shows the selection of the getCustomerAccountByKeyRequest element defined within the **XmlMetaSolvCustomerManagement.xsd** file.

17. Click **Finish**.

The XQuery Transformation that you created displays.



18. Map elements from source to target by dragging and dropping them from the Source section to the Target section.

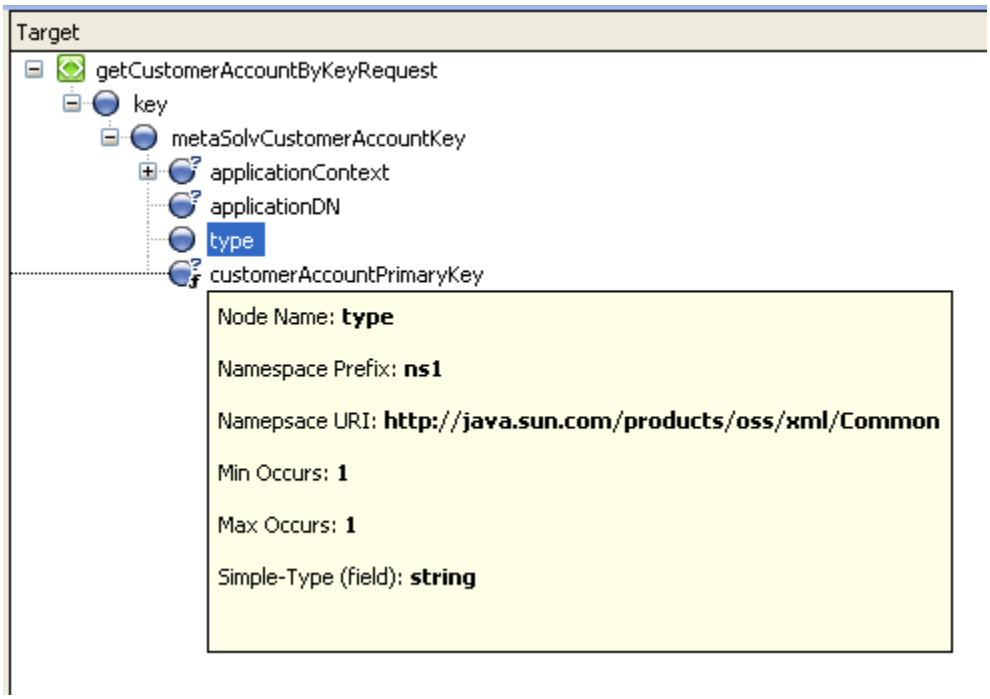
    This example shows the mapping of the source element, *integKeyCommon*, to the target element, *customerAccountPrimaryKey*. In this example, there is only one element to be mapped from the incoming XML because the customer account number is the only data required to export customer information.

Setting a constant value

When using the XML APIs, some elements require a value. If no incoming value is mapped to the elements from the source XML file, you must create a constant value for the element. Continuing with the getCustomerAccountByKeyRequest example, the element **type** requires a value.

You can hover the cursor over an element to see documentation on the element and determine whether a value must be assigned As a best practice, this information should be determined beforehand from the schema using an XML editor. The following figure shows the **type** element with the documentation displayed for the element. If **Min Occurs**

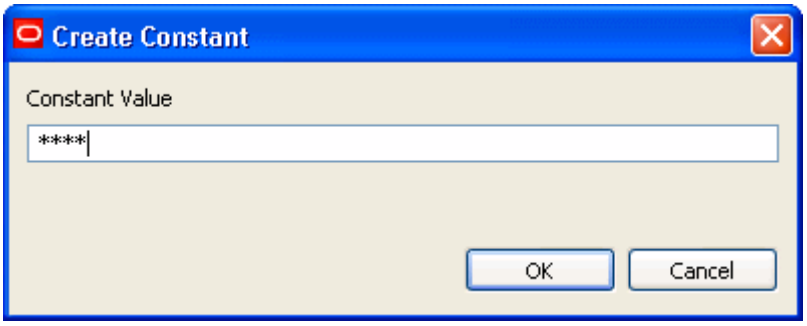
is 0, the value is not required. In this example, **Min Occurs** is 1, indicating that the value is required.



To set a constant value:

1. In the Target section, right-click an element (in this example, **type**) and select **Create constant** to set a value for the element.

   The Create Constant dialog box displays.

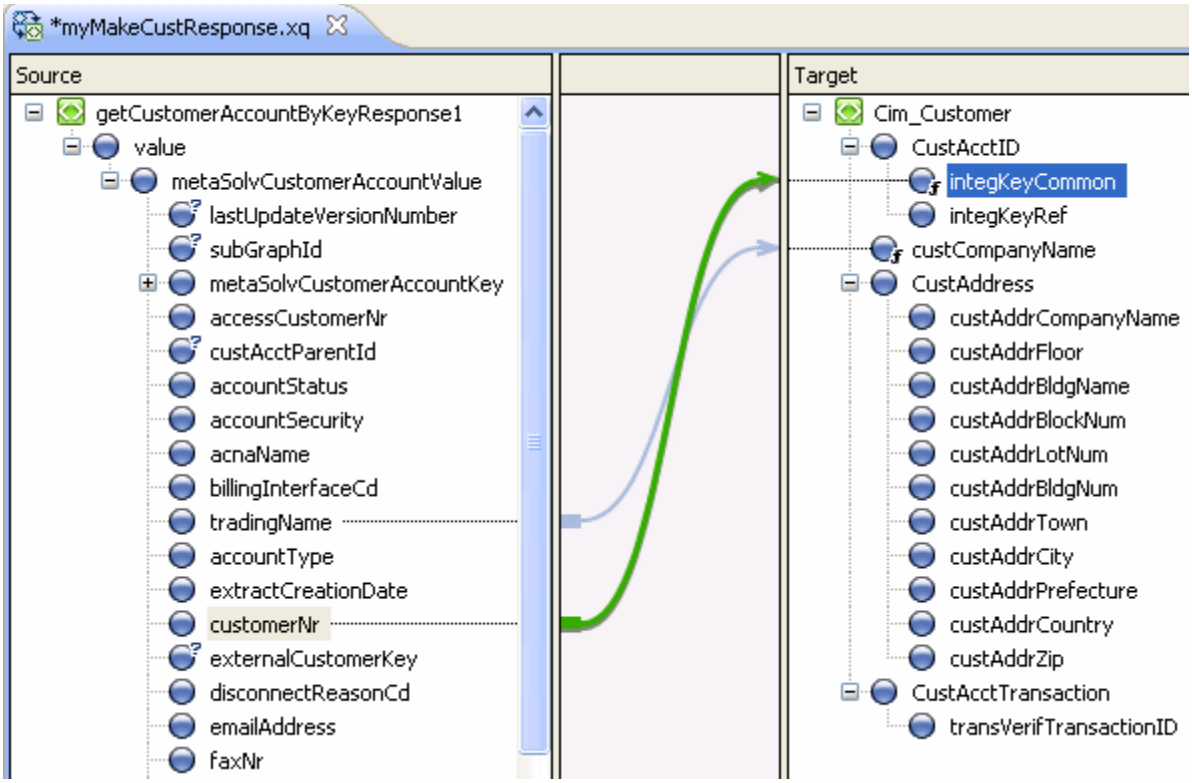


2. Enter "" and click **OK**.
3. Click **Save** and close the XQuery Transformation file.

Response transformation control

The response transformation for the GetCustomerAccountByKeyRequest example is created in the same manner as the request transformation. A new Transformation file is created, a new method is created for the Transformation file, and the source and target elements are selected, but they are selected in the reverse order.

The following figure shows the mapping for the response transformation.Two elements are mapped for the response: customerNr (to integKeyCommon) and tradingName (to custCompanyName).



Building the workflow

This section describes how to create a workflow in Workshop. The workflow contains all of the control and transformation methods necessary to complete the integration tasks you require. Workshop gives you the ability to construct the workflow graphically and generate the code automatically.

Step 1: Creating the workflow process file

Each workflow has a .java process file. This section explains how to create the process file and the workflow in Workshop.

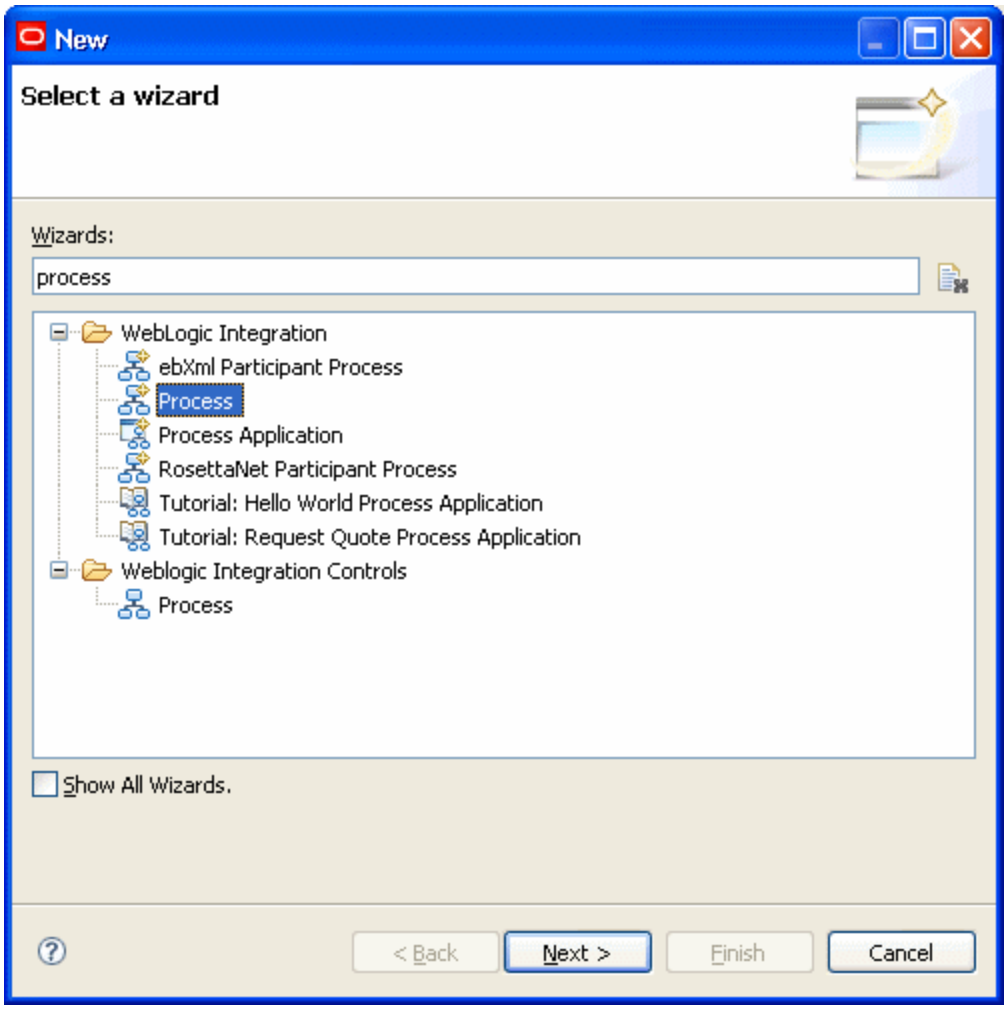
To create a workflow in Workshop:

1. In the Navigator view, select a directory for the location of the new .java process file.

   You can create a directory or place the file in an existing directory.

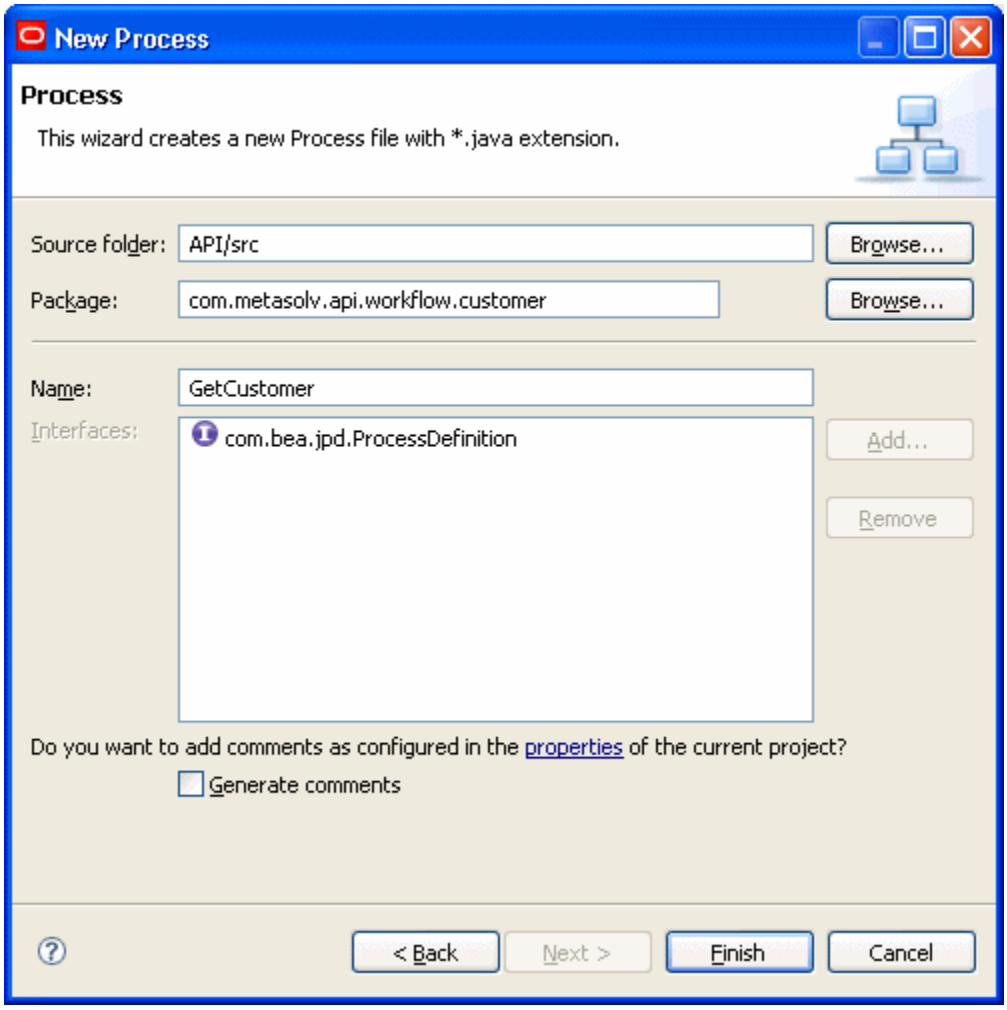
2. From the menu, select **File**, then **New**, then **Other**.

   The Select a Wizard window displays.



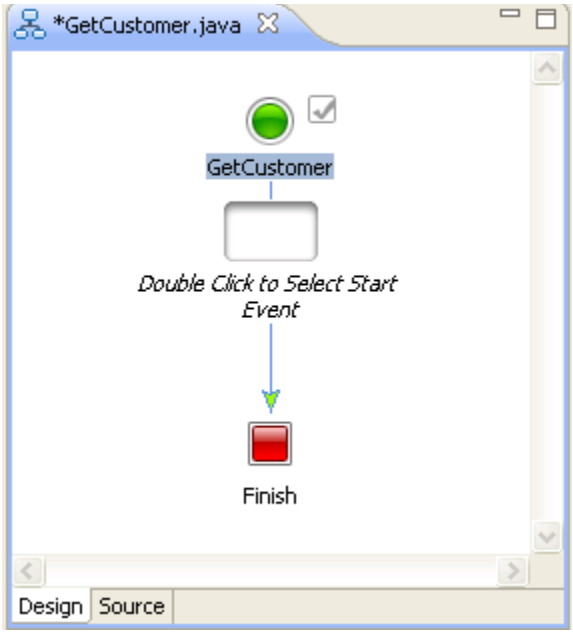
3. In the **Wizards** field, enter *process*.
4. Under WebLogic Integration, select **Process** and click **Next**.

The New Process window displays.



5. In **Name**, enter a name such as *GetCustomer* and click **Finish**.

An empty workflow displays reflecting the name of the process, which in this example is *GetCustomer*. The creation of the process provides the .java file extension.



After an empty workflow is created, you can add the controls to be used in the workflow to the Data Palette as described in the following section.

Step 2. Adding controls to the Workshop Data Palette

This section describes how to add controls to the Workshop Data Palette for use in a workflow.

Adding controls to a workflow

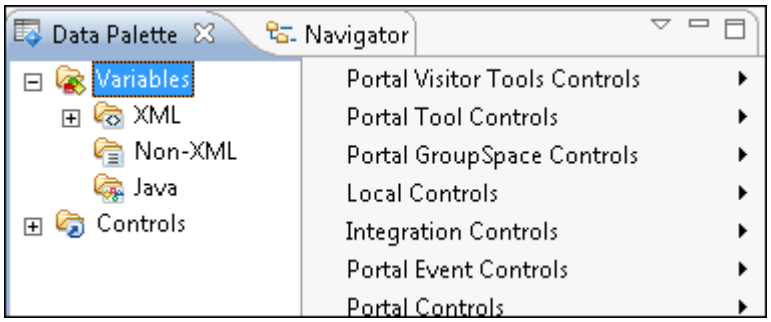
To add a control to the Data Palette:

1. To open the Data Palette view, do the following:
   a. From the menu, select **Window**, then **Show View**, then **Other**.

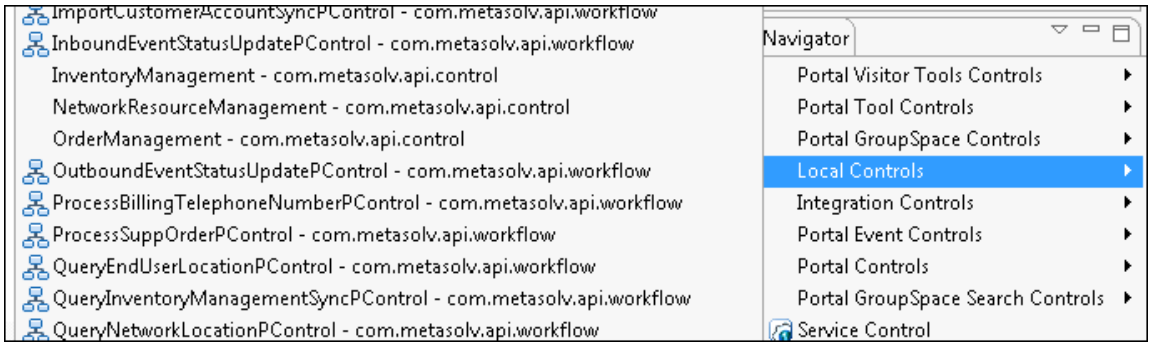
      The Show View window is displayed.

   b. Type **data palette** to filter the views.

   c. Select **Data Palette** and click **OK**.
2. From the menu list, select **Local Controls**.



3. Select the control file (for example, **OrderManagement** or **CustomerManagement**) that you want to add to the Data Palette.



The control and its methods are displayed in the Data Palette view and you can use them within the workflow. You can drag and drop any controls or data transformation controls that you created into the Data Palette.

Adding data transformation controls to a workflow

This section describes how to add data transformation controls to the Workshop Data Palette for use in a workflow.

To add a data transformation control to the Data Palette:

1. Open the Data Palette view:
   a. From the menu, select **Window**, then **Show View**, then **Other**.

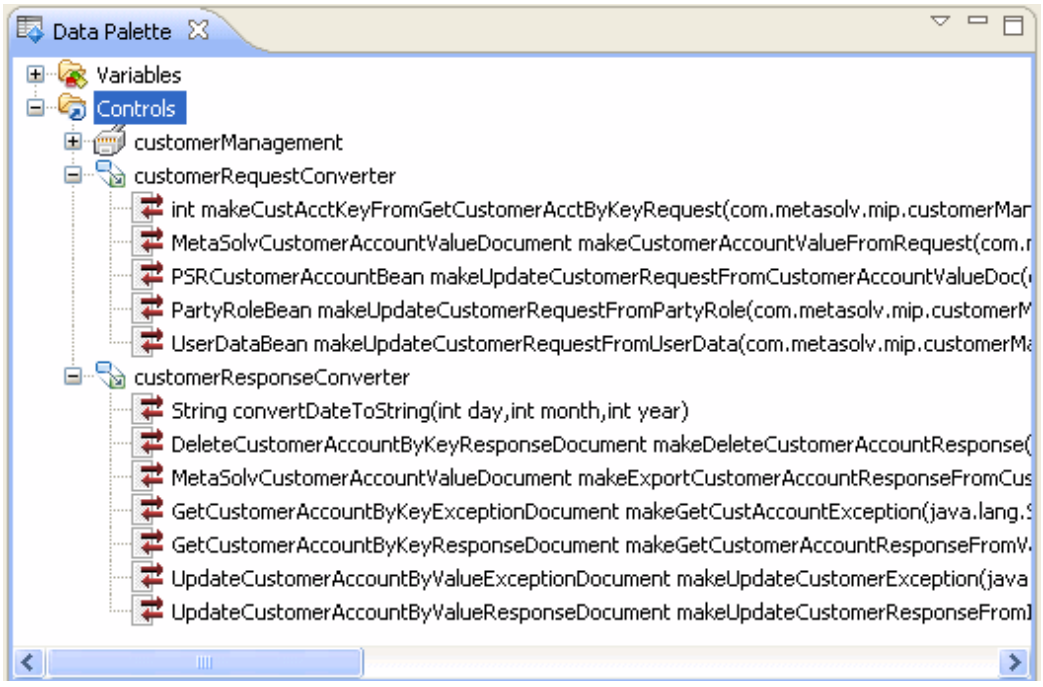
      The Show View window displays.

   b. Enter *data palette* to filter the choices.

   c. Select **Data Palette** and click **OK**.
2. In the Navigator view, navigate to the directory where the data transformation control is located.

   This example shows the addition of the CustomerRequestConverter and the CustomerResponseConverter controls, which are located in the **API/src/com/metasolv/converter/customer** directory.
3. Select the control file, such as **CustomerRequestConverter.java** or **CustomerResponseConverter.java**.
4. Drag and drop the selected file to the **controls** directory in the Data Palette view.

   The control and its methods display in the Data Palette view where they are available for use within the workflow.

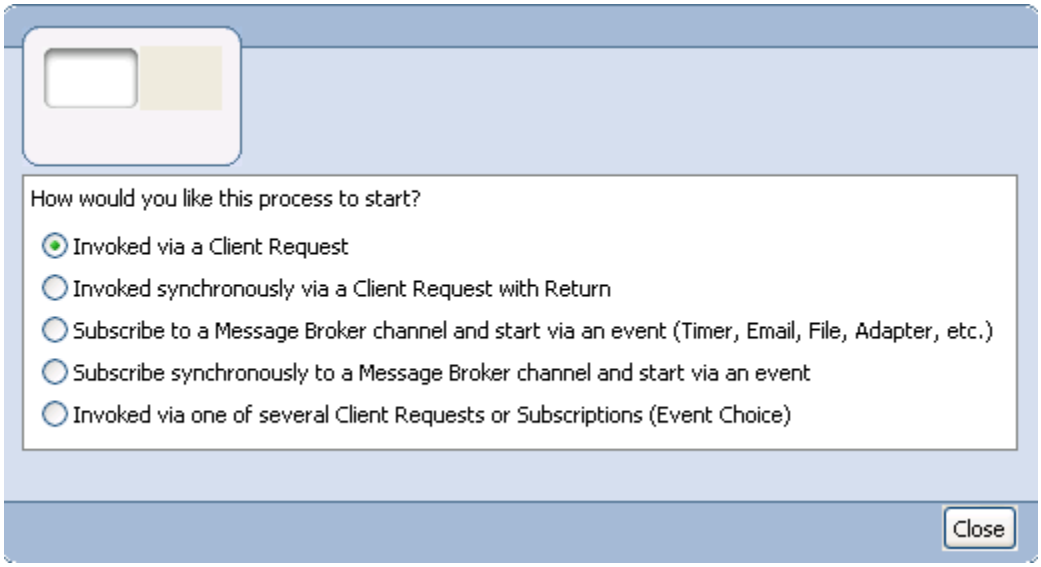


Step 3. Specifying how the request is invoked

To specify how the request is invoked:

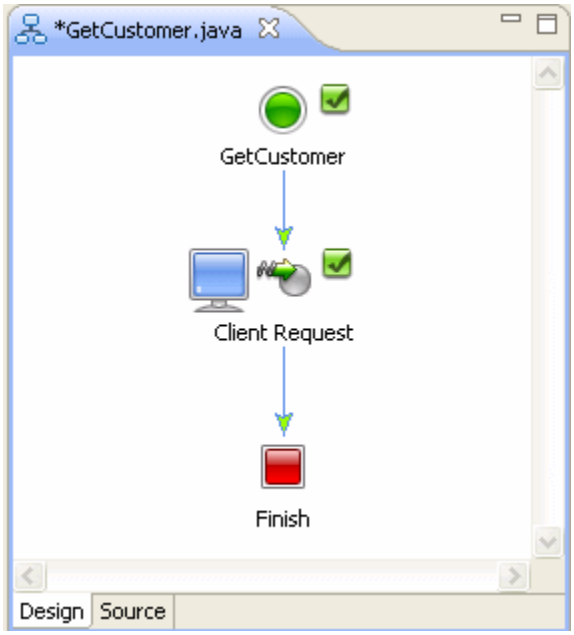
1. Within the process workflow, double-click on the default Start Event node.

   The following dialog box displays:.



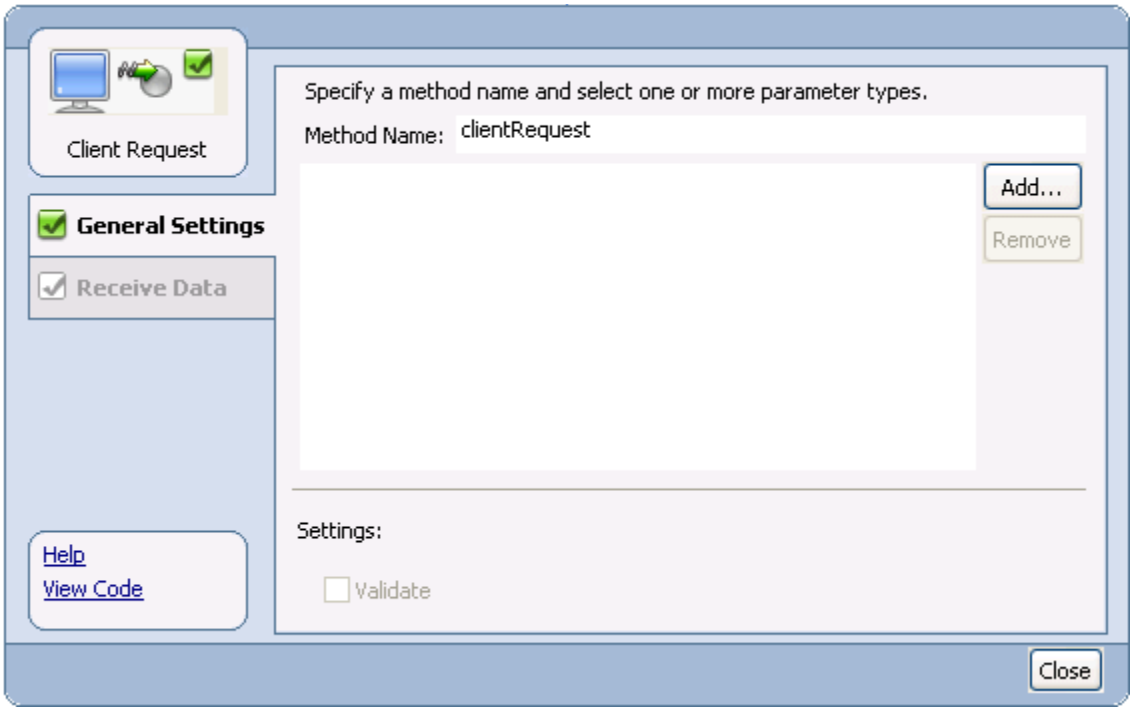
2. Choose the **Invoked via a Client Request** option, and click **Close**.

   The Client Request node now displays in the workflow.



3. Double-click the Client Request node.

   The Client Request dialog box displays.

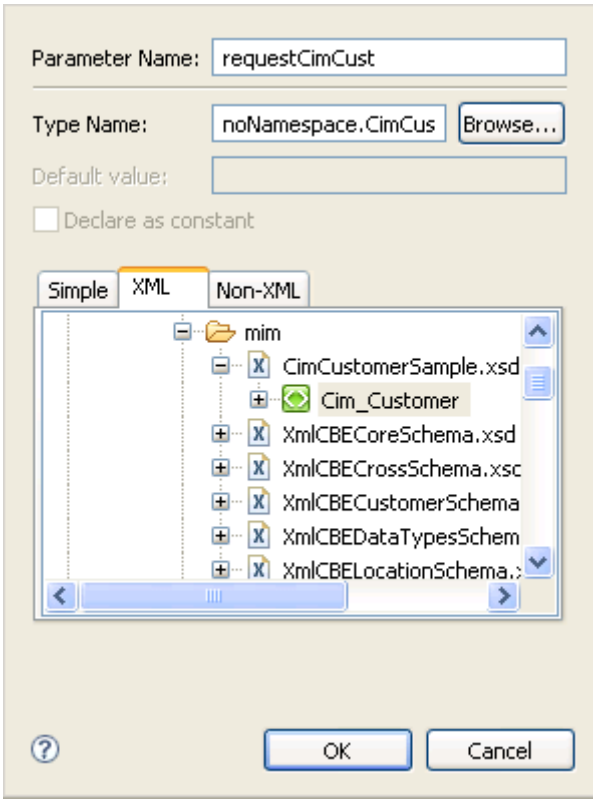


   Double-clicking any node within a workflow results in the node's properties dialog box displaying. The properties dialog box allows you to name the method that the node represents, and to provide the information that is necessary for the method to successfully execute.

4. Complete the following information in the dialog box:

   a. On the General Settings tab, click **Add.**

The following dialog box displays:



b. Click the **XML** tab.

c. Navigate to the location of the schema that defines the incoming XML, and select the appropriate element.

   This example shows the selection of the Cim_Customer element that is defined within the **CimCustomerSample.xsd** file. The **Type Name** field is automatically populated when you select an element.

d. In **Parameter Name**, enter a parameter name for the incoming XML element that was selected.

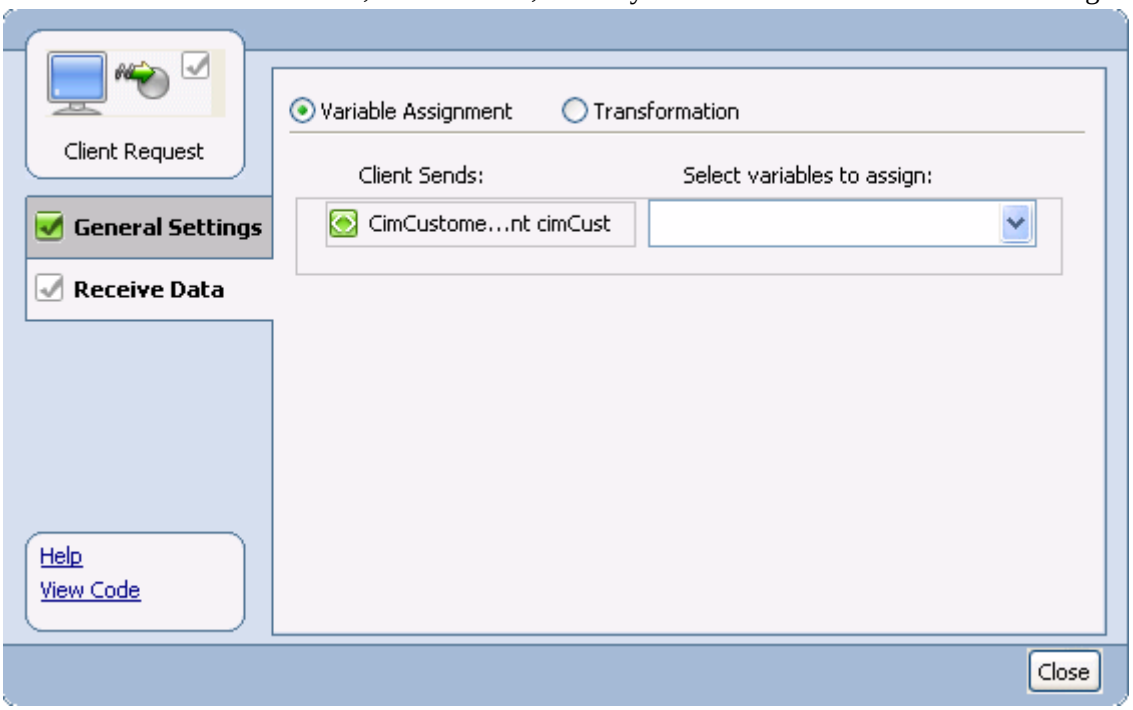
   This example shows a parameter name of *requestCimCust*.

e. Click **OK**.

   This returns you back to node's properties dialog box.

f. Click the **Receive Data** tab.

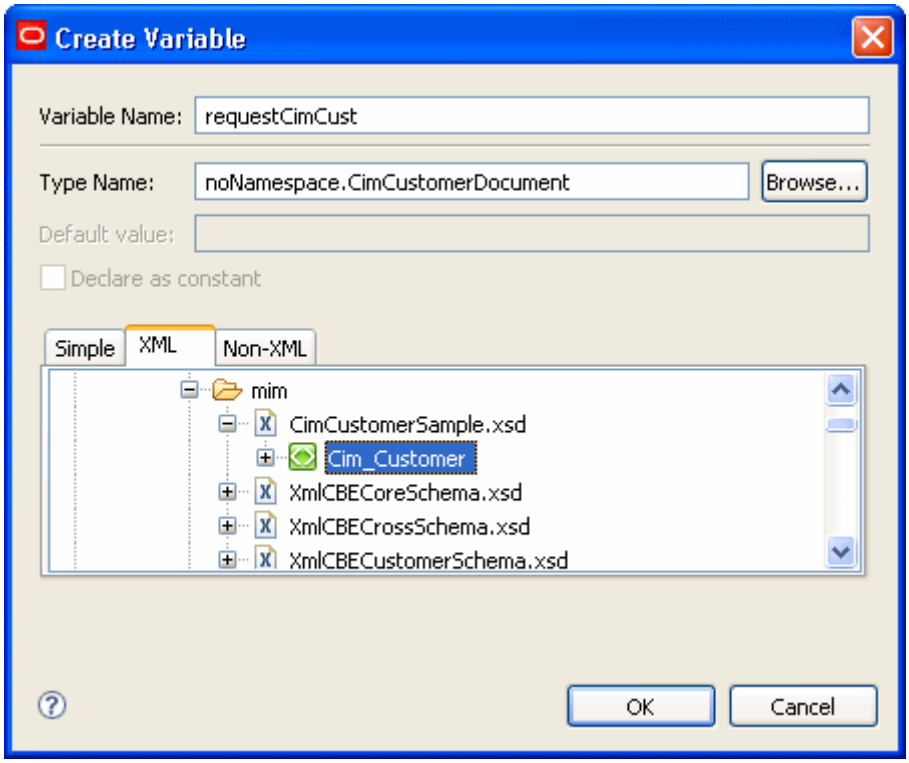
The Receive Data tab, shown below, allows you to define variables for the incoming data.



g. Choose the **Variable Assignment** option.

h. Under **Select variables to assign**, click the list arrow, and then select **Create new variable**.

The Create Variable dialog box displays.



i. Click the **XML** tab.

j. Navigate to the location of the schema that defines the incoming XML, and select the appropriate element.

   This example shows the selection of the Cim_Customer element that is defined within the **CimCustomerSample.xsd** file. The **Type Name** field is automatically populated when you select an element.

k. In **Variable Name**, enter a variable name for the incoming XML element that was selected.

   This example shows a parameter name of *requestCimCust*.

l. Click **OK**.

m. On the node properties dialog box, click **Close**.

Step 4. Adding a group to the workflow

The group box allows you to pull nodes that have a process in common together. In this example, the group box holds nodes that share the same exception processing. In this example, the group contains three methods to accomplish the following tasks:

- Transform incoming data from the requestor's format into the MIM format understood by MetaSolv Solution
- Process the request to export customer information
- Transform data from the MIM format into the requestor's XML format

To add a group to the workflow:

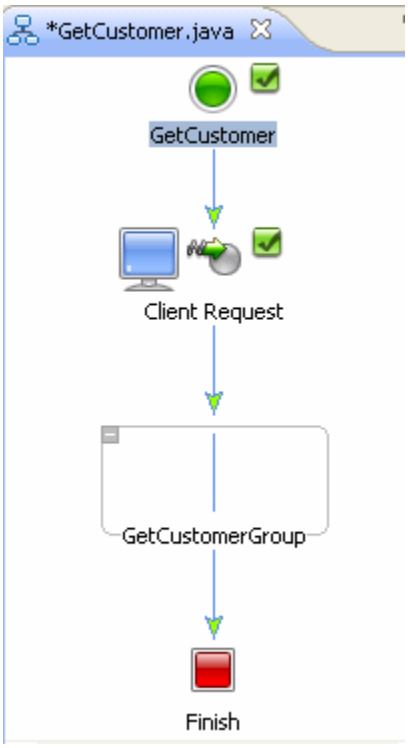
1. Open your workflow.

   In this example, the workflow is the **GetCustomer.java** file.

2. From the menu, select **Insert**, then **Group**.

3. Enter a name for the group, such as *GetCustomerGroup*, and press <*Enter*> on your keyboard.

   The empty group is added to the workflow below the Client Request node.



Step 5. Adding the request transformation method

This step adds the method to transform the data from the requestor's XML format into the MIM format used by MetaSolv Solution.

To add the request transformation method:

1. In the workflow, click on the group.

   In the example, the group name is *GetCustomerGroup*.

2. From the menu, select **Insert**, then **Control Send with Return**.

3. Enter a name for the control, such as *requestTransformation*, and press <*Enter*> on your keyboard.

   The control is added to the group within the workflow.

   For the GetCustomer example, the name is requestTransformation. This is a generic Workshop method. You can use the MetaSolv Solution methods under the MetaSolvCustomerManagement control, but this demonstrates the use of a generic method.

4. Double-click the **requestTransformation** node to open the node properties dialog box.

   Notice that there are now three tabs: General Settings, Send Data, and Receive Data. Previously, the node properties dialog box was shown for the client request node, which only sends data, so there were only two tabs: General Settings and Send Data. The requestTransformation node sends *and* receives data, so there are three tabs: General Settings, Send Data, and Receive Data.

5. On the General Settings tab, click the list arrow for the **Control** field, and select a control.

   For this example, select *MyCustomerConverterRequest*.

Based on the control you select, a list of methods that are defined for the control display.

6. Select a method.

   For this example, select *myMakeGetCustRequest.*

7. Click the **Send Data** tab.
8. Choose the **Transformation** option.
9. Click **Select Variable**.
10. Select the variable that you defined in the Client Request node, *requestCimCust*.
11. Click **Edit Transformation**.
12. Map the appropriate elements for your transformation and close the transformation mapping to return to the requestTransformation node properties dialog box.
13. Click the **Receive Data** tab.
14. Choose the **Variable Assignment** option.
15. Under **Select variables to assign**, click the list arrow, and then select **Create new variable**.

    The Create Variable dialog box displays. The purpose of the Receive Data tab is to create a new variable to receive the transformed data.

16. Click the **XML** tab.
17. Navigate to the location of the schema that defines the XML, and select the appropriate element.

    The **Type Name** field is automatically populated when you select an element.

18. In **Variable Name**, enter a variable name for the outgoing XML element that was selected.
19. This example shows a parameter name of *requestMimCust*.
20. Click **OK**.
21. On the node properties dialog box, click **Close**.

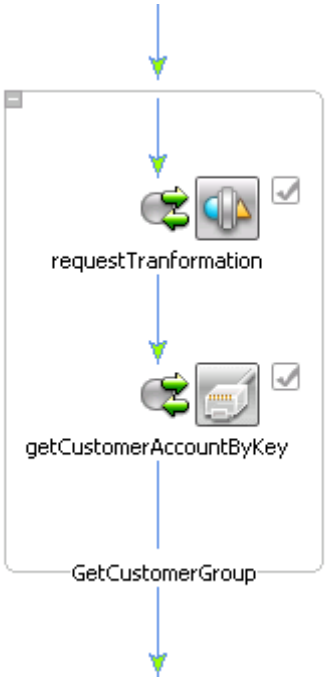
Step 6. Adding the method to process the request

Instead of a generic Workshop method that would have to be modified, this step uses a MetaSolv Customer Management API method created for processing this type of request.

To add the getCustomerAccountByKey method to the workflow:

1. In the Data Palette view, expand the **controls** directory, and expand the MetaSolvCustomerManagement control to display its methods.
2. Locate the getCustomerAccountByKey method.
3. Drag and drop the method from the Data Palette view into the workflow. Drop it into the GetCustomerGroup, below the requestTransformation node.

   The group within the workflow now looks like this:



4. Double-click the **getCustomerAccountByKeyRequest** node.

   The node properties dialog box displays.
5. On the **General Settings** tab, accept the defaults.
6. Click the **Send Data** tab.
7. Choose the **Transformation** option.
8. Click **Select Variable**.
9. Select the variable that you defined in the Client Request node, *requestMimCust*.

In this scenario, which is using a MetaSolv Solution provided converter, you do not need to click **Edit Transformation**, as the transformation is already defined for you.

10. Click the **Receive Data** tab.
11. Choose the **Variable Assignment** option.
12. Under **Select variables to assign**, click the list arrow, and then select **Create new variable**.

    The Create Variable dialog box displays. The purpose of the Receive Data tab is to create a new variable to receive the transformed data.
13. Click the **XML** tab.
14. Navigate to the location of the schema that defines the XML, and select the appropriate element.

    The **Type Name** field is automatically populated when you select an element.
15. In **Variable Name**, enter a variable name for the outgoing XML element that was selected.
16. This example shows a parameter name of *responseMimCust*.
17. Click **OK**.
18. On the node properties dialog box, click **Close**.

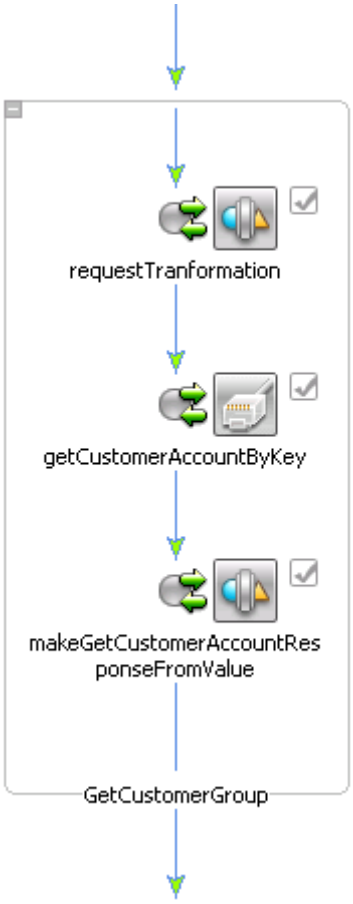
Step 7. Adding the response transformation method

This step uses a MetaSolv Customer Management API method created for processing this type of response.

To add the response transformation method:

1. In the Data Palette view, expand the **controls** directory, and expand the CustomerResponseConverter control to display its methods.
2. Locate the makeGetCustomerAccountResponseFromValue method.
3. Drag and drop the method from the Data Palette view into the workflow. Drop it into the GetCustomerGroup, below the getCustomerAccountByKey node.

   The group within the workflow now looks like this:



4. Double-click the **makeGetCustomerAccountResponseFromValue** node.

   The node properties dialog box displays.
5. On the **General Settings** tab, accept the defaults.

6. Click the **Send Data** tab.
7. Choose the **Transformation** option.
8. Click **Select Variable**.
9. Select the variable that you defined in the Client Request node, *responseMimCust*.

   In this scenario, which is using a MetaSolv Solution provided converter, you do not need to click **Edit Transformation**, as the transformation is already defined for you.
10. Click the **Receive Data** tab.
11. Choose the **Variable Assignment** option.
12. Under **Select variables to assign**, click the list arrow, and then select **Create new variable**.

    The Create Variable dialog box displays. The purpose of the Receive Data tab is to create a new variable to receive the transformed data.
13. Click the **XML** tab.
14. Navigate to the location of the schema that defines the XML, and select the appropriate element.

    The **Type Name** field is automatically populated when you select an element.
15. In **Variable Name**, enter a variable name for the outgoing XML element that was selected.
16. This example shows a parameter name of *responseCimCust*.
17. Click **OK**.
18. On the node properties dialog box, click **Close**.

Step 8. Setting up exception handling

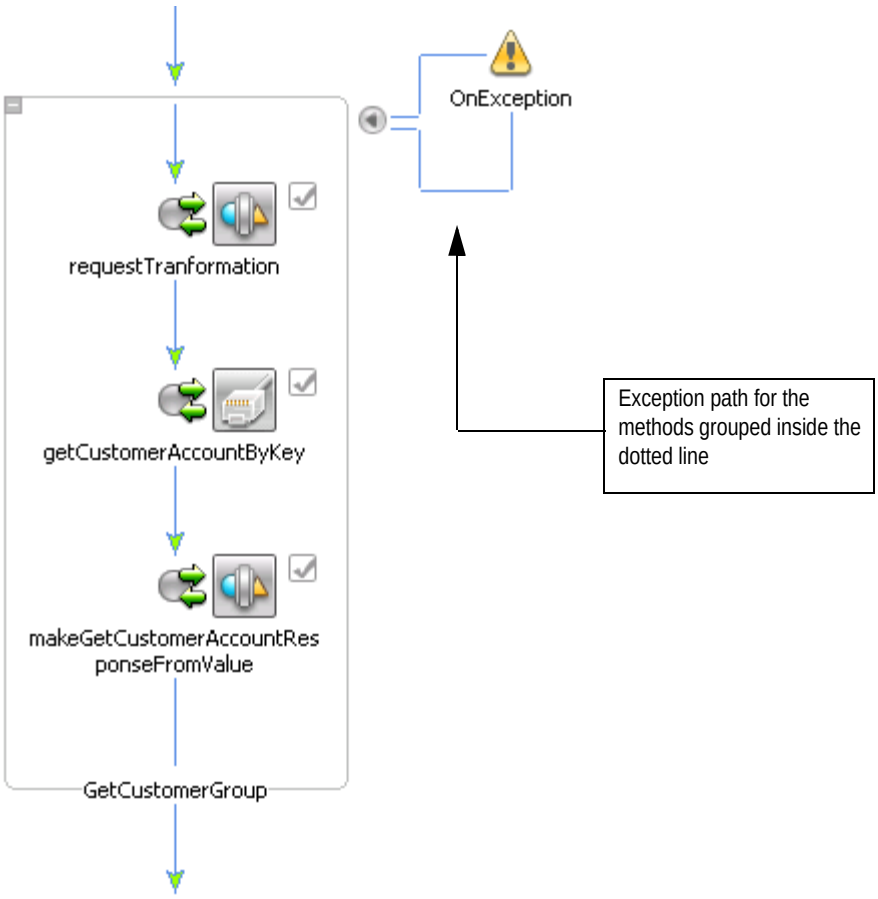
This section explains how to handle exceptions in the workflow. Data is logged in a file named **appserverlog.xml**.

To set up exception handling for the workflow:

1. Within the group, right-click and select **Add Exception Path.**

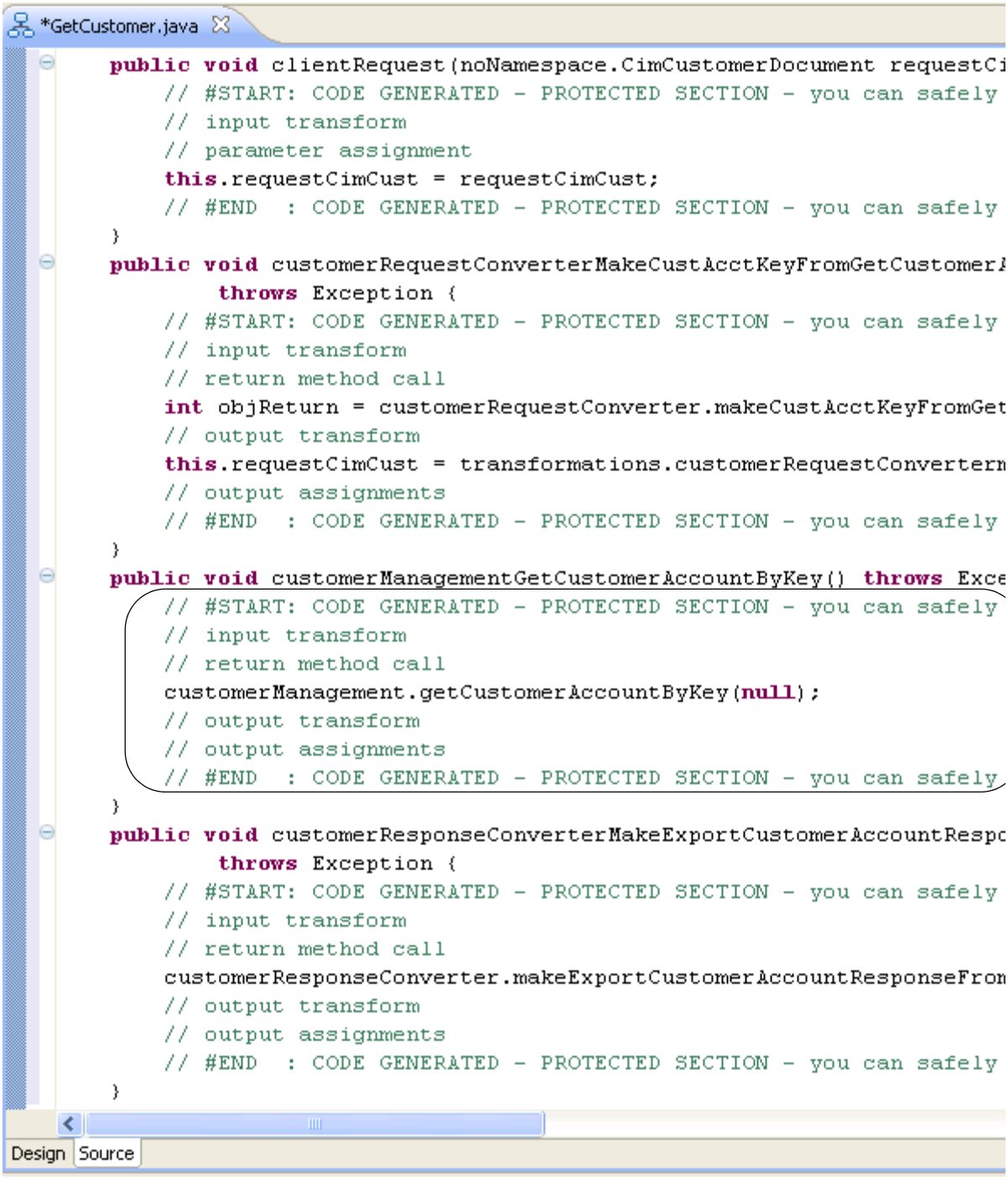
   Note: If you right-click *outside* of the group, you will not see the **Add Exception Path** menu option.

   The exception path is added to the group.



   To catch any exception returned by the XML APIs for one of the methods in the group, you must write a try-catch expression, which is defined in the Text view.

2. At the bottom of the **GetCustomer.java** workflow file that you are working in, click the **Source** tab to view the source code. The code is shown in the following figure.


*GetCustomer.java
public void clientRequest(noNamespace.CimCustomerDocument requestCi
// #START: CODE GENERATED - PROTECTED SECTION - you can safely
// input transform
// parameter assignment
this.requestCimCust = requestCimCust;
// #END : CODE GENERATED - PROTECTED SECTION - you can safely
}
public void customerRequestConverterMakeCustAcctKeyFromGetCustomerA
throws Exception {
// #START: CODE GENERATED - PROTECTED SECTION - you can safely
// input transform
// return method call
int objReturn = customerRequestConverter.makeCustAcctKeyFromGet
// output transform
this.requestCimCust = transformations.customerRequestConvertern
// output assignments
// #END : CODE GENERATED - PROTECTED SECTION - you can safely
}
public void customerManagementGetCustomerAccountByKey() throws Exce
// #START: CODE GENERATED - PROTECTED SECTION - you can safely
// input transform
// return method call
customerManagement.getCustomerAccountByKey(null);
// output transform
// output assignments
// #END : CODE GENERATED - PROTECTED SECTION - you can safely
}
public void customerResponseConverterMakeExportCustomerAccountRespc
throws Exception {
// #START: CODE GENERATED - PROTECTED SECTION - you can safely
// input transform
// return method call
customerResponseConverter.makeExportCustomerAccountResponseFron
// output transform
// output assignments
// #END : CODE GENERATED - PROTECTED SECTION - you can safely
}
Design
Source

Notice that there is one method per node in the workflow: The client request node, plus the three nodes within the group. The code that displays in green text is generated by Workshop and you cannot change this code.

The try-catch expression is needed for external processes that occur in the MetaSolv Solution core. The transformation processes are local and do not require a try-catch expression. In this example, the lines of code to wrap in the try-catch expression are circled in the above figure.

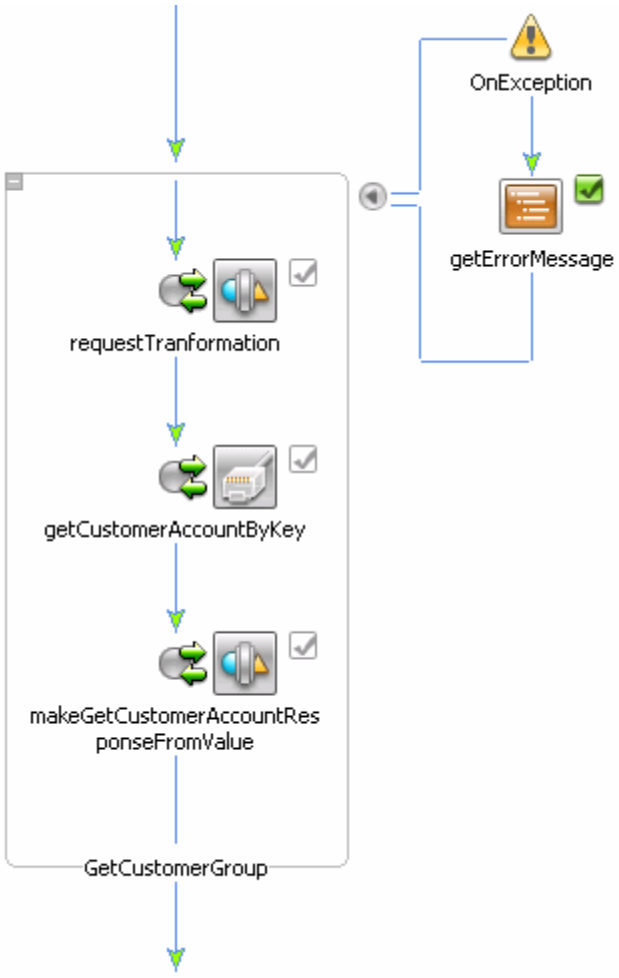
The following is an example of the try-catch expression code. The catch expression references the Logger object, so the import statement is also included in the example:

```
.
.
import com.metasolv.api.common.Logger;
.
.
try
{
   // #START: CODE GENERATED - PROTECTED SECTION - you can safely add code
   above this comment in this method. #//
   // input transform
   // return method call
   customerManagement.getCustomerAccountByKey(null);
   // output transform
   // output assignments
   // #END  : CODE GENERATED - PROTECTED SECTION - you can safely add code
   below this comment in this method. #//
}
catch (Exception e)
{
   String message=com.metasolv.api.common.ProcessUtils.ExtractMessageFrom
   ProcessException(e);
   Logger.logDebugMsg("DEBUG:customerRequestConverterMakeCustAcctKeyFromGet
   CustomerAcctByKeyRequest problem", message);
   throw new Exception(message);
}
```

3. At the bottom of the **GetCustomer.java** source file that you are working in, click the **Design** tab to return to the graphical view of the workflow.

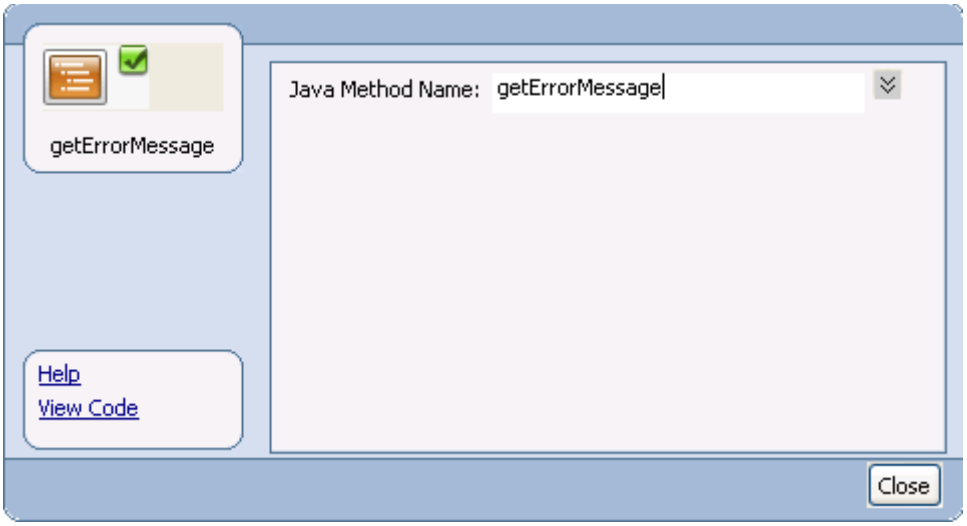
4. Click the **OnException** node in the workflow.
5. From the menu, select **Insert**, then **Perform.**
6. Enter a name for the control, such as *getErrorMessage*, and press <*Enter*> on your keyboard.

   The control is added below the **OnException** node.



7. Double-click the **getErrorMessage** node.

   The node properties dialog box displays.



8. In **Java Method Name**, enter getErrorMessage.
9. Click the **View Code** link.

   Workshop generates the method based on the method name you just entered.

```
public void getErrorMessage() throws Exception
{
}
```

10. Manually define the following variables in the code in two places: As class variables, and as method variables.
    - errorMessage
    - exceptionToString

```
public class GetCustomer implements ProcessDefinition
{
   String errorMessage;
   String exceptionToString;
   .
   .
   public void getErrorMessage() throws Exception
   {
```

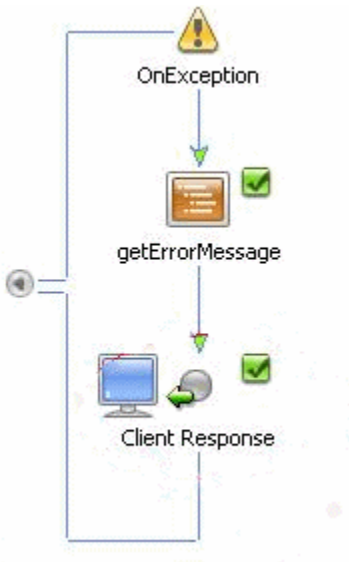
```
        this.errorMessage = this.errorMessage + "####" +
        context.getExceptionInfo.getException.getLocalizedMessage();

        this.exceptionToString = this.exceptionToString + "####" +
        context.getExceptionInfo.getException.toString();

        context.getExceptionInfo.getException.printStackTrace();
    } //end method
.
.
} // end class
```

11. At the bottom of the **GetCustomer.java** source file that you are working in, click the **Design** tab to return to the graphical view of the workflow.
12. Click **Close** to close the getErrorMessage node properites dialog box.
13. Click the **GetErrorMessage** node in the workflow.
14. From the menu, select **Insert**, then **Client Response**.
15. Leave the name as **Client Repsonse**, and press <*Enter*> on your keyboard.

    The Exception path within the workflow now looks like this:



16. Double-click the Client Response node.

17. On the **General Settings** tab, in **Method Name**, enter a method name such as *errorClientResponse*.
18. Click **Add**.
19. In the dialog box that displays, choose the **Java** option.
20. Expand the **Java Types** directory, and select **String**.

    This selection automatically populates the **Type** field.
21. In **Name**, enter *errorMessage*.

    errorMessage is the name of one of the exception variables that you manually defined in the code.
22. Click **OK**.
23. Repeat steps 18-22 to add *exceptionToString*, the other exception variable that you manually defined in the code.

    The following figure shows both variables defined for the *errorClientResponse* method on the **General Settings** tab.

24. Click the **Send Data** tab.
25. For each variable, select the appropriate variable in the **Select variables to assign** list.

26. Click **Close** to close the Client Request node properties dialog box.
27. On the menu, click **Save** to save the workflow.

Testing the application in Workshop

When the workflow is complete, you can test it in Workshop.

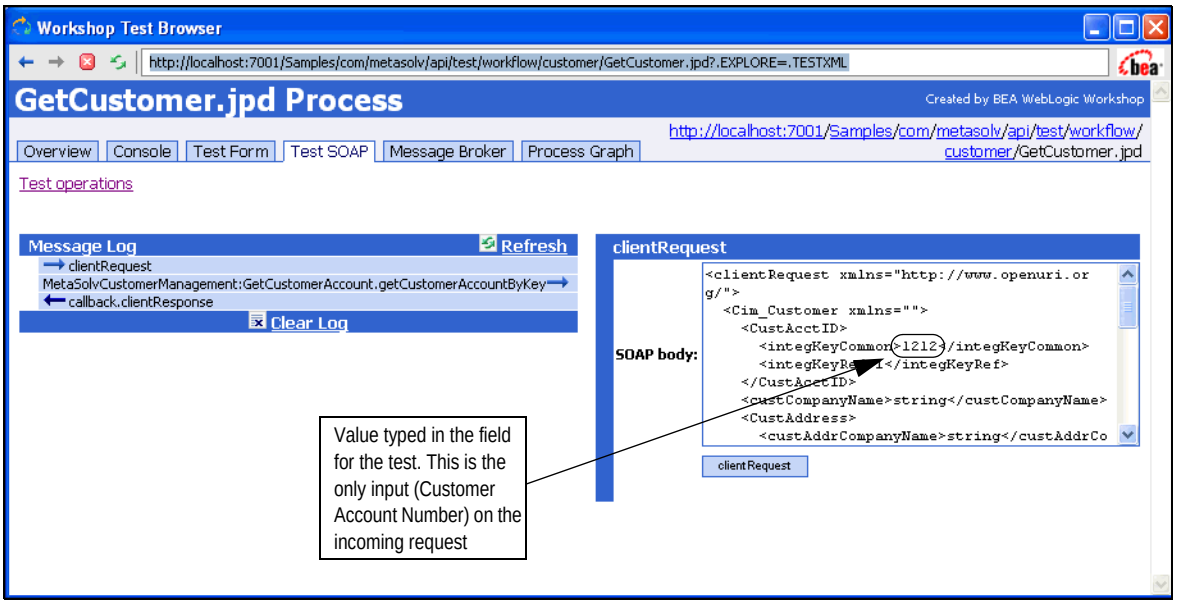
To test a workflow:

1. On the **Server** tab, right-click the server and select **Add and Remove Projects**.
2. Click **Add (>)** to move the created project from **Available Projects** section to **Configured Projects** section.
3. If the server status shows **Republish**, right-click the server, and then click **Publish**.

   This deploys the application on the server.
4. Right-click the **GetCustomer.java** file (workflow file), select **Run As**, then **Run on Server**.

   The workshop test browser opens.
5. Select the **Test SOAP** tab.

   Workshop creates an XML file for the Client Request method based on the requestor's XSD. In the case of the example used in this chapter, the test XML file was created from the Cim_Customer XSD.



Creating a build

To create a build:

1. From the File menu, select **Project**, then **Build All**.

To create a build and an ear file:

1. From the **File** menu, select **Export**.

   The Export window is displayed.

2. Under **Other**, select **EAR file with JSP pre-compilation**.
3. Click **Next**.
4. Specify the destination path where you want to extract the EAR file.
5. Click **Finish**.

   The EAR file is extracted.

Note: When you create an ear file in Workshop, it is automatically deployed to the WebLogic Server.

Migrating To MetaSolv Solution 6.2.1

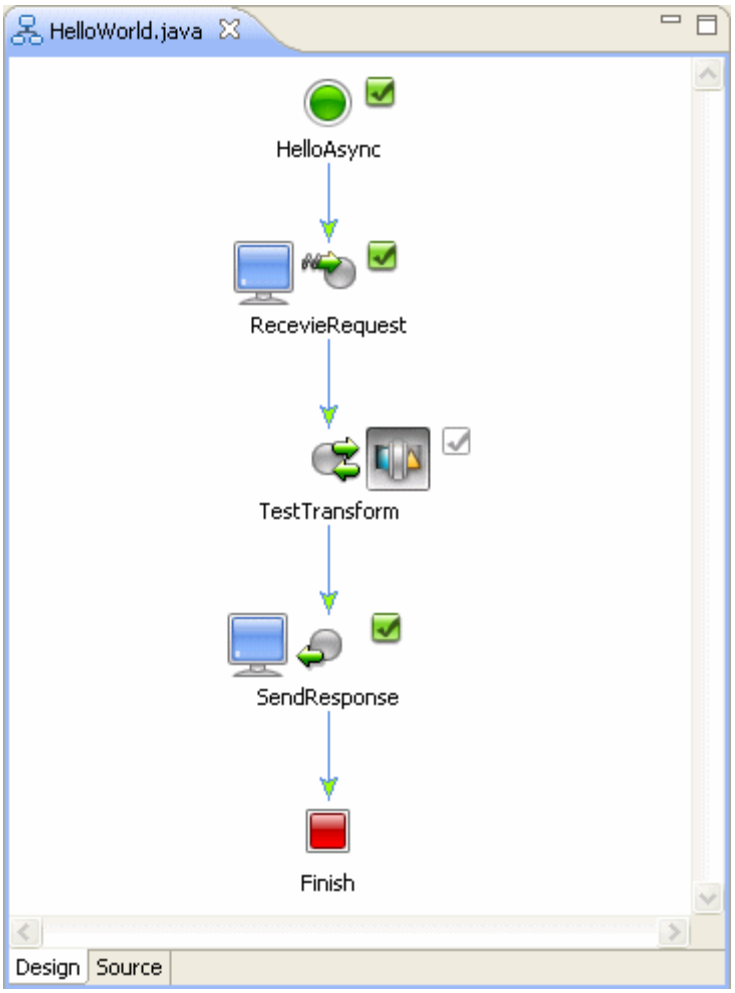
The previous release of the MetaSolv Solution XML APIs used WebLogic Workshop 8.1. The MetaSolv Solution 6.2.1 XML APIs use WebLogic Workshop 10.3.1. So, if you are migrating from a previous release of MetaSolv Solution, you must migrate the XML API-related files to the newer version of WebLogic Workshop.

The following procedure assumes that you have already installed MetaSolv Solution 6.2.1, which includes the WebLogic Workshop 10.3.1 installation. For detailed information on performing the actual installation, see *MetaSolv Solution Installation Guide*.

To migrate the XML API-related files:

1. Install WebLogic patches.
   For detailed information on installing WebLogic patches, see the *Installing WebLogic Server Patches* section, located in Chatper 3 of *MetaSolv Solution Installation Guide.*
2. Open WebLogic Workshop 8.1.

3. Create a test project that includes a basic workflow such as the one shown in the following JPD file.



4. Successfully build the project in WebLogic Workshop 8.1.
5. Open WebLogic Workshop 10.3.1.
6. On the Workspace Laucher window, select an empty workspace and go to the Workbench. In this example, the empty workspace is named *migration_project*.
7. From the menu, select **Project** and disable the **Build Automatically** option.
8. From the menu, select **Windows**, then **Preferences**.
9. In the navigation tree, select **Validation**.
10. Deselect the **XML Schema Validator** check boxes (Manual and Build).
11. Deselect the **XML Validator** check boxes (Manual and Build).

12. Click **OK**.
13. From the menu, select **File**, then **Import**.
    The Import window opens.
14. In the **Select an import source** text box, enter *workshop* to filter the selections.
15. Select **Workshop 8.1 Application** and click **Next**.
    The Workshop 8.1 Application Upgrade window opens.
16. Click **Browse** and navigate to the test project you created in step 3.
    This populates the **Workshop 8.1 Application** field.
17. Click **Next**.
18. Select the three check boxes for:
    - Properties File Upgrader options
    - JPD Document Upgrader options
    - JSP File Migrator options
19. Click **Finish**.
    This migrates the 8.1 application to a 10.3.1 application, with the resultant 10.3.1 application being placed in the empty workspace you created in step 6. In this example, the empty workspace was named *migration_project*.
20. Before you build the project, you must update the JAR files in the *migration_project/<APPLICATION_DIR>*/**EAR_content/APP-INF/lib** directory. To do this:

    Navigate to the directory and remove the following JAR files:
    - MetaSolvSolutionSchemas.jar
    - MetaSolvSolutionUtil.jar

    In the same directory, verify that the following JAR file is present. If the file is not present, copy it from the *mss_integration.ear* file to the directory.
    - MetaSolvSolutionUtility.jar
21. Successfully build the project in WebLogic Workshop 10.3.1.
22. Run the JPD to test the migrated application.
23. After successfully migrating and testing the test project, perform the migration on on all existing XML API projects. To do this, repeat steps 13 through 20 for each existing XML API project. (For step 16, select an existing XML API project instead of selecting the test project.)
24. After successfully migrating and testing all existing XML API proejcts, reselect the **XML Schema Validator** and **XML Validator** check boxes that you deselected in steps 10 and 11.

4

Post Development Tasks

Updating the production database

During development, the MetaSolv Solution XML API application requires the creation of a number of tables within the database used for WebLogic Integration conversation state tracking. The related database is defined within the WebLogic data-source named bpmArchDataSource. This data-source is configured during creation of the WebLogic domain using the Configuration wizard. To move the application into a production mode, you must recreate the tables in the production database.

To create the production database tables, you must complete the following tasks:

1. Create an SQL script to create tables in the production Oracle database.
2. Run the SQL script against the production database to add the tables.

Creating the SQL script

The following shell scripts *tableTool.sh* can be used to generate the Integration state tables:

Windows: tableTool.bat

UNIX: tableTool.sh

Prerequisites

You must have Oracle WebLogic.

To create the SQL script:

1. Copy the following files to a directory on the server:
   - ManifestTableGenerationTemplate.xsl
   - ManifestTableRemovalTemplate.xsl
   - tableTool.class
   - tableTool.sh
2. To generate the Integration State Tables SQL scripts, execute the tableTool.sh script with the following arguments:

   ./tableTool.sh [*JAVA_HOME*] [*mss_integration.ear*] [*createTable* | *dropTable*]

Arguments:

[*JAVA_HOME*] is the location of the JDK.
For example: /opt/bea/jdk160_05

[*mss_integration.ear*] is the location of the mss_integration.ear.
For example: /opt/bea/user_projects/domains/paetec/lib/mss_integration.ear

[*createTable* | *dropTable*] If **createTable** is specified, the SQL script tableGenerationTemplate.sql is generated.The SQL script can be used to create the Integration state tables.

If **dropTable** is specified, the SQL script tableRemovalTemplate.sql is generated. This SQL script can be used to drop the Integration state tables.

Example: The following example shows the command to generate a create table SQL script for the Integration state tables:

./tableTool.sh /opt/bea/jdk160_05 /opt/bea/user_projects/domains/newtel/lib/mss_integration.ear createTable

Example: The following example shows the command to generate the drop table SQL script for the Integration state tables:

./tableTool.sh /opt/bea/jdk160_05 /opt/bea/user_projects/domains/newtel/lib/mss_integration.ear dropTable

Running the SQL script

The SQL file is designed for configuring an Oracle database on a UNIX or Windows platform. The SQL syntax may vary slightly by database vendor. Modifications to the syntax of the commands may be required for successful creation of the tables. Some tables may already be present.

If some commands do not execute due to pre-existence of the table, you may ignore the error. A number of the tables within the script are common tables required by integration applications and may already be present on your system.

Note that the SQL file to drop tables is provided for your convenience.

The Java Naming and Directory Interface (JNDI) name must be exactly as shown. The names are case sensitive. Names must include the periods (.) and underscores (_) as shown.

To run the SQL script:

1. Connect to the WebLogic Integration database as a user having create table privileges.
2. Run the SQL file.

Setting up gateway events

In addition to the configuration required to make the MetaSolv Solution adapter functional at installation, MetaSolv Solution must be configured to receive data and return data.

The Integration server continuously checks the MetaSolv Solution database for events that are ready to be sent to an external application. The Integration server also monitors the external application for updates to the status of a gateway event. When an external application sends a status update, the Integration server records the new status in the MetaSolv Solution database.

The following sections describe how to set up gateway events for communication between MetaSolv Solution and the adapter.

Creating a gateway event

Gateway events are set up in the MetaSolv Solution user interface. Complete information on how to create a gateway event is located in MetaSolv Solution online Help.

To access the MetaSolv Solution online Help for gateway events:

1. On the main toolbar, click **Work Management**.
2. On the Work Management toolbar, click **Gateway**.
3. Press **F1** for Help.

The Help window that appears is specifically for the Gateway window. You will find a number of links on this window that explain gateway events and how to create them. The basic steps for creating a gateway event are outlined below. Each step is described in detail in the online Help.

Basic steps for creating a gateway event:

1. Create a new gateway event.

   For example *xxx_integration_order_event.* This is done on the Gateway window in MetaSolv Solution. MetaSolv Solution creates a gatewayEvent and assigns an eventID.

2. Add a binding to the gateway.

   Set the Binding Type to **IOR** and provide the location path to the NameService.ior file. The path should be similar to the following examples:

   Windows

   c:\Metasolv\Appserver\IOR\NameService.ior

   UNIX

   Metasolv/Appserver/IOR/NameService.ior

   The **Service Name** should be the same as the name of the WebLogic server that will receive events tied to the gateway.

3. Associate the gateway event with a task on the desired request.

Configuring the gateway.ini file

You must make sure the Integration server is configured in the gateway.ini file. This is a configuration file for MetaSolv Solution and it is located on the machine running the MetaSolv Solution application server. The file can be found in the \appserver\gateway directory.

Use an ASCII editor to open the gateway.ini file. Make sure the INTEGRATIONSERVER line located in the ThreadProcs section is uncommented. If INTEGRATIONSERVER is commented, uncomment it and save the changes.

The following sample shows an uncommented INTEGRATIONSERVER line (in bold typeface).

```
[ThreadProcs]
INTEGRATIONSERVER=com.mslv.integration.integrationServer.S3Startup
EVENTPROC=MetaSolv.eventServer.S3Startup
EVENT2PROC=MetaSolv.event2Server.Event2ServerStartup
SYSTEMTASKSERVERPROC=com.mslv.core.api.internal.WM.systemTaskServer.
       SystemTaskServer
SIGNALSERVERPROC=com.metasolv.system.StartServer INTERNET_SIGNAL_SER
       VER=MetaSolv.CORBA.WDIINTERNETSERVICES.WDIRoot,MetaSolv.Sig
       nalServer.WDIInternetSignalServerRootImpl
```

5

Troubleshooting

This chapter provides the information on troubleshooting servers, JDBC connections and error messages.

JRE update failure

Problem: In a Windows environment, a JRE update failure occurs during installation when multiple servers share the same *WebLogic_Home* directory, where *WebLogic_Home* is the directory in which the WebLogic server is installed.

Cause: Other servers are running and using the files needed for installation.

Solution: Shut down the servers and jorbd processes (java) and rerun the installation program on the server where the failure occurred, then restart all servers.

DEBUG_PORT

Problem: The default for this port is the value 8453 and currently it cannot be changed. This occurs only in development mode.

Solution: Currently none.

Testing JDBC connections

Generally, you should leave the JDBC connections settings on their default values. However, if you experience connection issues, there are some advanced options you can set to allow you to test your JDBC connections. To access these advanced options, do the following:

1. Open the WebLogic Server Administration Console by entering the following in your internet browser:

   http://*host_admin*:*port*/console

   where:

   host_admin is the name of the administration server

   port is the administration server port number

2. When the Login window appears for either console, enter your user name and password and click **Login**.

3. On the home page, expand **Domain Configurations**, then **Services**, then **JDBC**, and then click the **Data Sources** link.

   The Summary of JDBC Data Sources page displays.

4. In the Data Sources table, under the **Name** column, click the link that represents the applicable JDBC data source.

   The Settings page for the specific JDBC data source you selected displays.

5. Click the **Configuration** tab.

6. Click the **Connection Pool** tab.

7. Scroll down, and click **Advanced** to display the advanced options.

8. Use these options to test connections, change timeouts, and so on.

   These options can have an impact on performance. After you have finished testing your connections, set these options back to their default values to maintain a higher performance level.

   For more information about these advanced options, refer to the Oracle WebLogic documentation at:

   http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html

Firewall closes idle connections

If you have configured a firewall between the database and WebLogic Server, and this firewall closes idle connections after a certain amount of time, the JDBC pool refresh functionality can be used to ensure that connections from the pool are not closed by the firewall. A common error message thrown after such a closed connection is used follows:

```
java.sql.SQLException: ORA-03113: end-of-file on communication channel
at weblogic.db.oci.OciCursor.getCDAException(OciCursor.java:240)
at weblogic.jdbc.oci.Statement.executeQuery(Statement.java:916)
at ...
```

This error occurs because the socket connection is considered okay from both the WebLogic Server and the database side. So both may try to write into this socket connection and fail, because it has been closed by the firewall without notification or error message to the participating parties. Please use the refresh functionality to ensure that the connections are not idle long enough for the firewall to close them.

Configuring refresh functionality can be done by setting the `RefreshMinutes` property so that connections are tested at least one time during the idle period. To enable the refresh functionality, `TestTableName` property also has to be set. For more information, see:

http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html

However, every JMS server takes one connection from the JDBC pool if a JDBC store is defined. This connection is considered as reserved by the pool, so that the refresh functionality will not test and refresh those connections. A typical error message would be:

JMSServer "myJMSServer", store failure while writing message for queue myQueue, java.io.IOException

This kind of situation can be solved by either one of the following options:

- Send at least one dummy JMS message during the idle period, so that the firewall will not close the connection
- Disable the connection closure by the firewall
- Define a separate JDBC pool that will be used as JDBC store for JMS servers and use weblogic.Admin RESET_POOL to reopen the connections at least one time during the idle period

This problem has been addressed in later WebLogic Server versions (WLS 6.1 SP7, WLS 7.0 SP3, and WLS 8.1 SP1), so that if JMS is idle, the database is pinged every 5 minutes to keep the connection fresh and prevent the firewall from closing these connections

Table errors

If you are getting table or view errors after deploying to production, read the following sections to make sure you have properly deployed your application.

- **Development mode and production mode**

  When you are developing, deploying and testing an application with WebLogic Workshop, the instance of WebLogic Server you are deploying to runs by default in *development mode.* In development mode, WebLogic Server behaves in ways that make it easier to iteratively develop and test an application: it automatically deploys the current application in an exploded format, server resources such as database tables and JMS queues necessary for the application to run are automatically created, and so on.

  When the development cycle is complete, and the application is ready for use, you deploy it to an instance (or instances) of WebLogic Server running in *production* mode. In production mode applications are not automatically deployed and the server resources necessary for running an application are not automatically generated.

- **Manual Creation of Server Resources**

  When deploying EAR files to a production server, a certain amount of manual resource creation is necessary. When an application is built in an EAR file, a wlw-manifest.xml file is produced and placed in the application's META-INF directory. This file lists the JMS

queues and database tables that need to be manually created on the target WebLogic Server for the application to run properly.

When you are developing and testing an application with WebLogic Workshop, the creation of the necessary JMS queues and datatables on WebLogic Server takes place automatically on demand.

Required database tables are indicated by a **<con:conversation-state-table />** tag. These tables are used by web services to store conversational state. For each occurrence of the <con:conversation-state-table /> tag in the wlw-manifest.xml file, you must create a corresponding data table on WebLogic Server.

Required JMS queues are indicated by pairs of **<con:async-request-queue>** and **<con:async-request-error-queue>** tags. For each occurrence of these tags in the wlw-manifest.xml file, you must create a corresponding JMS queue on WebLogic Server *and* you must associate the members of the pair by referencing the <con:async-request-error-queue> in the ErrorDestination attribute of the <con:async-request-queue>.

Optionally, you may want to enforce role restrictions on any controls that receive external callbacks. Controls that can receive external callbacks are indicated within a **<con:external-callbacks/>** tag in the wlw-manifest.xml file. Since the compilation process turns control files into individual methods on an EJB, you enforce the role restrictions on these post-compilation EJB methods.

MetaSolv Solution adapter error message

Problem: The MetaSolv Solution adapter returns the following message to the client during a XML API invocation when the MetaSolv Solution application server is down:

java.lang.Exception: Error connecting to Metasolv Solution Server

Below is a sample response.

```
<ns:clientResponse xmlns:ns="http://www.openuri.org/">
<inv:createEntityByValueException xmlns:inv="http://www.metasolv.com/MIP
   InventoryManagementAPI">
<inv:createException>
<com:message xmlns:com="http://java.sun.com/products/oss/xml/Common">
   java.lang.Exception: Error connecting to Metasolv Solution Server
      </com:message>
</inv:createException>
</inv:createEntityByValueException>
</ns:clientResponse>
```

Solution: If you receive this error, restart the MetaSolv Solution application server, then the Integration server.

XQuery transformation errors

Problem: XQuery transformation error:

FORG0005: expected exactly one item, got 0 items, passes null as the input to the xquery.

Solution: This scenario is internally handled in WebLogic Server 8.1. To resolve this issue in WebLogic Server 10.3.1, specify the input value as optional.

Problem: XQuery transformation error:

Error converting Xquery parameter to Java parameter "convertStringToInt" in external function “String."

Solution: This scenario is internally handled in WebLogic Server 8.1. To resolve this issue in WebLogic Server 10.3.1, you must do typecasting.

Java Process Definition (JPD) issues

Problem: While executing the customized JPDs, the output contains different namespaces in the result data.

Solution: Use the following factory.parse method to the output data:

this.orderResponse = com.metasolv.mip.orderManagementAPI.CreateOrderByValueResponseDocument.Factory.parse(this.orderResponse.toString());

Problem: After you build a JPD and deploy it to the server on WebLogic Workshop on a Linux machine, the server crashes displaying any of the following SIGSEGV errors:

An unexpected error has been detected by Java Runtime Environment:

SIGSEGV (0xb) at pc=0x062565bc, pid=5420, tid=2243156880

Java VM: Java HotSpot(TM) Server VM (10.0-b19 mixed mode linux-x86)

Problematic frame:

V [libjvm.so+0x2565bc]

If you would like to submit a bug report, please visit:

http://java.sun.com/webapps/bugreport/crash.jsp

The crash happened outside the Java Virtual Machine in native code.

See problematic frame for where to report the bug.

Solution: Add the following to the **workshop.ini** file.

-XX:CompileCommand=exclude,org/eclipse/core/internal/dtree/DataTreeNode,forwardDeltaWith

WebLogic Workshop issues on Linux

Problem: By default the WLI installer installs both the Sun and JRockit JDKs. When configuring workshop, JRockit is selected automatically.

Solution: Update the **workshop.ini** file to point to a Sun JRE.

Using XML APIs on Cluster Environment

Problem: You cannot run the XML APIs through Test SOAP (HTTP) in a cluster environment. The XML APIs work fine only in Production mode in a Cluster environment.

Solution: See the following Web page for more information:

https://support.us.oracle.com/oip/faces/secure/km/DocumentDisplay.jspx?id=1150266.1&h=Y

Sample API errors

Problem: When running the Sample APIs, you receive the following error:

“com.bea.wli.knex.runtime.jcx.dispatcher.JcxDispClass cannot be cast to

com.bea.wli.knex.runtime.jws.dispatcher.JwsDispClass”

Solution: Ensure that you have applied the patches mentioned in the 6.2.1 Installation guide.

JMS issue

Problem: After you migrate from WebLogic Server 8.1 to 10.3.1, you cannot use the JMS defined in WebLogic Server 8.1 to pass messages from one queue to another.

Solution: Define WLI JMS in WebLogic Server 10.3.1.

A

Appendix A: XML API Sample Code

The MetaSolv Solution installation provides a sample application, *mss_samples*, as a reference and guideline for developing Workshop applications to interface with MetaSolv Solution XML APIs. If you are a developer setting up a workstation to do integration development, this application is designed to help you understand and work with the XML APIs.

Each sample is a test of a method included in the XML APIs. The sample code is placed in the following directory names: *customer, events, order,* and *inventory* so that you can quickly find the method you want.

The samples demonstrate the following information:

- Initialization
- Error handling
- Asynchronous interaction
- Http transmission
- JMS transmission

Where to find the sample files

The sample application is provided in the jar file *mss_xml_apiR#_b#.jar*

where:

R# is the release version

b# is the build version

For example: mss_xml_api_R603_b185.jar

The file contains the following entries:

- **mss_samples.jar**—This file contains the code, libraries and other files required for WebLogic Workshop-based development for the mss_samples application.
- **mss_samples.ear**—This is the representation of an ear file that results from a successful build of the application.
- **ReleaseNotes.doc**—This contains any release specific notes, including enhancements and fixes.

During installation, the jar file containing the sample code is stored on the application server in the following location:

MSLV_Home/server/**appserver/samples**

where:

MSLV_Home is the directory in which the MetaSolv Solution software is installed

server is the name of the WebLogic server

You can extract the contents of the jar file and place it in any location. The following procedure shows how to set up a sample application from the *samples* directory in a Windows environment.

Setting up the sample code

To set up the XML API sample code on a Windows workstation:

1. Extract the contents of mss_samples.jar file into your Eclipse workspace directory.

   The directory can be an existing one or you can create a new directory. For example: *WebLogic_Home***user_projects\workspaces\mss_samples**.

2. Open the Workshop IDE and select the workspace that contains the extracted mss_samples.jar file.

3. Add a server to the provided project.To do this, follow the detailed instructions in "Creating a new server in Workshop", which are breifly outlined here:

   a. Select **File**, then **New**, then **Other**.

   b. In the **Wizards** field, enter *server.*

   c. Select **Server** and click **Next**.

   d. By default, Oracle WebLogic Server v10.3 is selected. Click **Next**.

   e. Oracle recommends keeping the server and the domain on the same machine in a development environment, so for **Server Type** choose **Local**.

   f. In **Domain Directory**, click **Browse** to navigate to the following directory:

      C:/bea103mp1/user_projects/domains/test/base_domain

   g. Click **Next**.

   h. Copy the respective projects from **Available projects** to **Configured Projects**.

   i. Open the Servers view to see your cofigured server.

4. In the Package Explorer view, select the highest level available, right-click, and select **Build Application**.

This process can take from 15 to 45 minutes, depending on the size of the machine you are building the application on. When the build process is finished, a message appears in the Build tab indicating the build was successful.

5. When the build is complete, restart the application server.

Execute the workflows or browse through the workflows to see how they are constructed.

Upgrading sample files

When you move from one release to another, you must also upgrade the sample file.

To upgrade a sample file:

1. Upgrade the MetaSolv Solution core application.

   See *MetaSolv Solution Installation Guide XML API Option* for instructions.

2. Log on to the WebLogic Server Administration Console.
3. On the Home page, under Domain Configurations, under Your Deployed Resources, click **Deployments**.

   The Summary of Deployments page displays.

4. Scroll down to see the list of deployments in the Deployments table.
5. Select the check box to the left of the mss_samples deployment.
6. Click **Delete**.
7. Repeat the steps in the previous procedure to add the new sample deployment.

Viewing the samples in Workshop

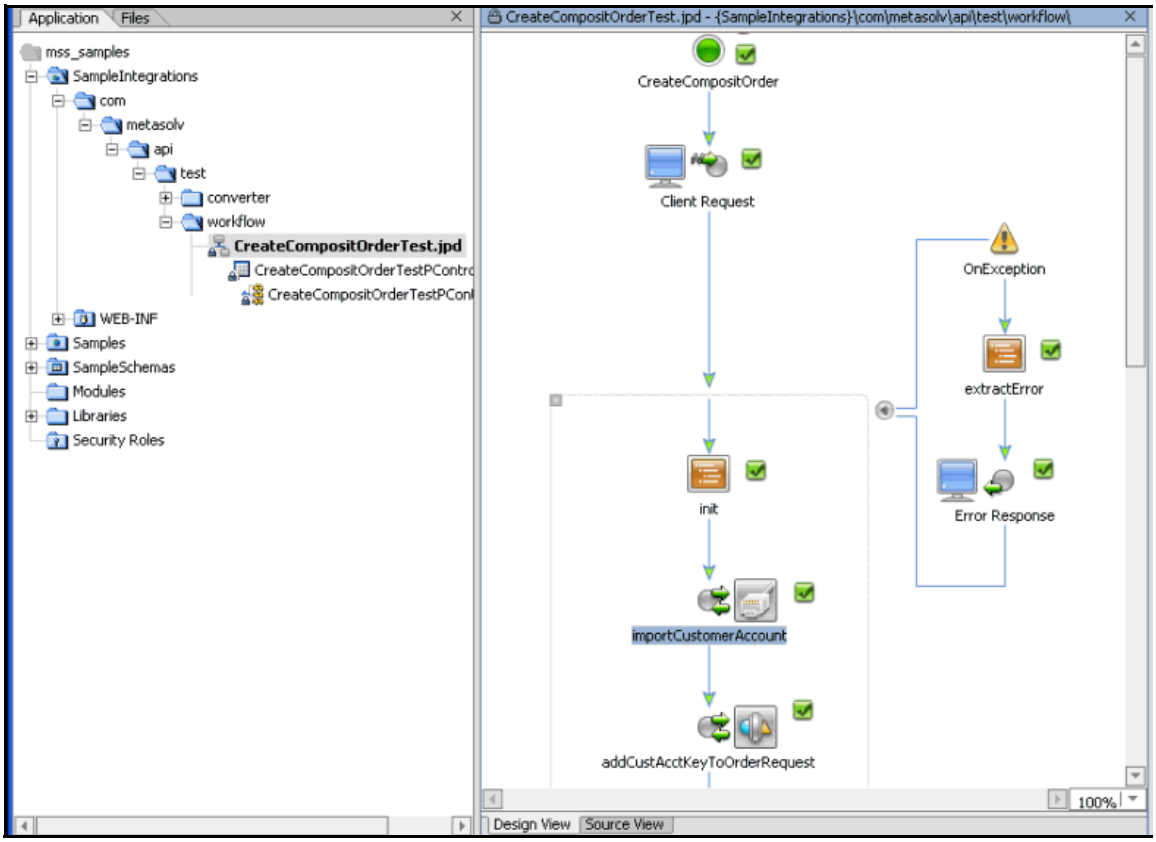
To view the samples in Workshop:

1. In the Navigator view, locate the **mss_samples** directory.
2. Navigate to the **src/com/metasolv/api/workflow** directory.
3. Double-click a .java file under one of the following **workflow** subdirectories: **customer**, **events**, **inventory**, **order**.

   The workflow appears on the Workshop canvas.

Composite sample

A composite order is also included in the samples. The composite order combines multiple methods to get a desired result. The following figure shows where in the directory structure the composite order sample is located and how the workflow looks displayed in Design View.



B

Appendix B: Navigating The XSD

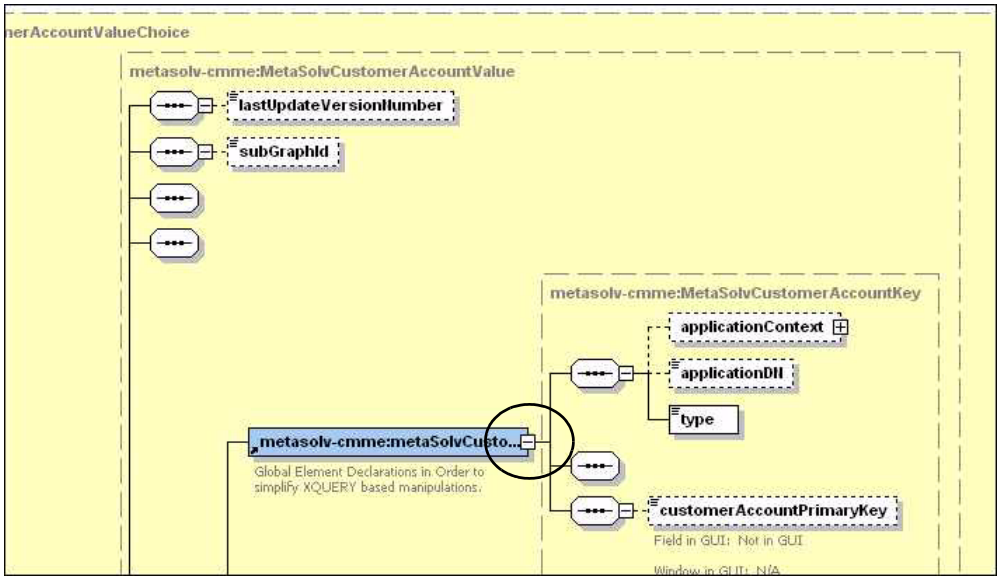
The information in this appendix describes how to navigate through the Request and Response structures defined in the xsd. The Request and Response structures defined in the xsd are used by the control methods as input and output parameters. Several examples will show screen shots of the xsd in various states of expansion. You can view the xsd in such a manner by using a tool such as XMLSpy.

XMLSpy offers several ways to view xml. You may be used to a more traditional view of xml, such as the text view shown below. However, this can become very difficult to read when dealing with large structures because typically elements within the structure reference other structures, which you then have scroll around to locate. Therefore, these examples show how to view the xml using the "Schema/WSDL Design View", which allows you to the view top level structures and then expand and collapse them as needed. Viewing the xml structure in this manner automatically pulls in the referenced structures, so there is no need to scroll around to locate them.

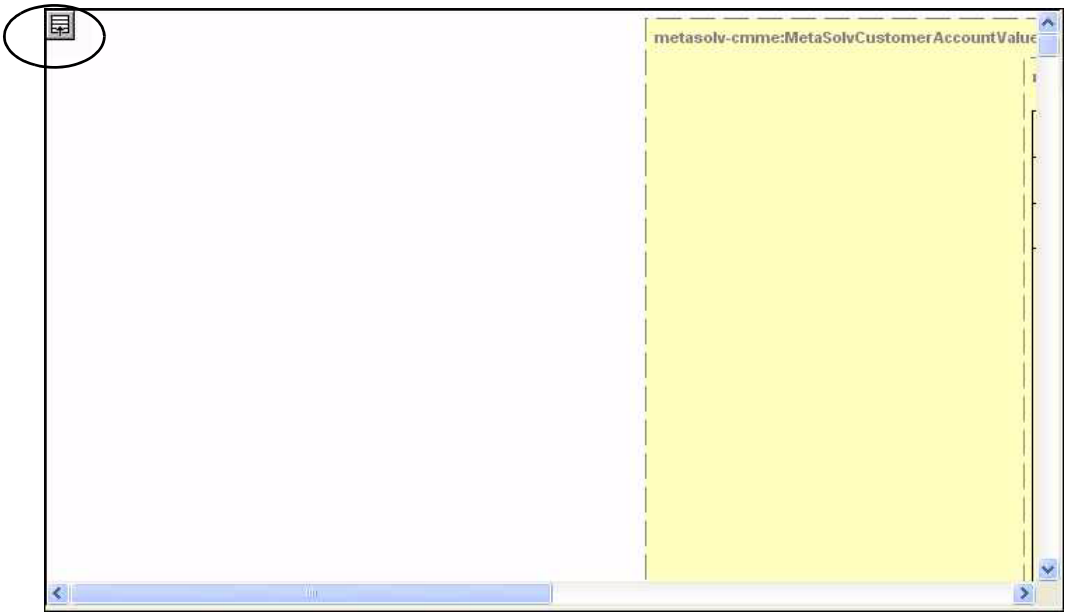
```
    <element name="updateCustomerAccountByValueRequest">
        <annotation>
            <appinfo>MetaSolv Customer Management</appinfo>
            <documentation>Creates a new customer account or modifies an existing customer account.
Response: updateCustomerAccountByValueResponse.
</documentation>
        </annotation>
        <complexType>
            <sequence>
                <element ref="metasolv-cmme:value"/>
            </sequence>
        </complexType>
    </element>
    <element name="updateCustomerAccountByValueResponse">
        <annotation>
            <appinfo>MetaSolv Customer Management</appinfo>
            <documentation>Returns the customer account key corresponding to a customer created or updated in the system.
Response: Customer account key.
</documentation>
        </annotation>
        <complexType>
            <sequence>
                <element ref="metasolv-cmme:key"/>
            </sequence>
        </complexType>
    </element>
```

Before going through the examples, note that the following two screen shots apply to all examples. The examples will explain how to navigate through an xml structure by *expanding*

the structure as you read it. If you wish to *collapse* the structures, you can collapse an individual structure by clicking on the "-" as shown below.



Or, you can collapse the entire structure by clicking on the collapse button as shown below. This button is only visible in the upper left corner, so you must scroll all the way up and all the way to the left to see it.



Example 1: importCustomerAccount

This example shows how a typical XML API import method works. The importCustomerAccount method is defined in the CustomerManagement control class as follows. You can see a full list of controls in Appendix B.

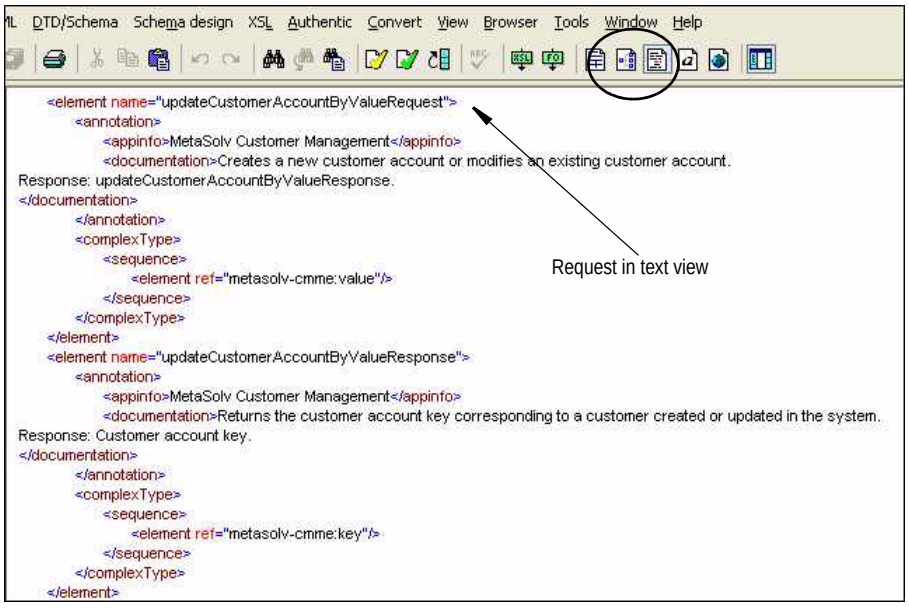
```
com.metasolv.mip.customerManagementAPI.UpdateCustomerAccountByValueRes
ponseDocument
importCustomerAccount(com.metasolv.mip.customerManagementAPI.UpdateCus
tomerAccountByValueRequestDocument request);
```

Request structure

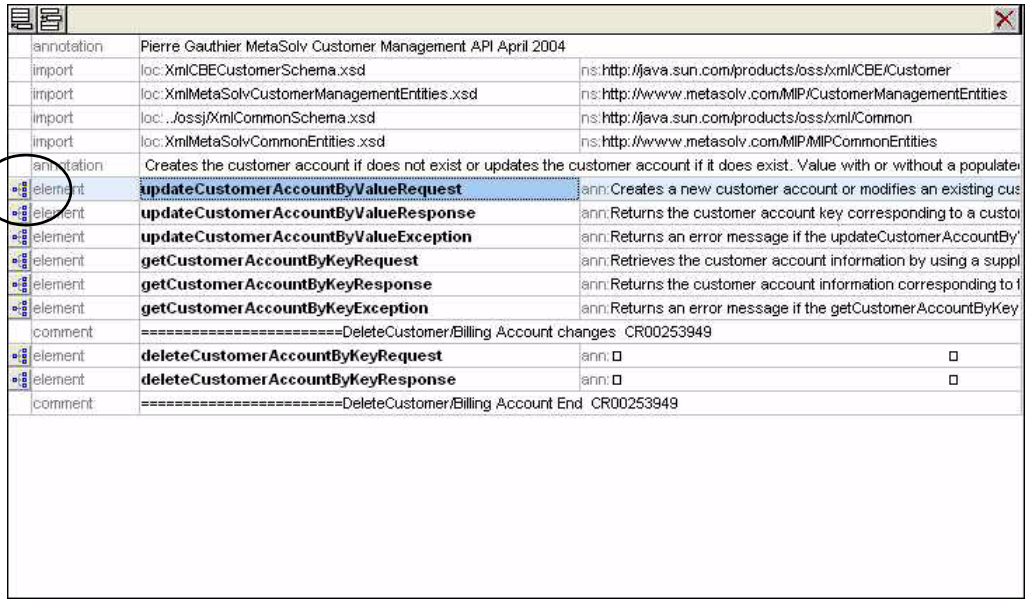
The method defines one input parameter, "request", which is defined as type com.metasolv.mip.customerManagementAPI.UpdateCustomerAccountByValueRequestDocument. This tells us that an xml structure named *updateCustomerAccountByValueRequest* is defined in the CustomerManagementAPI.xsd. Therefore, we will examine the xml structure *updateCustomerAccountByValueRequest*, which is the input to the control method importCustomerAccount.

The following steps will walk you through viewing and understanding the UpdateCustomerAccountByValueRequest structure.

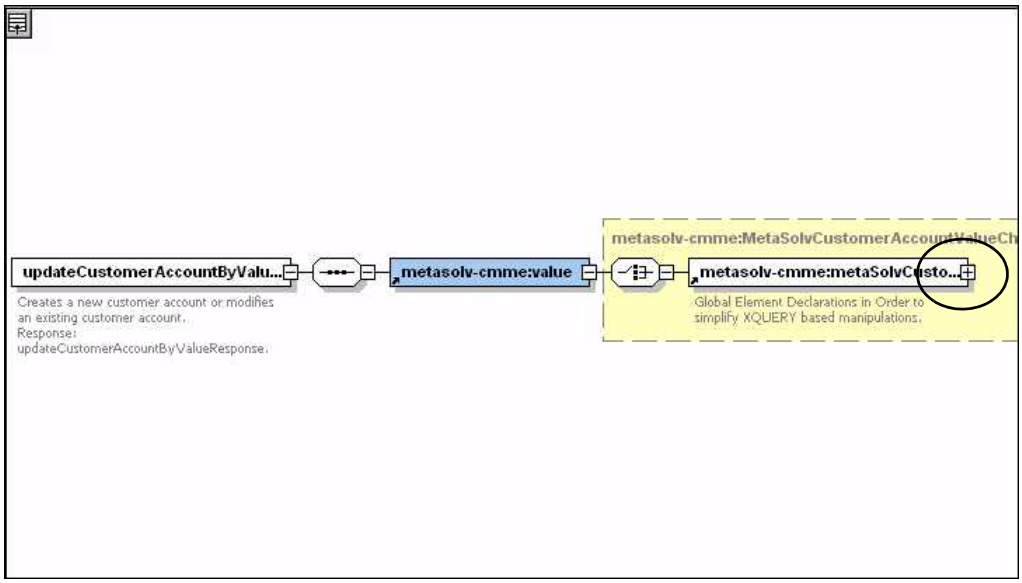
1. Open the CustomerManagementAPI.xsd, using an xml tool such as XMLSpy.
2. If the xml file comes up as text view, change the view by selecting a different view as shown below.



3. The top-level structures are now clearly listed. Expand the updateCustomerAccountByValueRequest structure by clicking on the expand button as indicated below.

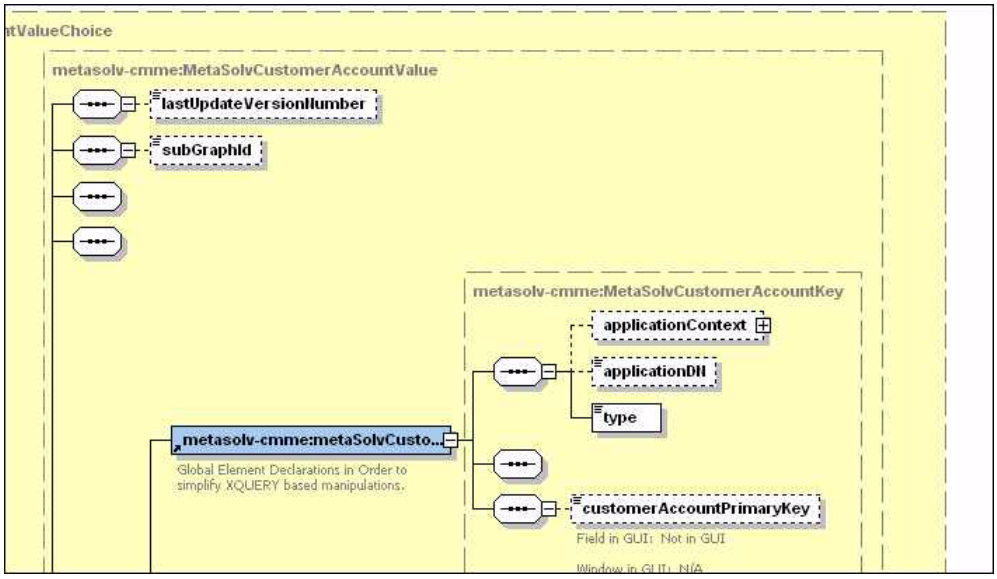


4. You are now viewing the contents of the updateCustomerAccountByValueRequest structure. Further expand the udpateCustomerAccountByValueRequest structure by clicking on the expand button as indicated below.

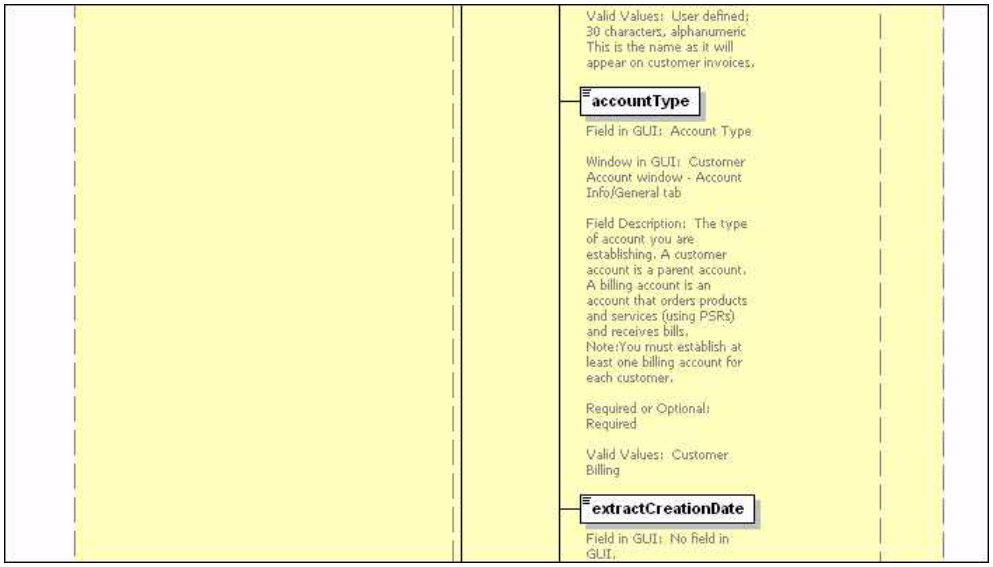


5. You are now viewing elements that define a data type. These elements will house the data that comprise the customer account being imported. For example, lastUpdateVersionNumber and subGraphId. The metaSolvCustomerAccountKey defines another structure. Click the "+" to further expand the structure.

6. You are now viewing additional elements that define a data type, such as applicationDN, type, and customerAccountPrimaryKey.



7. Scroll down, and you can view field properties, such as Field in GUI, Window in GUI, Field Description, Required or Optional, and Valid Values.

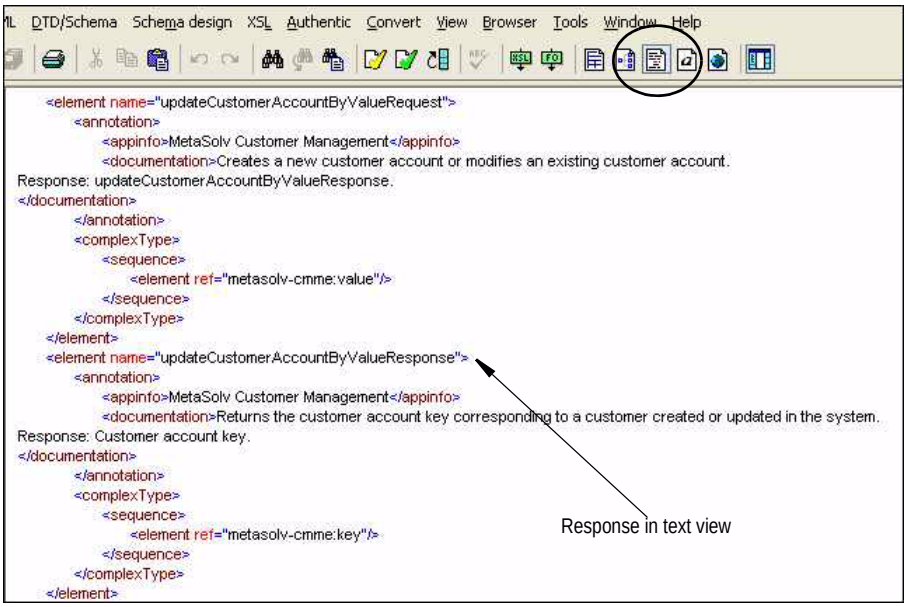


Response structure

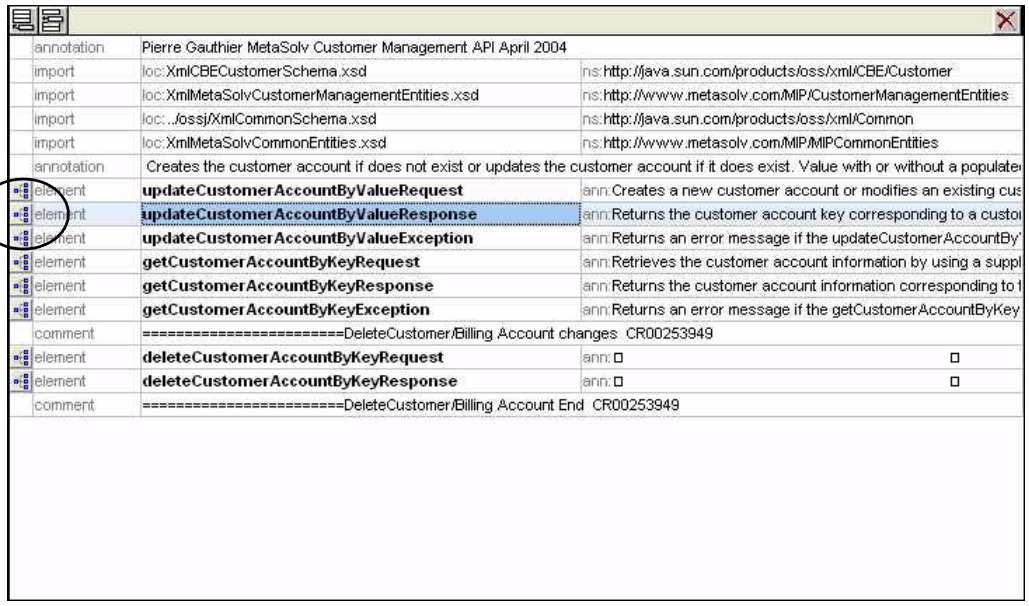
The method defines it's return as type com.metasolv.mip.customerManagementAPI.UpdateCustomerAccountByValueResponseDocument. This tells us that an xml structure named *updateCustomerAccountByValueResponse* is defined in the CustomerManagementAPI.xsd. Therefore, we will examine the xml structure *updateCustomerAccountByValueResponse*, which is what is returned by the control method importCustomerAccount.

The following steps will walk you through viewing and understanding the UpdateCustomerAccountByValueResponse structure.

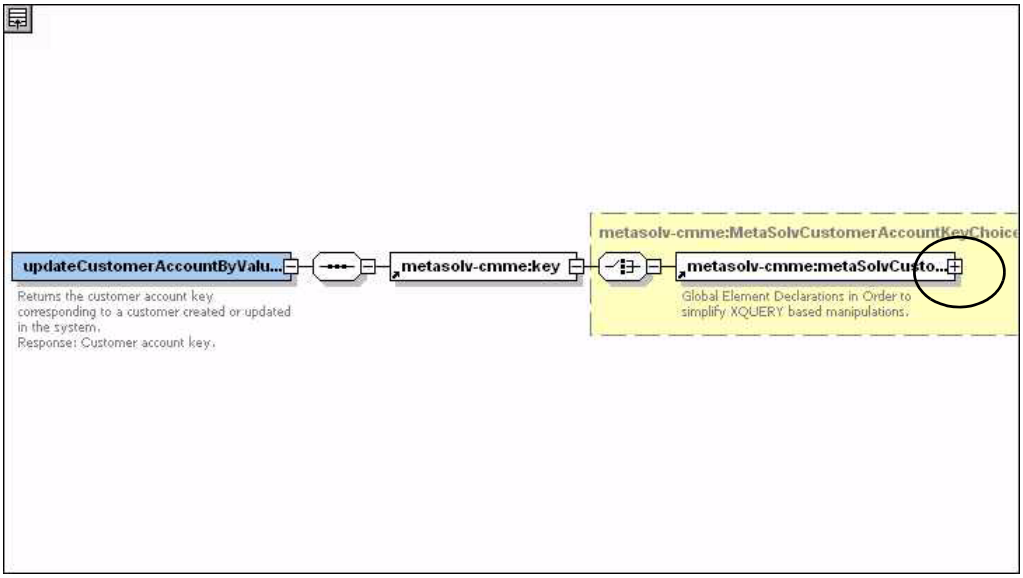
1. Open the CustomerManagementAPI.xsd, using an xml tool such as XMLSpy.
2. If the xml file comes up as text view, change the view by selecting a different view as shown below.



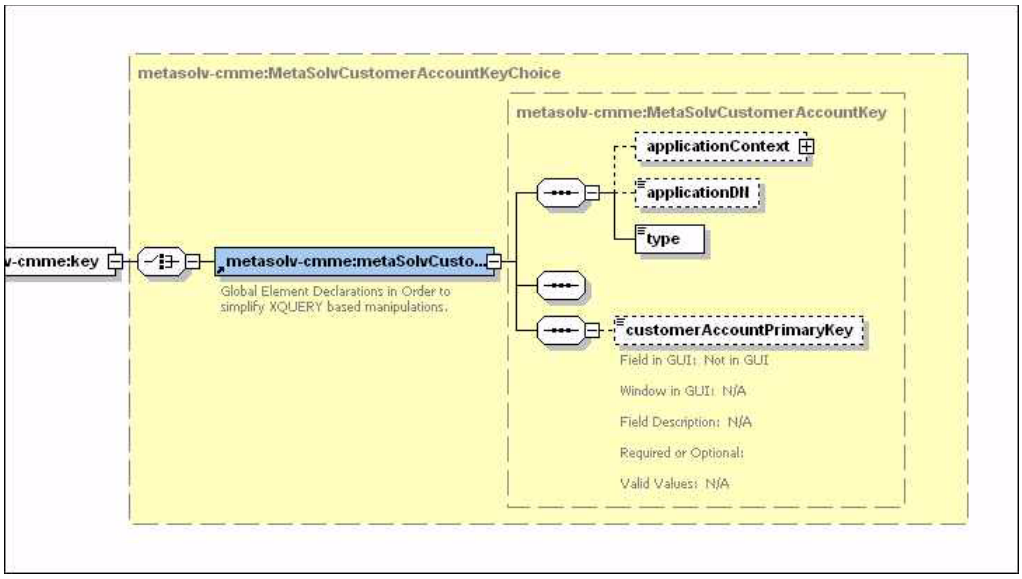
3. The top-level structures are now clearly listed. Expand the updateCustomerAccountByValueResponse structure by clicking on the expand button as indicated below.



4. You are now viewing the contents of the updateCustomerAccountByValueResponse structure. Further expand the udpateCustomerAccountByValueResponse structure by clicking on the expand button as indicated below.



5. You are now viewing elements that define a data type. These elements will house the data that is returned in the response regarding the customer account that was imported. For example, applicationDN, type, and customerAccountPrimaryKey.



Example 2: getCustomerAccountByKey

This example shows how a typical XML API export method works.The getCustomerAccountByKey method is defined in the CustomerManagement control class as follows. You can see a full list of controls in Appendix B.

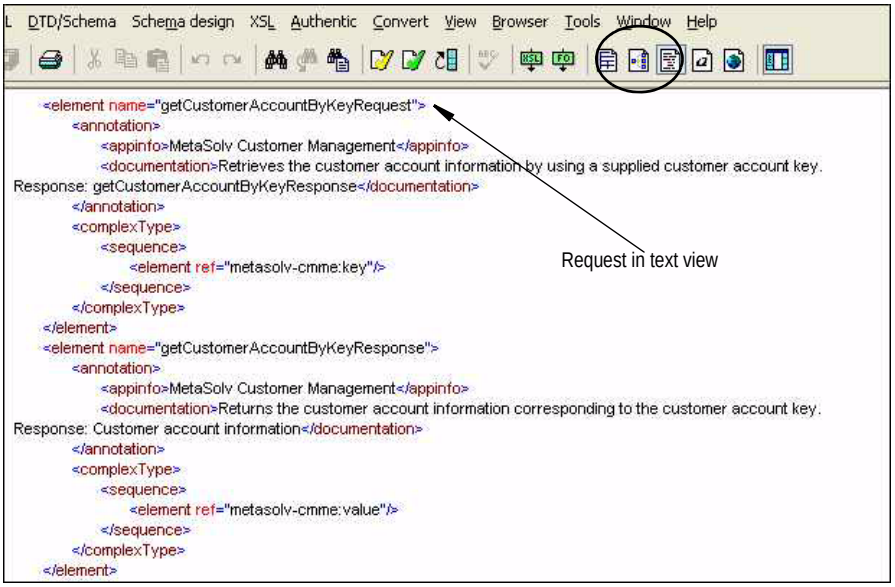
```
com.metasolv.mip.customerManagementAPI.GetCustomerAccountByKeyResponse
Document
getCustomerAccountByKey(com.metasolv.mip.customerManagementAPI.GetCust
omerAccountByKeyRequestDocument request);
```

Request structure

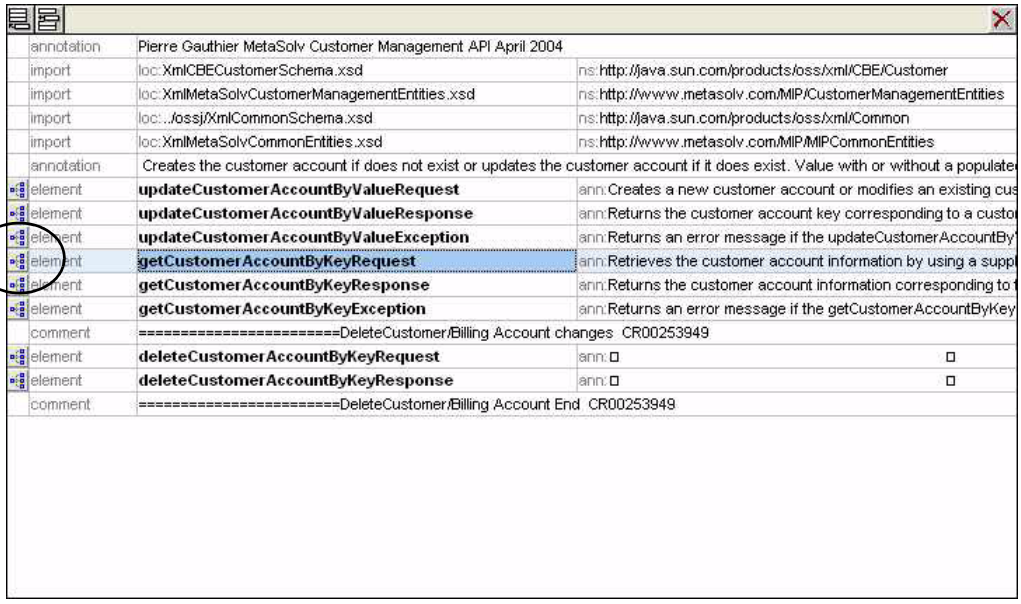
The method defines one input parameter, "request", which is defined as type com.metasolv.mip.customerManagementAPI.GetCustomerAccountByKeyRequestDocument. This tells us that an xml structure named *getCustomerAccountByKeyRequest* is defined in the CustomerManagementAPI.xsd. Therefore, we will examine the xml structure *getCustomerAccountByKeyRequest*, which is the input to the control method getCustomerAccountByKey.

The following steps will walk you through viewing and understanding the GetCustomerAccountByKeyRequest structure.

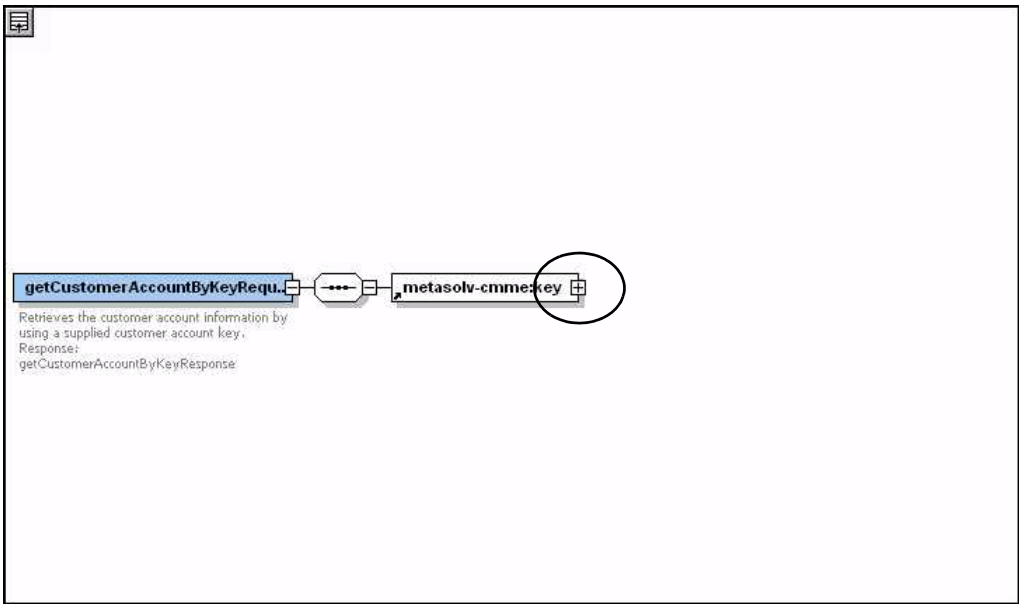
1. Open the CustomerManagementAPI.xsd, using an xml tool such as XMLSpy.
2. If the xml file comes up as text view, change the view by selecting a different view as shown below.



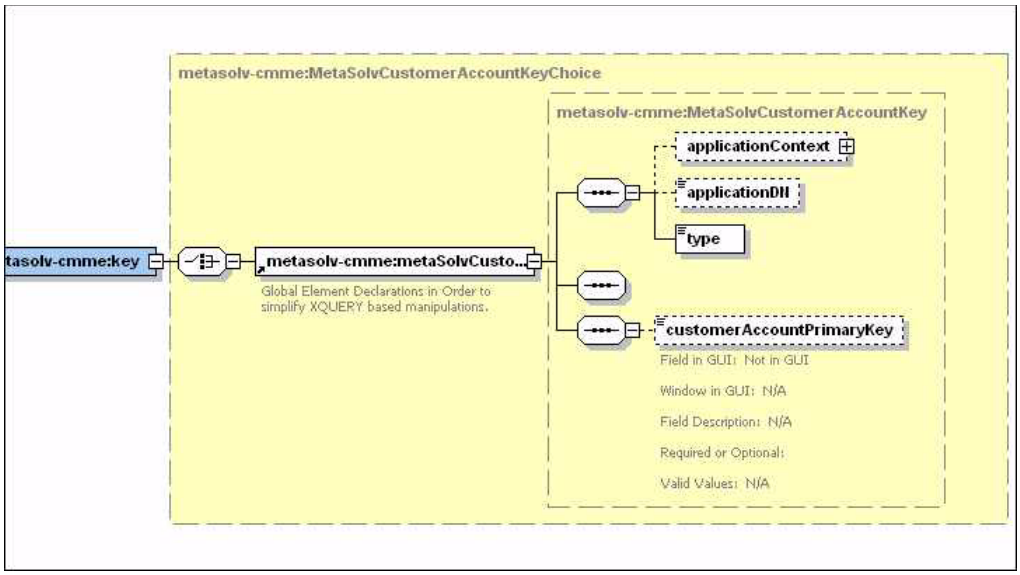
3. The top-level structures are now clearly listed. Expand the getCustomerAccountByKeyRequest structure by clicking on the expand button as indicated below.



4. You are now viewing the contents of the getCustomerAccountByKeyReqeust structure. Further expand the getCustomerAccountByKeyRequest structure by clicking on the expand button as indicated below.



5. You are now viewing elements that define a data type. These elements will house the data that is required to get the customer account. For example, applicationDN, type, and customerAccountPrimaryKey.

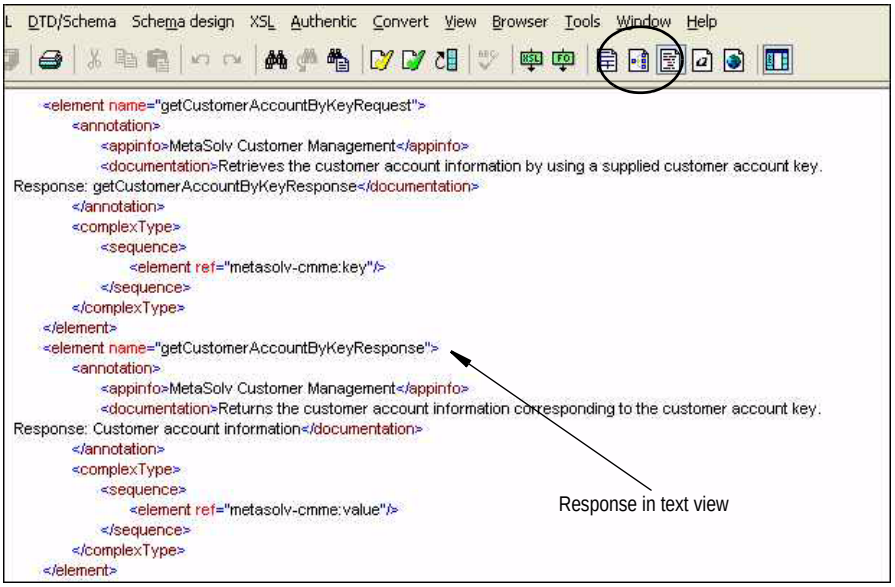


Response structure

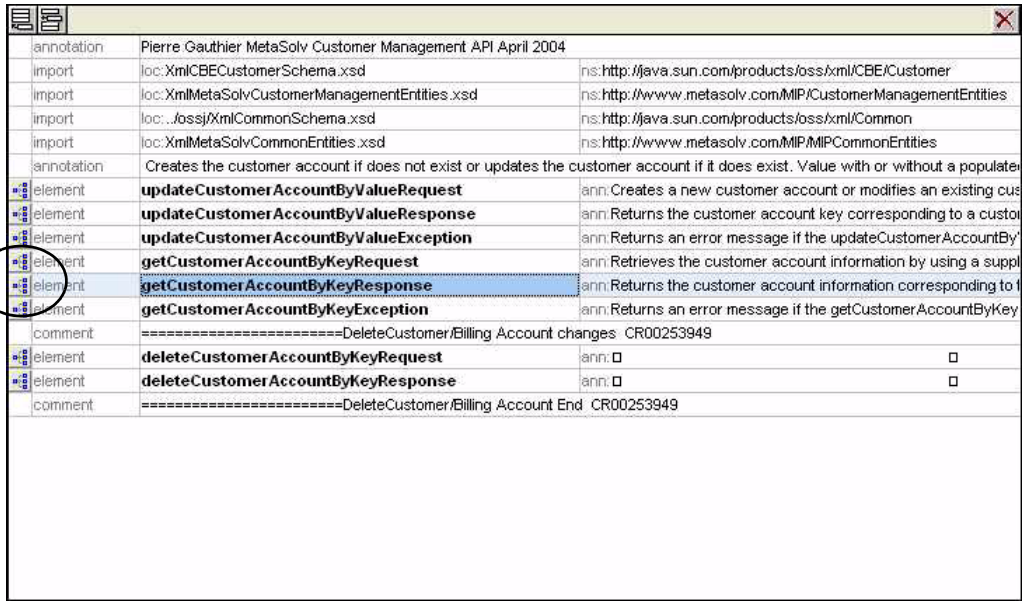
The method defines it's return as type com.metasolv.mip.customerManagementAPI.GetCustomerAccountByKeyResponseDocument. This tells us that an xml structure named *getCustomerAccountByKeyResponse* is defined in the CustomerManagementAPI.xsd. Therefore, we will examine the xml structure *getCustomerAccountByKeyResponse*, which is what is returned by the control method getCustomerAccountByKey.

The following steps will walk you through viewing and understanding the UpdateCustomerAccountByValueRequest structure.

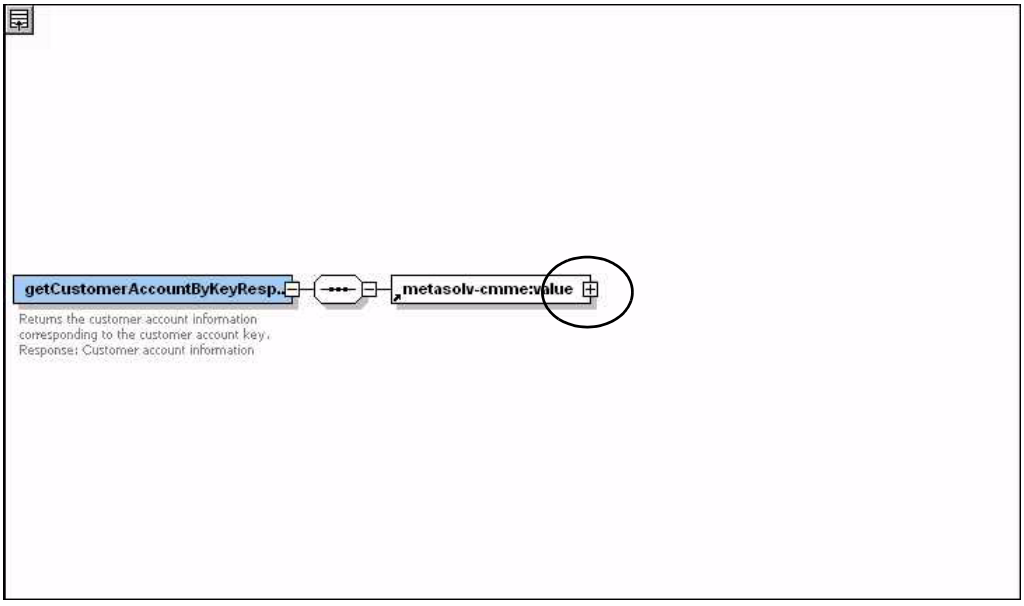
1. Open the CustomerManagementAPI.xsd, using an xml tool such as XMLSpy.
2. If the xml file comes up as text view, change the view by selecting a different view as shown below.



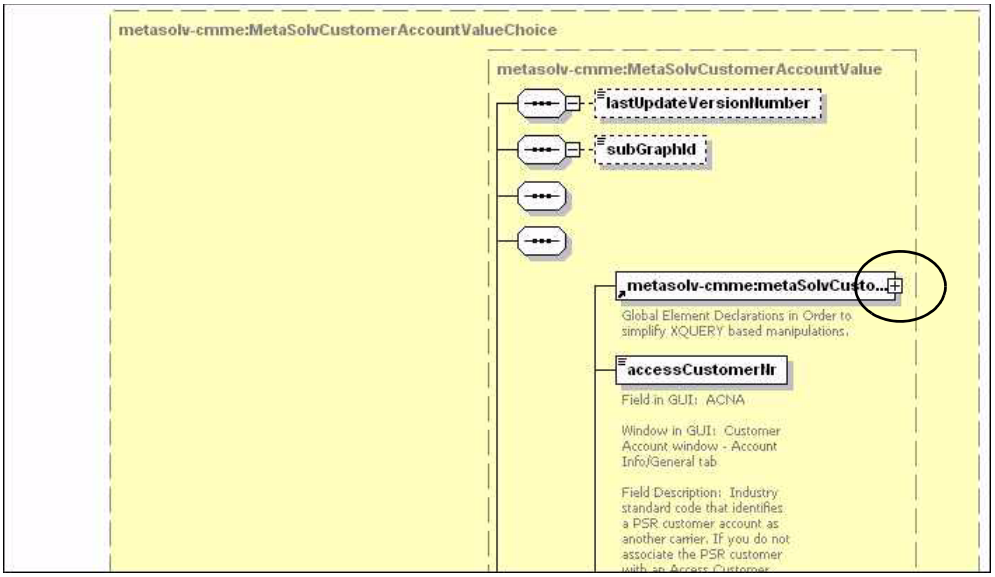
3. The top-level structures are now clearly listed. Expand the getCustomerAccountByKeyResponse structure by clicking on the expand button as indicated below.



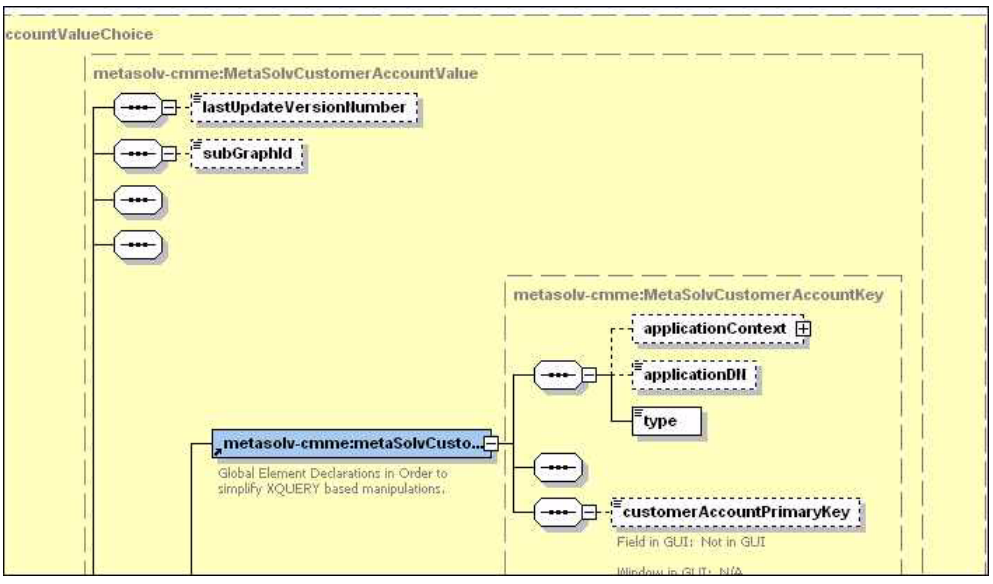
4. You are now viewing the contents of the getCustomerAccountByKeyResponse structure. Further expand the getCustomerAccountByKeyResponse structure by clicking on the expand button as indicated below.



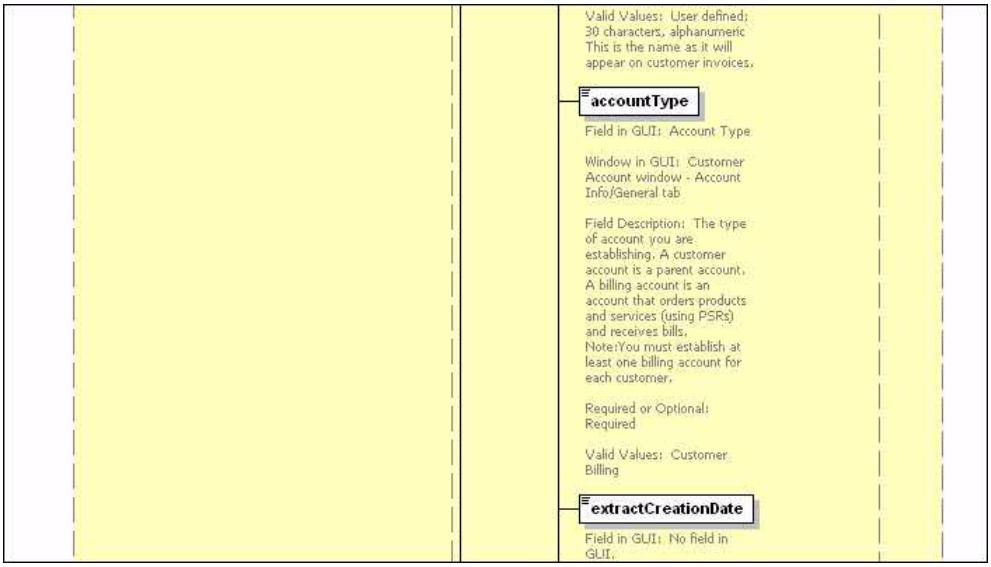
5. You are now viewing elements that define a data type. These elements will house the data that is returned in the response regarding the customer account being exported. For example, lastUpdateVersionNumber, subGraphId, and accessCustomerNr. The metaSolvCustomerAccountKey defines another structure. Click the "+" to further expand the structure.



6. You are now viewing additional elements that define a data type, such as applicationDN, type, and customerAccountPrimaryKey.



7. Scroll down, and you can view field properties, such as Field in GUI, Window in GUI, Field Description, Required or Optional, and Valid Values.



Example 3: createEntityByValueRequest

This example shows how a typical XML API method that defines choices of structures within the Request and Response structures works.The createEntityByValueRequest method is defined in the InventoryManagement control class as follows. You can see a full list of controls in Appendix B.

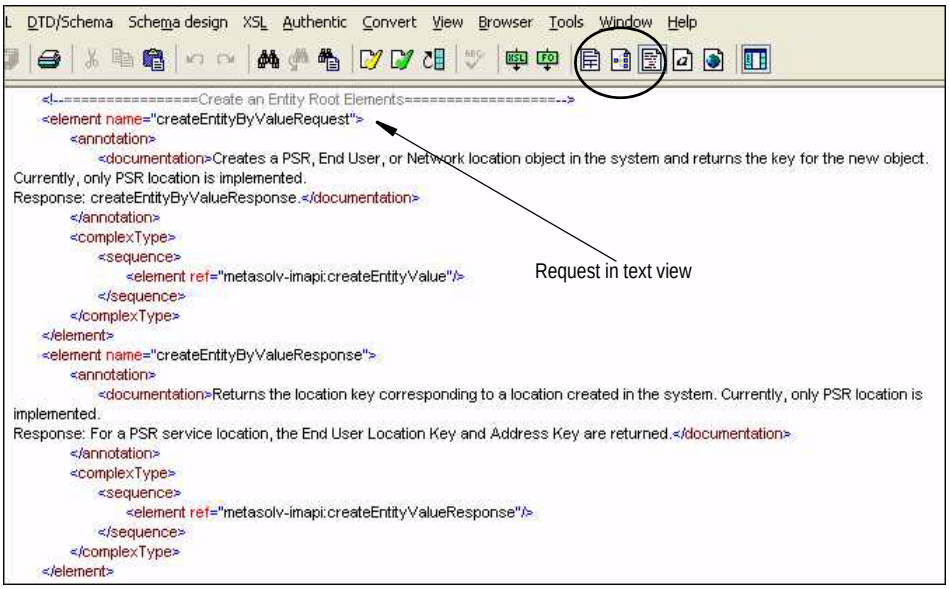
```
com.metasolv.mip.inventoryManagementAPI.CreateEntityByValueResponseDoc
ument
createEntityByValueRequest(com.metasolv.mip.inventoryManagementAPI.Cre
ateEntityByValueRequestDocument request);
```

Request structure

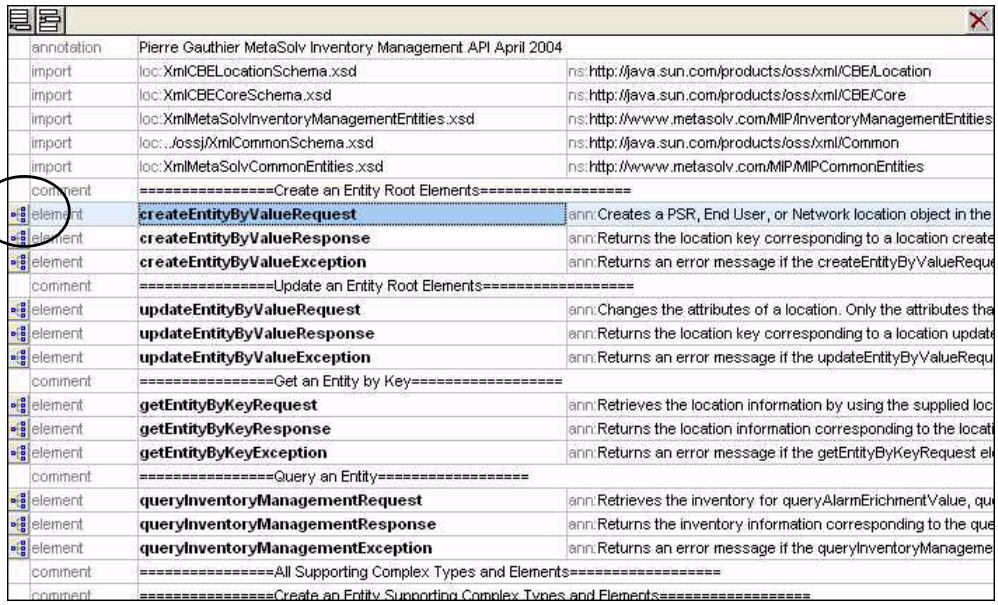
The method defines one input parameter, "request", which is defined as type com.metasolv.mip.inventoryManagementAPI.CreateEntityByValueRequestDocument. This tells us that an xml structure named *createEntityByValueRequest* is defined in the InventoryManagementAPI.xsd. Therefore, we will examine the xml structure *createEntityByValueRequest*, which is the input to the control method createEntityByValueRequest.

The following steps will walk you through viewing and understanding the CreateEntityByValueRequest structure.

1. Open the InventoryManagementAPI.xsd, using an xml tool such as XMLSpy.
2. If the xml file comes up as text view, change the view by selecting a different view as shown below.

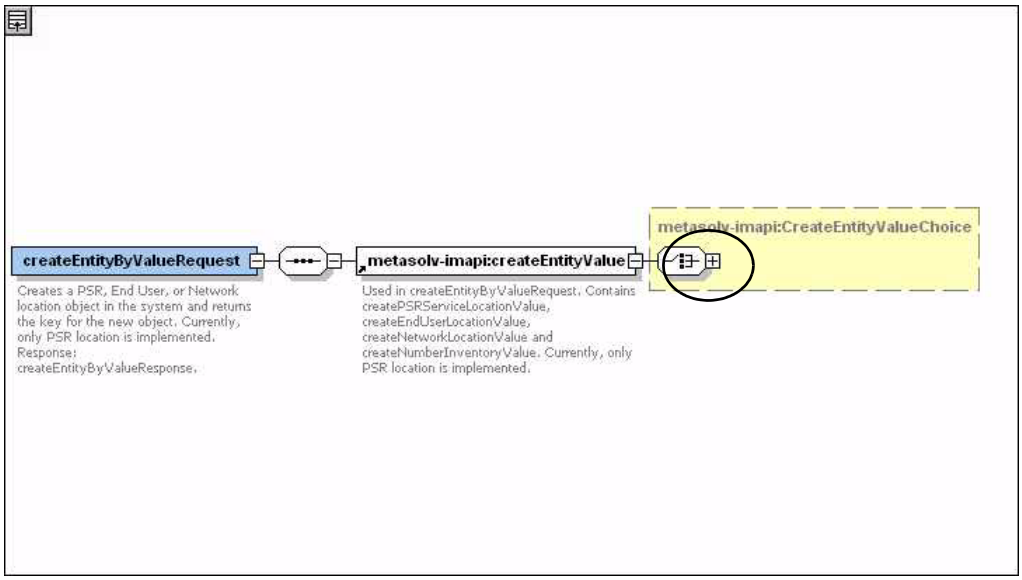


3. The top-level structures are now clearly listed. Expand the createEntityByValueRequest structure by clicking on the expand button as indicated below.

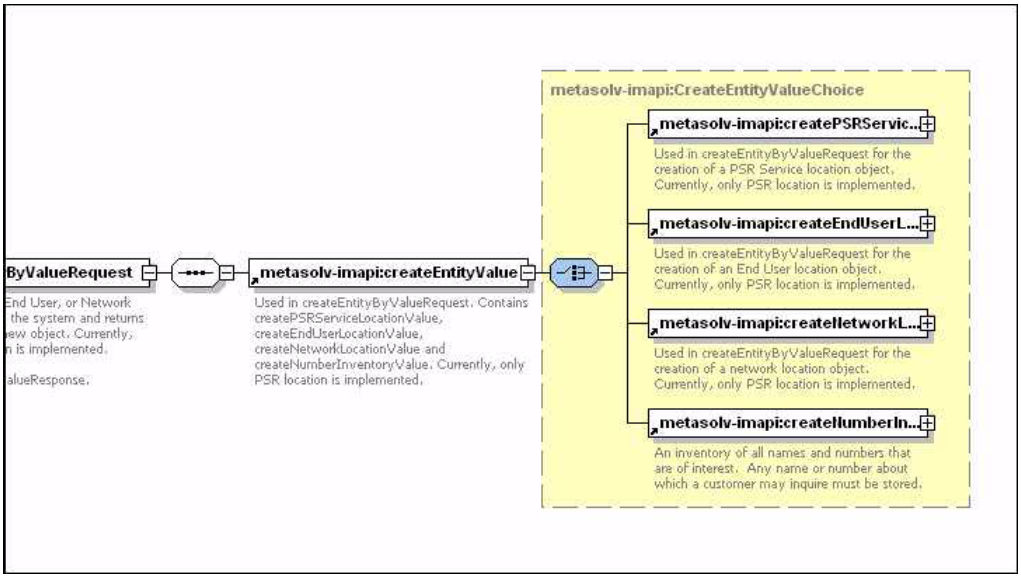


	annotation	Pierre Gauthier MetaSolv Inventory Management API April 2004	
	import	loc:XmlCBELocationSchema.xsd	ns:http://java.sun.com/products/oss/xml/CBE/Location
	import	loc:XmlCBECoreSchema.xsd	ns:http://java.sun.com/products/oss/xml/CBE/Core
	import	loc:XmlMetaSolvInventoryManagementEntities.xsd	ns:http://www.metasolv.com/MIP/InventoryManagementEntities
	import	loc:../ossj/XmlCommonSchema.xsd	ns:http://java.sun.com/products/oss/xml/Common
	import	loc:XmlMetaSolvCommonEntities.xsd	ns:http://www.metasolv.com/MIP/MIPCommonEntities
	comment	==========Create an Entity Root Elements==========	
	element	**createEntityByValueRequest**	ann:Creates a PSR, End User, or Network location object in the
	element	**createEntityByValueResponse**	ann:Returns the location key corresponding to a location create
	element	**createEntityByValueException**	ann:Returns an error message if the createEntityByValueReque
	comment	==========Update an Entity Root Elements==========	
	element	**updateEntityByValueRequest**	ann:Changes the attributes of a location. Only the attributes tha
	element	**updateEntityByValueResponse**	ann:Returns the location key corresponding to a location updat
	element	**updateEntityByValueException**	ann:Returns an error message if the updateEntityByValueRequ
	comment	==========Get an Entity by Key==========	
	element	**getEntityByKeyRequest**	ann:Retrieves the location information by using the supplied loc
	element	**getEntityByKeyResponse**	ann:Returns the location information corresponding to the locati
	element	**getEntityByKeyException**	ann:Returns an error message if the getEntityByKeyRequest el
	comment	==========Query an Entity==========	
	element	**queryInventoryManagementRequest**	ann:Retrieves the inventory for queryAlarmEnrichmentValue, qu
	element	**queryInventoryManagementResponse**	ann:Returns the inventory information corresponding to the que
	element	**queryInventoryManagementException**	ann:Returns an error message if the queryInventoryManageme
	comment	==========All Supporting Complex Types and Elements==========	
	comment	==========Create an Entity Supporting Complex Types and Elements==========	

4. You are now viewing the contents of the createEntityByValueReqeust structure. Further expand the createEntityByValueReqeust structure by clicking on the expand button as indicated below.



5. You are now viewing sub-structures that are a choice. For example, if you wish to create a PSR Service Location you would populate the first choice of sub-structures with input data; if you wish to create a Network Location you would populate the third choice of sub-structures with input data. Do not include the remaining empty choice structures in the request.

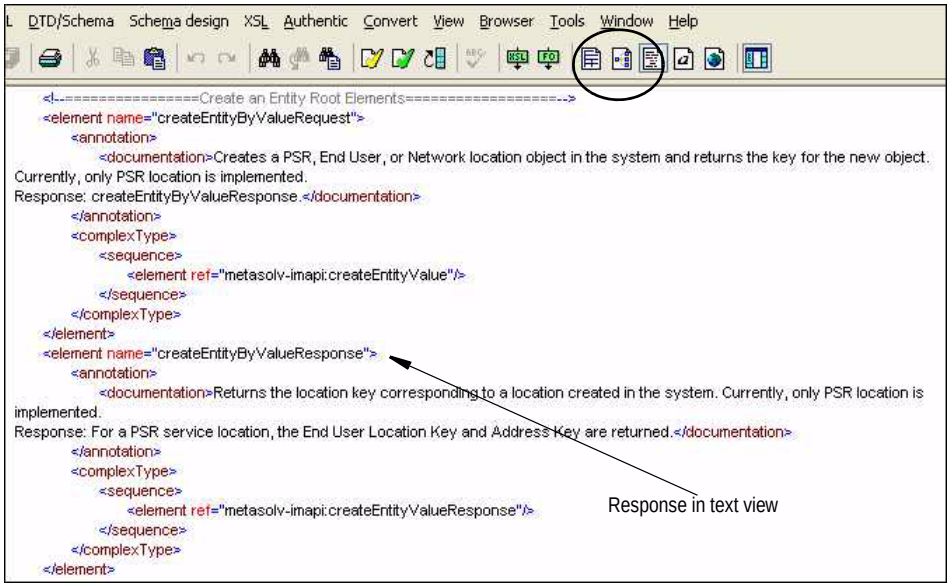


Response structure

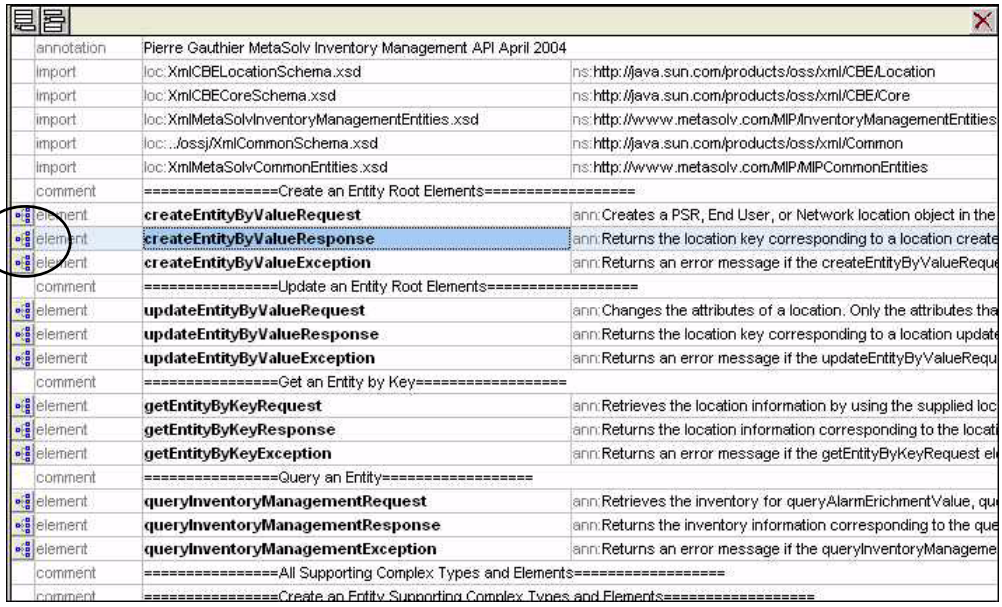
The method defines it's return as type com.metasolv.mip.inventoryManagementAPI.CreateEntityByValueResponseDocument. This tells us that an xml structure named *createEntityByValueResponse* is defined in the InventoryManagementAPI.xsd. Therefore, we will examine the xml structure *createEntityByValueResponse*, which is what is returned by the control method createEntityByValueRequest.

The following steps will walk you through viewing and understanding the CreateEntityByValueRequest structure.

1. Open the InventoryManagementAPI.xsd, using an xml tool such as XMLSpy.
2. If the xml file comes up as text view, change the view by selecting a different view as shown below.

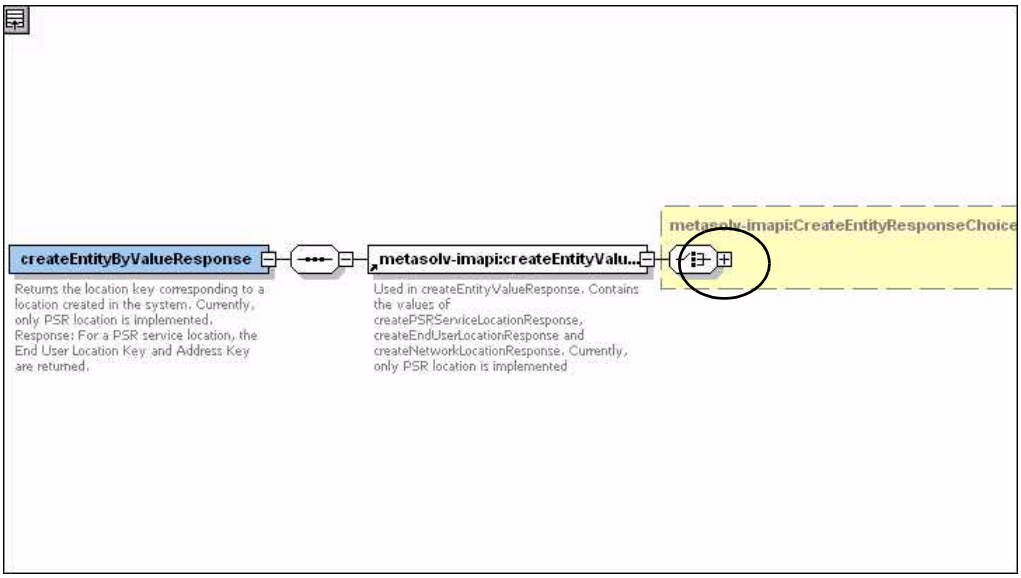


3. The top-level structures are now clearly listed. Expand the createEntityByValueResponse structure by clicking on the expand button as indicated below.

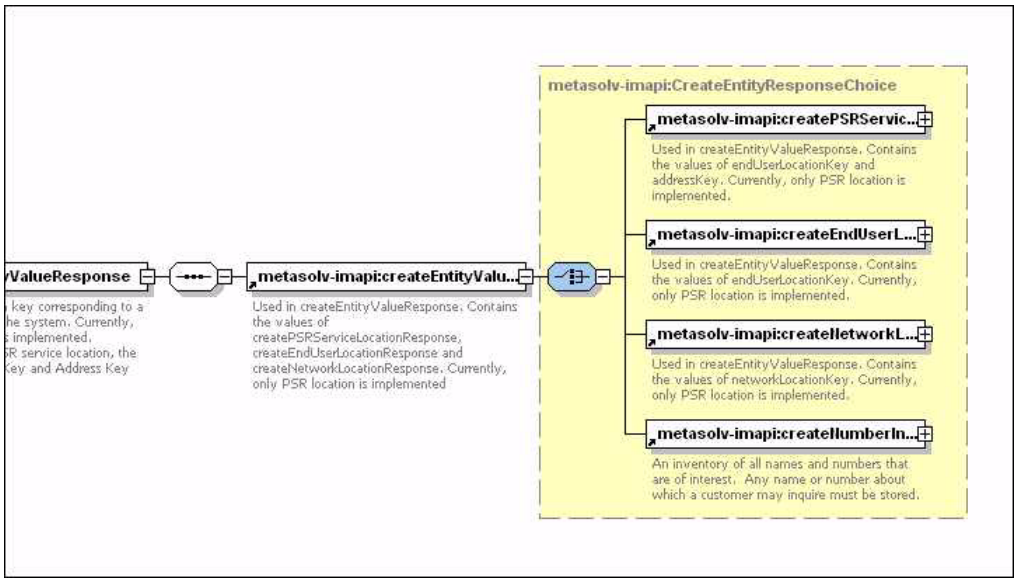


	annotation	Pierre Gauthier MetaSolv Inventory Management API April 2004	
	import	loc:XmlCBELocationSchema.xsd	ns:http://java.sun.com/products/oss/xml/CBE/Location
	import	loc:XmlCBECoreSchema.xsd	ns:http://java.sun.com/products/oss/xml/CBE/Core
	import	loc:XmlMetaSolvInventoryManagementEntities.xsd	ns:http://www.metasolv.com/MIP/InventoryManagementEntities
	import	loc:../ossj/XmlCommonSchema.xsd	ns:http://java.sun.com/products/oss/xml/Common
	import	loc:XmlMetaSolvCommonEntities.xsd	ns:http://www.metasolv.com/MIP/MIPCommonEntities
	comment	==========Create an Entity Root Elements==========	
	element	**createEntityByValueRequest**	ann:Creates a PSR, End User, or Network location object in the
	element	**createEntityByValueResponse**	ann:Returns the location key corresponding to a location create
	element	**createEntityByValueException**	ann:Returns an error message if the createEntityByValueReque
	comment	==========Update an Entity Root Elements==========	
	element	**updateEntityByValueRequest**	ann:Changes the attributes of a location. Only the attributes tha
	element	**updateEntityByValueResponse**	ann:Returns the location key corresponding to a location update
	element	**updateEntityByValueException**	ann:Returns an error message if the updateEntityByValueRequ
	comment	==========Get an Entity by Key==========	
	element	**getEntityByKeyRequest**	ann:Retrieves the location information by using the supplied loc
	element	**getEntityByKeyResponse**	ann:Returns the location information corresponding to the locati
	element	**getEntityByKeyException**	ann:Returns an error message if the getEntityByKeyRequest el
	comment	==========Query an Entity==========	
	element	**queryInventoryManagementRequest**	ann:Retrieves the inventory for queryAlarmEnrichmentValue, qu
	element	**queryInventoryManagementResponse**	ann:Returns the inventory information corresponding to the que
	element	**queryInventoryManagementException**	ann:Returns an error message if the queryInventoryManageme
	comment	==========All Supporting Complex Types and Elements==========	
	comment	==========Create an Entity Supporting Complex Types and Elements==========	

4. You are now viewing the contents of the createEntityByValueResponse structure. Further expand the createEntityByValueResponse structure by clicking on the expand button as indicated below.



5. You are now viewing sub-structures that are a choice. For example, if you created a PSR Service Location, the first choice of sub-structures will be populated in the response; if you created a Network Location, the third choice of sub-structures will be populated in the in the response. Only one of the sub-structures will be returned in response.



C

Appendix C: XML API Methods

The information provided for each method includes:

- Control name
- XML-defined Request name
- XML-defined Response name
- Description of method
- Input structure
- Response structure
- MetaSolv Solution Path > Page (where applicable)
- Additional information (where applicable)

Customer Management API

The Customer Management API is a collection of methods that provide the ability to import and maintain customer accounts in the MetaSolv Solution database from an outside source, without using the MetaSolv Solution GUI. The XmlMetaSolvCustomerManagementAPI.xsd defines the following methods:

importCustomerAccount

updateCustomerAccountByValueRequest

updateCustomerAccountByValueResponse

Description	This method either creates or updates the specified customer account based on the input data	
Input Structure	MetaSolvCustomerAccountValue Choice	MetaSolvCustomerAcountValue
Response Structure	MetaSolvCustomerAccountKey	
MetaSolv Solution Path > Page	Order Management > Customer Account	

Table 2: importCustomerAccount

Additional Information

This method provides the ability to import a new customer account, and to update an existing customer account. When the customerNr and suppType are not populated, the code processes the request as an import. When the customerNr and suppType are populated, the code processes the request as an update. When importing a new customer, the customerNr and suppType must not be populated. Also, importing a new customer requires additional data that is not required for an update.

getCustomerAccountByKey

getCustomerAccountByKeyRequest

getCustomerAccountByKeyResponse

Description	This method returns existing customer account information based on the input customer account key.	
Input Structure	MetaSolvCustomerAccountKey Choice	MetaSolvCustomerAccountKey
Response Structure	MetaSolvCustomerAccountValue Choice	MetaSolvCustomerAccountValue
MetaSolv Solution Path > Page	Order Management > Customer Account	

Table 3: getCustomerAccountByKey

deleteCustomerRequest

deleteCustomerAccountByKeyRequest

deleteCustomerAccountByKeyResposne

Description	This method deletes an existing customer account based on the input customer account key.	
Input Structure	MetaSolvCustomerAccountKey Choice	MetaSolvCustomerAccountKey
Response Structure	status (String)	
MetaSolv Solution Path > Page	Order Management > Customer Account	

Table 4: deleteCustomerRequest

Order Management API

The Order Management API is a collection of methods that provide the ability to import and maintain orders in the MetaSolv Solution database from an outside source, without using the MetaSolv Solution GUI. Orders include Internal Service Requests (ISRs) and Product Service Requests (PSRs). Local Service Requests (LSRs) are handled by a separate API, Local Service Request API. The XmlMetaSolvOrderManagementAPI.xsd defines the following methods:

queryOrderManagementRequest

queryOrderManagementRequest

queryOrderManagementResponse

Description	This method returns various order information based on the choice of input structure.	
Input Structure	MetaSolvQueryValueChoice	ValidateOrderQueryValue, GetTaskGWEventQueryValue, GetServReqTasksQueryValue, GetServiceRequestDLRsValue, GetDLRInfosByOrderAndServiceItemIdValue, GetDLRInfosByServiceItemIdInServiceValue, GetServItemReferenceValue, GetServItemsValue, GetProductCatalog,or GetOrderStatus
Response Structure	MetaSolvQueryResponseChoice	ValidateOrderQueryResponse, GetTaskGWEventQueryResponse, GetServReqTasksQueryResponse, GetServiceRequestDLRsResponse, GetDLRInfosByOrderAndServiceItemIdValue, GetDLRInfosByServiceItemIdInServiceValue, GetServItemReferenceValue, GetServItemsValue, GetDLRInfosByOrderAndServiceItemIdResponse, GetDLRInfosByServiceItemIdInServiceResponse, GetServItemReferenceResponse, GetServItemsResponse, GetProductCatalogResponse, or GetOrderStatusResponse
MetaSolv Solution Path > Page		

Table 5: queryOrderManagementRequest

startOrderByKeyRequest

startOrderByKeyRequest

startOrderByKeyResponse

Description	This method initiates the 'Finish Order' processing for the input order.	
Input Structure	MetaSolvOrderKeyChoice	OrderKey
Response Structure	MetaSolvOrderKeyChoice	OrderKey
MetaSolv Solution Path > Page	Order Management > Product Service Request or Order Management > Internal Service Request	

Table 6: startOrderByKeyRequest

Additional Information

This method needs to be called after a successful response from createOrderByValueRequest, and before calling assignProvisionPlanProcedureRequest. It is the API equivalent of clicking the GUI link for "Finish Order".

updateOrderManagementRequest

updateOrderManagementRequest

updateOrderManagementResponse

Description	This method updates various order information, based on the choice input structure and the input data.	
Input Structure	MetaSolvUpdateProcedureValueChoice	UpdateServicesInOrderProcedureValue, UpdateOrderTaskGWEventValue, CompleteTaskProcedureValue, UpdateOrderTaskEventProcedureValue, or ReopenTaskProcedureValue
Response Structure	MetaSolvUpdateProcedureValueResponseChoice	UpdateServicesInOrderProcedureResponse, UpdateOrderTaskGWEventResponse, CompleteTaskProcedureResponse, UpdateOrderTaskEventProcedureResponse, or ReopenTaskProcedureResponse
MetaSolv Solution Path > Page		

Table 7: updateOrderManagementRequest

getOrderByKeyRequest

getOrderByKeyRequest

getOrderByKeyResponse

Description	This method returns order information based on the input order key.	
Input Structure	MetaSolvOrderKeyChoice	OrderKey
Response Structure	MetaSolvOrderValueChoice	MetaSolvPSROrderValue or MetaSolvISROrderValue
MetaSolv Solution Path > Page	Order Management > Product Service Request or Order Management > Internal Service Request	

Table 8: getOrderByKeyRequest

createOrderByValueRequest

createOrderByValueRequest

createOrderByValueResponse

Description	This method creates a new PSR or ISR order based on the input data.	
Input Structure	MetaSolvOrderValueChoice	MetaSolvPSROrderValue or MetaSolvISROrderValue
Response Structure	MetaSolvOrderKeyChoice	OrderKey
MetaSolv Solution Path > Page	Order Management > Product Service Request or Order Management > Internal Service Request	

Table 9: createOrderByValueRequest

Additional Information

This method will create a PSR order or an ISR order, depending on the choice of the input structure. Both input structures define the same sub-structure, OrderHeaderType, which is where the mutual required data is defined.

assignProvisionPlanProcedureRequest

assignProvisionPlanProcedureRequest

assignProvisionPlanProcedureResponse

Description	This method assigns a provisioning plan to an order based on the input data.	
Input Structure	AssignProvisionPlanProcedure Value	ProvisionPlanValue
Response Structure	MetaSolvOrderKeyChoice	OrderKey
MetaSolv Solution Path > Page		

Table 10: assignProvisionPlanProcedureRequest

getActivationDataByKeyRequest

getActivationDataByKeyRequest

getActivationDataByKeyResponse

Description	This method returns activation information based on the input order key and service key.	
Input Structures	OrderKey MetaSolvServiceKey	
Response Structure	MetaSolvServiceActivationType	
MetaSolv Solution Path > Page		

Table 11: getActivationDataByKeyRequest

transferTaskRequest

transferTaskRequest

transferTaskResponse

Description	This method transfers tasks between work queues based on the input data.	
Input Structure	TransferTaskValueType	orderKey, taskNumber, currentWorkQueue, newWorkQueue
Response Structure	OrderKey	
MetaSolv Solution Path > Page		

Table 12: tranferTaskRequest

updateE911DataRequest

updateE911DataReqest

updateE911DataResponse

Description	This method updates E911 data based upon the input data.	
Input Structure	E911DataType	
Response Structure		
MetaSolv Solution Path > Page		

Table 13: updateE911DataRequest

getE911DataRequest

getE911DataRequest

getE911DataResponse

Description	This method returns E911 data, based upon the input data.	
Input Structure		
Response Structure	E911DataType	
MetaSolv Solution Path > Page		

Table 14: getE911DataRequest

updateEstimationCompletedDateRequest

updateEstimatedCompletionDateRequest

updateEstimatedCompletionDateResponse

Description	This method updates the estimated completion dates of tasks based on the input order and specified tasks associated with the order.	
Input Structure	UpdateEstimatedCompletionDateValueType	
Response Structure	status (String)	
MetaSolv Solution Path > Page		

Table 15: updateEstimationCompletedDateRequest

addTaskJeopardyRequest

addTaskJeopardyRequest

addTaskJeopardyResponse

Description	This method adds task jeopardy information based on the input task data.	
Input Structure	AddTaskJeopardyRequestValue Type	
Response Structure	status (String)	
MetaSolv Solution Path > Page		

Table 16: addTaskJeopardyRequest

getTaskDetailRequest

getTaskDetailRequest

getTaskDetailResponse

Description	This method returns task detail information based on the input data.	
Input Structure	GetTaskDetailRequestValueType	
Response Structure	GetTaskDetailResponseValue Type	
MetaSolv Solution Path > Page		

Table 17: getTaskDetailRequest

TaskJeopardyRequest

getTaskJeopardyRequest

getTaskJeopardyResponse

Description	This method returns task jeopardy information based on the input task data.	
Input Structure	GetTaskJeopardyRequestValue Type	
Response Structure	GetTaskJeopardyResponseValueType	
MetaSolv Solution Path > Page		

Table 18: TaskJeopardyRequest

getPSROrderByTN

getPSROrderByTNRequest

getPSROrderByTNResponse

Description	This method returns PSR order information based on the input telephone number.	
Input Structure	GetPSROrderByTNRequestValue Type	
Response Structure	MetaSolvOrderValueChoice	MetaSolvPSROrderValue or MetaSolvISROrderValue
MetaSolv Solution Path > Page		

Table 19: getPSROrderByTN

processSuppOrder

processSuppOrderRequest

processSuppOrderResponse

Description	This method processes a supplement order based on the input data.	
Input Structure	ProcessSuppOrderRequestValue Type	
Response Structure	message (String)	
MetaSolv Solution Path > Page		

Table 20: processSuppOrder

getCNAMDataRequest

getCNAMDataRequest

getCNAMDataResponse

Description	This method returns CNAM data based upon the input data.	
Input Structure		
Response Structure	CNAMDataType	
MetaSolv Solution Path > Page		

Table 21: getCNAMDataRequest

getLidbDataRequest

getLIDBDataRequest

getLIDBDataResponse

Description	This method returns LIDB data based on the input data.	
Input Structure		
Response Structure	LIDBDataType	
MetaSolv Solution Path > Page		

Table 22: getLidbDataRequest

updateCNAMDataRequest

updateCNAMDataRequest

updateCNAMEDataResponse

Description	This method updates CNAM data based on the input data.	
Input Structure	CNAMDataType	
Response Structure	status (String)	
MetaSolv Solution Path > Page		

Table 23: updateCNAMDataRequest

updateLidbDataRequest

updateLIDBDataRequest

updateLIDBDataResponse

Description	This method updates LIDB data based on the input data.	
Input Structure	LIDBDataType	
Response Structure	status (String)	
MetaSolv Solution Path > Page		

Table 24: updateLidbDataRequest

reopenTaskRequest

reopenTaskRequest

reopenTaskResponse

Description	This method reopens a task based on the input data.	
Input Structure	ReopenTaskValueType	
Response Structure	MetaSolvOrderValueChoice	MetaSolvPSROrderValue or MetaSolvISROrderValue
MetaSolv Solution Path > Page		

Table 25: reopenTaskRequest

createAttachmentRequest

createAttachmentRequest

createAttachmentResponse

Description	This method creates an xml document attachment that is associated with a PSR based on the input data.	
Input Structure	CreateAttacmentType	
Response Structure	message (String)	
MetaSolv Solution Path > Page		

Table 26: createAttachmentRequest

createOrderRelationshipRequest

createOrderRelationshipRequest

createOrderRelationshipResponse

Description	This method creates a parent/child relationship based on the two input orders.	
Input Structures	OrderKey (parent) OrderKey (child)	
Response Structure	message (String)	
MetaSolv Solution Path > Page	Service Request>Service Request Hierarchy	

Table 27: createOrderRelationshipRequest

Additional Information

This method only supports the order relationship type of parent/child. This is important to note because MetaSolv Solution supports several different relationship types, but this method only supports the relationship type of parent/child.

processBillingTelephoneNumber

billingTelephoneNumber

billingTelephoneNumberResponse

Description	This method receives a structure to be passed to process the number as Billing Telephone Number.	
Input Structures	documentNumber (long) servItemId(long) BtnFunction-Enum(enum defined in same xsd file, as a String with value of 0 or 1) nbrInvId(long)	
Response Structure	documentNumber (integer)	
MetaSolv Solution Path > Page	Order Management > Product Service Request	

Table 28: processBillingTelephoneNumber

Inventory Management API

The Inventory Management API is a collection of methods that provide the ability to import and maintain inventory in the MetaSolv Solution database from an outside source, without using the MetaSolv Solution GUI. Inventory items include Locations (PSR Service Locations, End User Locations, and Network Locations), Telephone Numbers, Physical Ports, and IP Addresses. The XmlMetaSolvInventoryManagementAPI.xsd defines the following methods:

createEntityByValueRequest

createEntityByValueRequest

createEntityByValueResponse

Description	This method creates a new entity based on the choice of input structure.	
Input Structure	CreateEntityValueChoice	CreatePSRServiceLocationValue, CreateEndUserLocationValue, CreateNetworkLocationValue, or CreateNumberInventoryValue
Response Structure	CreateEntityResponseChoice	CreatePSRServiceLocation Response, CreateEndUserLocationResponse, CreateNetworkLocationResponse, or CreateNumberInventoryResponse
MetaSolv Solution Path > Page		

Table 29: createEntityByValueRequest

getServiceRequestDLRsValue

createEntityByValueRequest

createEntityByValueResponse

Description		
Input Structure		
Response Structure		
MetaSolv Solution Path > Page		

Table 30: getServiceRequestDLRsValue

getEntityByKeyRequest

getEntityByValueRequest

getEntityByValueResponse

Description	This method returns various entity information based on the choice of input structure.	
Input Structure	GetEntityByKeyValueChoice	GetPSRServiceLocationByKey, GetEndUserLocationByKey, GetNetworkLocationByKey, GetDlrByKey
Response Structure	GetEntityByKeyResponseChoice	GetPSRServiceLocationResponse GetEndUserLocationResponse, GetNetworkLocationResponse, GetDlrByKeyResponse
MetaSolv Solution Path > Page		

Table 31: getEntityByKeyRequest

updateEntityByValueRequest

updateEntityByValueRequest

updateEntityByValueResponse

Description	This method updates an existing entity based on the choice of input structure.	
Input Structure	UpdateEntityValueChoice	UpdatePSRServiceLocationValue or UpdatePreAssignTelephone NumberValue
Response Structure	UpdateEntityResponseChoice	UpdatePSRServiceLocation Response or UpdatePreAssignTelephone NumberResponse
MetaSolv Solution Path > Page		

Table 32: updateEntityByValueRequest

queryInventoryManagementRequest

queryInventoryManagementRequest

queryInventoryManagementResponse

Description	This method returns various inventory management information based on the choice of input structure.	
Input Structure	MetaSolvInventoryQueryValue Choice	QueryAlarmEnrichmentValue, QueryEquipmentCapacityValue, QueryTelephoneNumber InventoryValue, QueryMSAGInventoryValue
Response Structure	MetaSolvInventoryQuery ResponseChoice	QueryAlarmEnrichmentResponse, QueryEquipmentCapacity Response, QueryTelephoneNumber InventoryResponse, QueryMSAGResponse
MetaSolv Solution Path > Page		

Table 33: queryInventoryManagementRequest

updateTNRequest

updateTNRequest

updateTNResponse

Description	This method updates an existing telephone number based on the input data.	
Input Structure	UpdateTNRequestValueType	
Response Structure	UpdateTNResponseValue (int)	
MetaSolv Solution Path > Page		

Table 34: updateTNRequest

tnRecall

processTNRecallRequest

processTNRecallResponse

Description	This method recalls a telephone number based on the input data.	
Input Structure	tn (String)	
Response Structure	ProcessTNRecallResponseValue Type	
MetaSolv Solution Path > Page		

Table 35: tnRecall

tnValidationRequest

processTNValidationRequest

processTNValidationResponse

Description	This method validates the TN.	
Input Structure	tn (String)	
Response Structure	ProcessTNValidationResponse Value (String)	
MetaSolv Solution Path > Page		

Table 36: tnValidationRequest

auditTrailRecording

Request

Response

Description		
Input Structure		
Response Structure		
MetaSolv Solution Path > Page		

Table 37: auditTrailRecording

getNetworkAreasByGeoAreaRequest

getNetworkAreasByGeoAreaRequest

getNetworkAreasByGeoAreaResponse

Description	This method returns Network Area information based on the input Geographical Area.	
Input Structure	GeoAreaCriteria	
Response Structure	NetworkArea	
MetaSolv Solution Path > Page		

Table 38: getNetworkAreasByGeoAreaRequest

getNetworkComponentsRequest

getNetworkComponentsRequest

getNetworkComponentsResponse

Description	This method returns Network Component information based on the input data.	
Input Structure	getNetworkComponentsRequest ValueType	
Response Structure	NetworkComponent	
MetaSolv Solution Path > Page		

Table 39: getNetworkComponentsRequest

getIpAddressesRequest

getIpAddressesRequest

getIpAddressesResponse

Description	This method returns ip address information based on the input data.	
Input Structure	IpAddressCriteria	
Response Structure	IpAddressesValue (sequence)	
MetaSolv Solution Path > Page		

Table 40: getIpAddressesRequest

createInventoryAssociationRequest

createInventoryAssociationRequest

createInventoryAssociationResponse

Description	This creates a relationship between two inventory items based on the input data.	
Input Structure	ImportInventoryAssociation	
Response Structure	status	
MetaSolv Solution Path > Page		

Table 41: createInventoryAssociationRequest

createNewInventoryItemRequest

createNewInventoryItemRequest

createNewInventoryItemResponse

Description	This method creates a new inventory item based on the input data.	
Input Structure	InventoryItem	
Response Structure	MetaSolvNumberInventoryKey	
MetaSolv Solution Path > Page		

Table 42: createNewInventoryItemRequest

queryNetworkLocation

queryNetworkLocationRequest

queryNetworkLocationResponse

Description	This method retrieves multiple network locations from the MSS database based on the input data.	
Input Structure	NetworkLocationQueryValue	
Response Structure	NetworkLocationResultValue	
MetaSolv Solution Path > Page		

Table 43: queryNetworkLocation

queryEndUserLocation

queryEndUserLocationRequest

queryEndUserLocationResponse

Description	This method retrieves multiple end user locations from the MSS database based on the input data.	
Input Structure	EndUserLocationQueryValue	
Response Structure	EndUserLocationResultValue	
MetaSolv Solution Path > Page		

Table 44: queryEndUserLocation

getLocationRequest

getLocationRequest

getLocationResponse

Description	This method retrieves a specific location from the MSS database based on the input data key.	
Input Structure	NetworkLocationKey	
Response Structure	LocationValue	
MetaSolv Solution Path > Page		

Table 45: getLocationRequest

deleteLocationRequest

deleteLocationByKey

deleteLocationResponse

Description	This method deletes a specific location from the MSS database based on the input data key.	
Input Structure	NetworkLocationKey	
Response Structure	NetworkLocationKey	
MetaSolv Solution Path > Page		

Table 46: deleteLocationRequest

updateLocationRequest

updateLocationRequest

updateLocationResponse

Description	This method updates a specific location in the MSS database based on the input data.	
Input Structure	LocationValue	
Response Structure	NetworkLocationKey	
MetaSolv Solution Path > Page		

Table 47: updateLocationRequest

createLocationRequest

createLocationRequest

createLocationResponse

Description	This method creates a new location in the MSS data base based on the input data.	
Input Structure	LocationValue	
Response Structure	NetworkLocationKey	
MetaSolv Solution Path > Page		

Table 48: createLocationRequest

getAvailablePhysicalPortsRequest

getAvailablePhysiclaPortsRequest

getAvailablePhysicalPortsResponse

Description	This method returns a sequence of available physical ports based on the input data.	
Input Structure	getAvailablePhysicalPortsRequest ValueType	
Response Structure	PhysicalPort (sequence)	
MetaSolv Solution Path > Page		

Table 49: getAvailablePhysicalPortsRequest

Additional Information

Note that this method is defined in a different control class than the rest of the methods defined in this section. While the InventoryManagementAPI.xsd defines the structures for this method, the InventoryManagement control class does not define the control. Rather, the control is defined in the NetworkResourceManagement control class. The NetworkResourceManagement will continue to grow in future releases, at which point a new xsd will be created.

Service Order Activation API

The Service Order Activation API is a collection of methods that provide the ability to activate services, previously placed on orders, in the MetaSolv Solution database from an outside source, without using the MetaSolv Solution GUI. The XmlMetaSolvInventoryManagementAPI.xsd defines the following methods:

createSOAMessageRequest

createSOAMessageRequest

createSOAMessageResponse

Description	This method creates a SOA message based on the input data.	
Input Structure	SOATransactionType, OrderKey	
Response Structure	SOATransactionKey	
MetaSolv Solution Path > Page		

Table 50: createSOAMessageRequest

getSoaTnsForOrderRequest

getSOATNsForOrderRequest

getSOATNsForOrderResponse

Description	This method returns SOA telephone numbers based on the input order.	
Input Structure	OrderKey	
Response Structure	SOATelephoneNumberType (sequence)	
MetaSolv Solution Path > Page		

Table 51: getSoaTnsForOrderRequest

getSoaDefaultsRequest

getSOADefaultsRequest

getSOADefaultsResponse

Description	This method returns the SOA defaults based on the input data.	
Input Structure	OrderKey, SOATelephoneNumberType	
Response Structure	SOADefaultsType	
MetaSolv Solution Path > Page		

Table 52: getSoaDefaultsRequest

getSoaInformationRequest

getSOAInformationRequest

getSOAInformationResponse

Description	This method returns SOA information based on the input data.	
Input Structure	OrderKey, SOATelephoneNumberType	
Response Structure	SOAInformationType	
MetaSolv Solution Path > Page		

Table 53: getSoaInformationRequest

getSoaMessageToSendRequest

getSOAMessagesToSendRequest

getSOAMessagesToSendResponse

Description	This method returns SOA messages to send based on the input data.	
Input Structure	OrderKey, checkGatewayEventReactivated (boolean)	
Response Structure	SOATransactionType (sequence)	
MetaSolv Solution Path > Page		

Table 54: getSoaMessagesToSendRequest

setTnSoaCompleteRequest

setTNSOACompleteRequest

setTNSOACompleteResponse

Description	This method sets SOA for the telephone number to complete.	
Input Structure	OrderKey, SOATelephoneNumberType	
Response Structure	successfulCompletion (boolean)	
MetaSolv Solution Path > Page		

Table 55: setTnSoaCompleteRequest