Information Builders

iWay

iWay Integration Tools Transformer User's Guide

Version 7.0.x and Higher

August 16, 2018

Contents

Preface

This document is written for system integrators who require data transformations and have the need to define and edit the structure of metadata. It describes how to use iWay Integration Tools (iIT) Transformer.

Note: This Release 7.0.x content is currently being updated to support iWay Release 8.0.x software. In the meantime, it can serve as a reference for your use of iWay Release 8. If you have any questions, please contact *Customer_Success@ibi.com*.

How This Manual Is Organized

This manual includes the following chapters:

	Chapter/Appendix	Contents
1	iWay Transformer Overview	Provides an overview of the major facilities, tools, and data formats supported by iWay Transformer.
2	Getting Started With iWay Transformer	Provides getting started information for iWay Transformer, including three tutorials.
3	iWay Transformer Concepts	Describes concepts related to iWay Transformer, such as transformation and mappings.
4	iWay Transformer Tasks	Describes the basic menus and options available in iWay Transformer.
5	iWay Transformer Tips and Tricks	Provides useful tips and tricks that are related to iWay Transformer.

Documentation Conventions

The following table describes the documentation conventions that are used in this manual.

Convention	Description
THIS TYPEFACE or this typeface	Denotes syntax that you must enter exactly as shown.
this typeface	Represents a placeholder (or variable), a cross-reference, or an important term. It may also indicate a button, menu item, or dialog box option that you can click or select.

Convention	Description
<u>underscore</u>	Indicates a default setting.
Key + Key	Indicates keys that you must press simultaneously.
{ }	Indicates two or three choices. Type one of them, not the braces.
|	Separates mutually exclusive choices in syntax. Type one of them, not the symbol.
...	Indicates that you can enter a parameter multiple times. Type only the parameter, not the ellipsis (...).
. . .	Indicates that there are (or could be) intervening or additional commands.

Related Publications

Visit our Technical Documentation Library at *http://documentation.informationbuilders.com*. You can also contact the Publications Order Department at (800) 969-4636.

Customer Support

Do you have any questions about this product?

Join the Focal Point community. Focal Point is our online developer center and more than a message board. It is an interactive network of more than 3,000 developers from almost every profession and industry, collaborating on solutions and sharing tips and techniques. Access Focal Point at *http://forums.informationbuilders.com/eve/forums*.

You can also access support services electronically, 24 hours a day, with InfoResponse Online. InfoResponse Online is accessible through our website, *http://www.informationbuilders.com*. It connects you to the tracking system and known-problem database at the Information Builders support center. Registered users can open, update, and view the status of cases in the tracking system and read descriptions of reported software issues. New users can register immediately for this service. The technical support section of *http://www.informationbuilders.com* also provides usage techniques, diagnostic tips, and answers to frequently asked questions.

Call Information Builders Customer Support Services (CSS) at (800) 736-6130 or (212) 736-6130. Customer Support Consultants are available Monday through Friday between 8:00 a.m. and 8:00 p.m. EST to address all your questions. Information Builders consultants can also give you general guidance regarding product capabilities and documentation. Please be ready to provide your six-digit site code number (*xxxx.xx*) when you call.

To learn about the full range of available support services, ask your Information Builders representative about InfoResponse Online, or call (800) 969-INFO.

Help Us to Serve You Better

To help our consultants answer your questions effectively, be prepared to provide specifications and sample files and to answer questions about errors and problems.

The following tables list the environment information our consultants require.

Platform	
Operating System	
OS Version	
JVM Vendor	
JVM Version	

The following table lists the deployment information our consultants require.

Adapter Deployment	For example, JCA, Business Services Provider, iWay Service Manager
Container	For example, WebSphere
Version	
Enterprise Information System (EIS) - if any	
EIS Release Level	
EIS Service Pack	
EIS Platform	

The following table lists iWay-related information needed by our consultants.

iWay Adapter	
iWay Release Level	
iWay Patch	

The following table lists additional questions to help us serve you better.

Request/Question	Error/Problem Details or Information
Did the problem arise through a service or event?	
Provide usage scenarios or summarize the application that produces the problem.	
When did the problem start?	
Can you reproduce this problem consistently?	
Describe the problem.	
Describe the steps to reproduce the problem.	
Specify the error message(s).	
Any change in the application environment: software configuration, EIS/database configuration, application, and so forth?	
Under what circumstance does the problem *not* occur?	

The following is a list of error/problem files that might be applicable.

- Input documents (XML instance, XML schema, non-XML documents)
- Transformation files
- Error screen shots
- Error output files
- Trace files
- Service Manager package to reproduce problem
- Custom functions and agents in use
- Diagnostic Zip
- Transaction log

For information on tracing, see the *iWay Service Manager User's Guide*.

User Feedback

In an effort to produce effective documentation, the Technical Content Management staff welcomes your opinions regarding this document. Please use the Reader Comments form at the end of this document to communicate your feedback to us or to suggest changes that will support improvements to our documentation. You can also contact us through our website, *http://documentation.informationbuilders.com/connections.asp*.

Thank you, in advance, for your comments.

Information Builders Consulting and Training

Interested in training? Information Builders Education Department offers a wide variety of training courses for this and other Information Builders products.

For information on course descriptions, locations, and dates, or to register for classes, visit our website (*http://education.informationbuilders.com*) or call (800) 969-INFO to speak to an Education Representative.

Chapter 1

iWay Transformer Overview

This section provides an overview of the major facilities, tools, and data formats supported by iWay Transformer.

In this chapter:

About iWay Transformer

iWay Transformer offers integrated data transformation services within iWay Service Manager. It provides support for creating, modifying, and publishing Transform components.

A graphical tool called Mapping Builder provides design services for specifying and defining transformations. It establishes mapping rules and relationships within your output document, based on the input structure if required. You can test the Transform by executing it with an actual input document to see the results. When you confirm that the Transform is stable, you can publish it, making it available to iWay Service Manager.

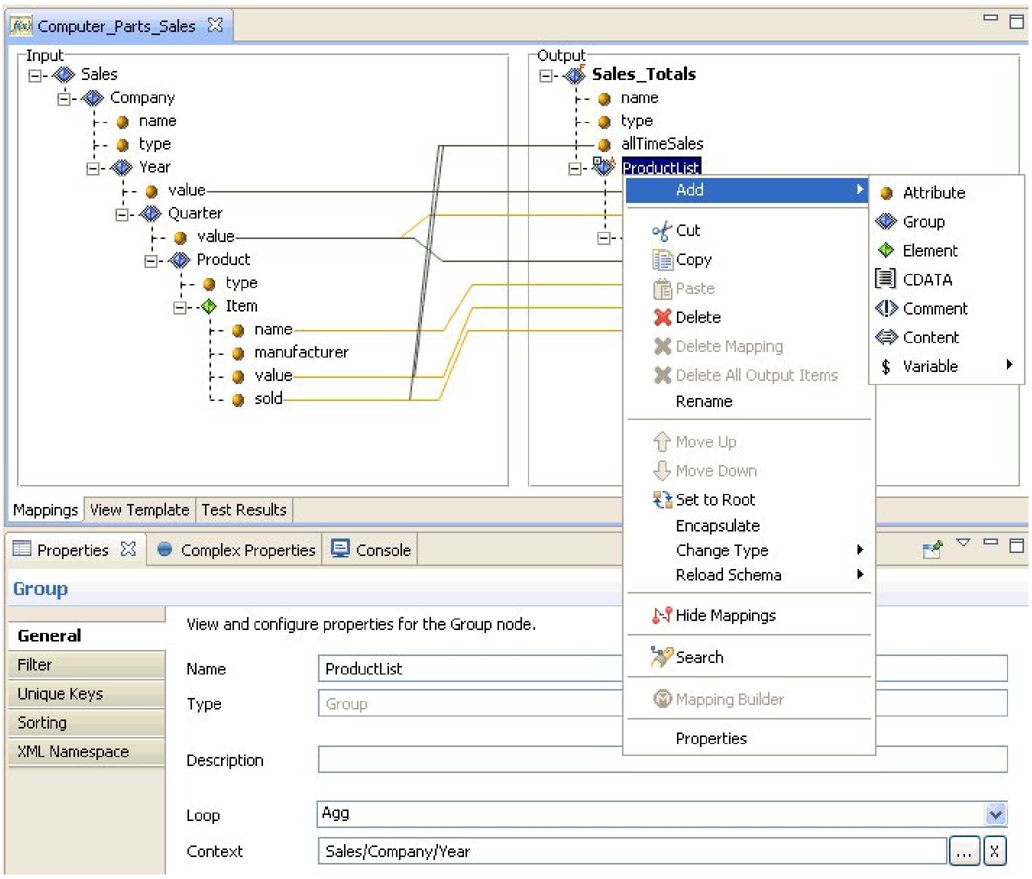
The main facilities within iWay Transformer are Mapping Builder, Dictionary Builder, Template Viewer, and Test Results.

Mapping Builder

Mapping Builder is a graphical editor that enables you to manage the rules or mappings of your transformations. It is identified by the Mappings tab and allows you to navigate through the input and output side of a transformation, and design mapping rules for it using an extensive set of customizable tools.

You can quickly and easily accomplish mappings between individual structure nodes of the input (incoming) document and the output (outgoing) document using drag and drop operations. The Mapping Builder interface supports one-to-one, one-to-many, and many-to-one mapping relationships, referring to the fields in the input and output documents, respectively. It also supports a set of functions that enable various forms of data manipulation.

The following image shows the Mapping Builder.



View and configure properties for the Group node.

Name: ProductList
Type: Group
Description:
Loop: Agg
Context: Sales/Company/Year

For more information, see *Working With the Mapping Builder* on page 224.

Dictionary Builder

Dictionary Builder is a graphical editor that is integrated with iWay Transformer. Dictionary Builder provides an interface for creating and modifying e-business metadata currently supported by the iWay framework as a data dictionary component of a Transform project and by stand-alone Ebixes.

A dictionary is a metadata representation that describes the structure of a transaction or a document that is used in EDI, SWIFT, or proprietary formatted message.

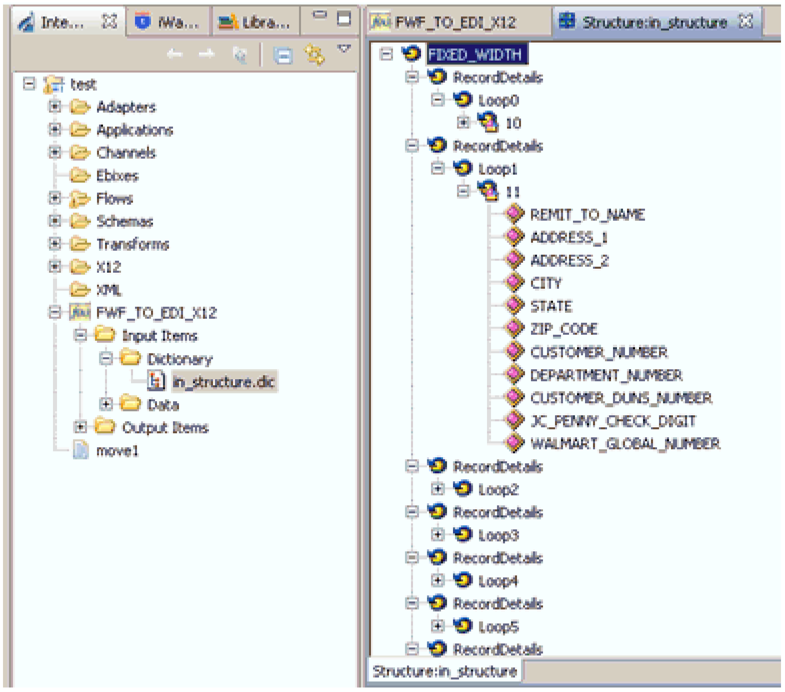
Dictionary Builder is an efficient interface that you can use in EDI, SWIFT, or proprietary format message solutions for composing, editing, and distributing e-business metadata in the form of Ebixes or Transform data dictionary components. Dictionary Builder in iWay Transformer contains the following management facilities:

- ❑ Data Dictionaries (Metadata Management Facility)
- ❑ Ebixes or Ebix Entry Projects (Ebix Management Facility)

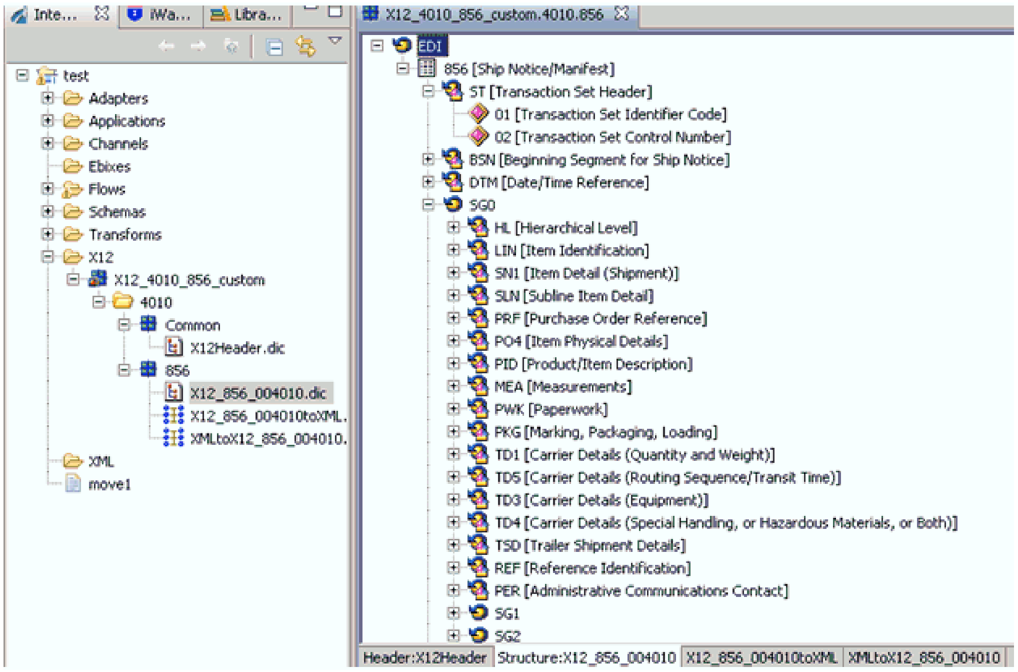
You can use the Dictionary Builder interface to browse, in a structured way, all the components of e-business data dictionaries. You can access the Dictionary Builder interface in iWay Transformer, as follows.

Procedure: How to Access Dictionary Builder

1. Open a Transform project (.gxp) that uses a dictionary of supported data formats for its input or output, as shown in the following image.



2. Open an Ebix Entry project (.ebx), as shown in the following image.

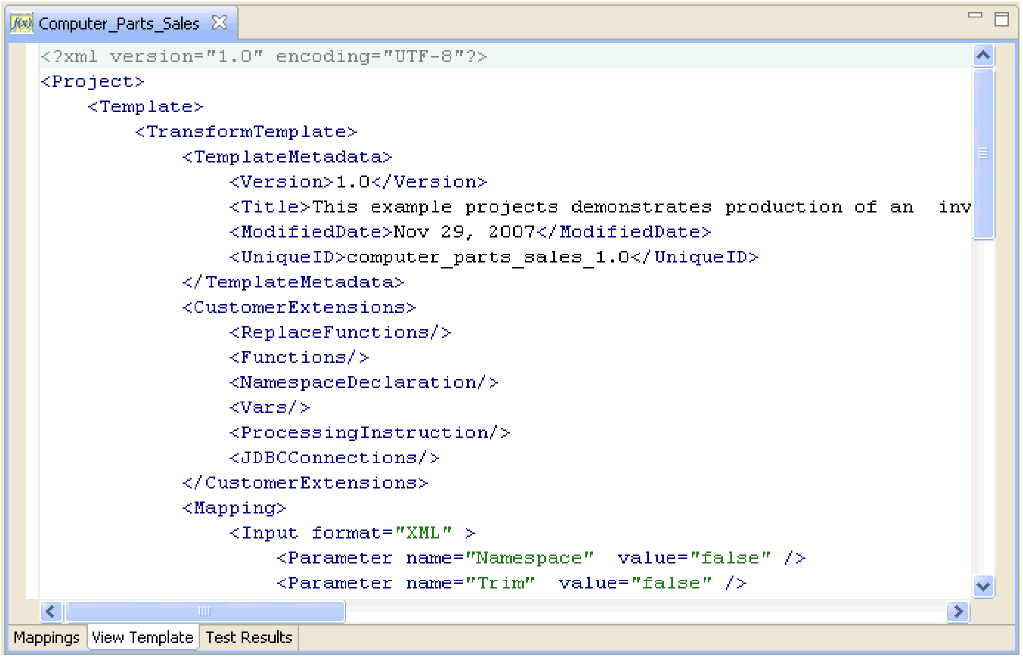


```
test
  Adapters
  Applications
  Channels
  Ebixes
  Flows
  Schemas
  Transforms
  X12
    X12_4010_856_custom
      4010
        Common
          X12Header.dic
        856
          X12_856_004010.dic
          X12_856_004010toXML.
          XMLtoX12_856_004010.
  XML
  move1
```

```
EDI
  856 [Ship Notice/Manifest]
    ST [Transaction Set Header]
      01 [Transaction Set Identifier Code]
      02 [Transaction Set Control Number]
    BSN [Beginning Segment for Ship Notice]
    DTM [Date/Time Reference]
    SG0
      HL [Hierarchical Level]
      LIN [Item Identification]
      SN1 [Item Detail (Shipment)]
      SLN [Subline Item Detail]
      PRF [Purchase Order Reference]
      PO4 [Item Physical Details]
      PID [Product/Item Description]
      MEA [Measurements]
      PWK [Paperwork]
      PKG [Marking, Packaging, Loading]
      TD1 [Carrier Details (Quantity and Weight)]
      TD5 [Carrier Details (Routing Sequence/Transit Time)]
      TD3 [Carrier Details (Equipment)]
      TD4 [Carrier Details (Special Handling, or Hazardous Materials, or Both)]
      TSD [Trailer Shipment Details]
      REF [Reference Identification]
      PER [Administrative Communications Contact]
      SG1
      SG2
```

Template Viewer

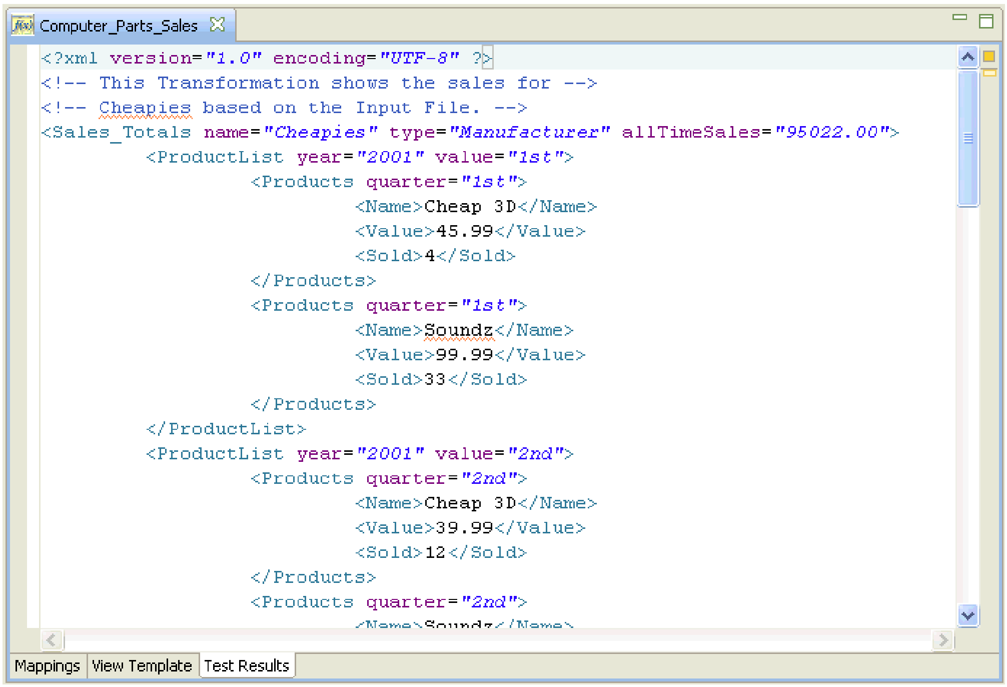
Template Viewer provides a read-only view of a Transform template, as an XML representation of a Transform component. It is identified by the View Template tab, as shown in the following image.



```
<?xml version="1.0" encoding="UTF-8"?>
<Project>
    <Template>
        <TransformTemplate>
            <TemplateMetadata>
                <Version>1.0</Version>
                <Title>This example projects demonstrates production of an  inv
                <ModifiedDate>Nov 29, 2007</ModifiedDate>
                <UniqueID>computer_parts_sales_1.0</UniqueID>
            </TemplateMetadata>
            <CustomerExtensions>
                <ReplaceFunctions/>
                <Functions/>
                <NamespaceDeclaration/>
                <Vars/>
                <ProcessingInstruction/>
                <JDBCConnections/>
            </CustomerExtensions>
            <Mapping>
                <Input format="XML" >
                    <Parameter name="Namespace"  value="false" />
                    <Parameter name="Trim"  value="false" />
```

Test Results

Test Results displays the output document that is generated by a transformation. It is identified by the Test Results tab, as shown in the following image.



```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This Transformation shows the sales for -->
<!-- Cheapies based on the Input File. -->
<Sales_Totals name="Cheapies" type="Manufacturer" allTimeSales="95022.00">
         <ProductList year="2001" value="1st">
                  <Products quarter="1st">
                           <Name>Cheap 3D</Name>
                           <Value>45.99</Value>
                           <Sold>4</Sold>
                  </Products>
                  <Products quarter="1st">
                           <Name>Soundz</Name>
                           <Value>99.99</Value>
                           <Sold>33</Sold>
                  </Products>
         </ProductList>
         <ProductList year="2001" value="2nd">
                  <Products quarter="2nd">
                           <Name>Cheap 3D</Name>
                           <Value>39.99</Value>
                           <Sold>12</Sold>
                  </Products>
                  <Products quarter="2nd">
                           <Name>Soundz</Name>
```

Chapter 2

Getting Started With iWay Transformer

This section provides getting started information for iWay Transformer, including three tutorials.

In this chapter:

iWay Transformer Basic Tutorial

This topic provides a basic tutorial for iWay Transformer.

Transformer Workbench Basics

The Transformer workbench is an Eclipse-based, end-to-end, integrated development environment (IDE) for data transformation design, testing, and management within iWay Service Manager (iSM). It uses a collection of visual layout, build, and debugging tools. Some of the basic functionality of the Transformer IDE comes from Eclipse. For example, managing, searching, and navigating resources are inherited from the core features.

About the Transformer Workbench

The Transformer workbench is a full-featured environment that is tailored to assist you in developing data transformations for iSM.

Unless you are using other Eclipse plug-ins, you do not need to be concerned with the underlying Eclipse framework.

Workbench. The workbench refers to the iWay Transformer development environment, which contains three primary facilities:

- ❑ **Perspective.** A perspective is a group of views and editors in the workbench. Essentially, it is a special work environment that helps you accomplish a specific type of task. The Transformer workbench is located within the Integration Explorer perspective of iWay Integration Tools (iIT).
- ❑ **Editor.** An editor allows you to edit various file types. iWay Transformer contains editors for creating transform (.gxp) and dictionary (structure) files.

- **View.** A view typically supports an editor. For example, when you are editing a Transform component, Output, Input, Mapping Builder, Mapping Properties, and Template views are also displayed in the Transform editor.

You use all three facilities in various combinations and at various points during the transformation development process. The workbench is the container of all the tools that are used to develop transformations, which are also called Transform components in iSM.

About iWay Transformer Editors

iWay Transformer contains editors that allow you to edit Transform project (.gxp) files, as well as its metadata, such as structure (dictionary) files. Editors are associated with resource types. As resources are opened in the workbench, the appropriate editor is opened:

- **Transform Editor.** The Transform editor is used to edit transformations. It has two modes: Mappings and Template. You use Mappings mode to visually design and structure your transformation. Template mode allows you to view an XML representation of your transformation. The two modes are synchronized, and changes in one mode are immediately reflected in the other.

- **Dictionary Builder.** The Dictionary Builder is used to edit dictionary (structure) files or ebix metadata for e-business and proprietary data formats (for example, FWF, CDF, and X12).

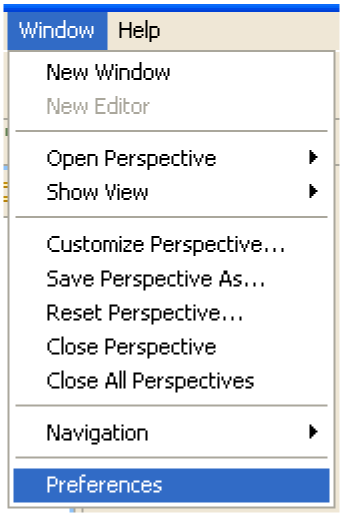
Setting iWay Transformer Preferences

When you start iWay Transformer for the first time, the iWay home directory must be set. We also recommend that you verify the iWay Transformer preferences that are in use. For more information on how to set the iWay home directory, see the *iWay Integration Tools Suite User's Guide*.

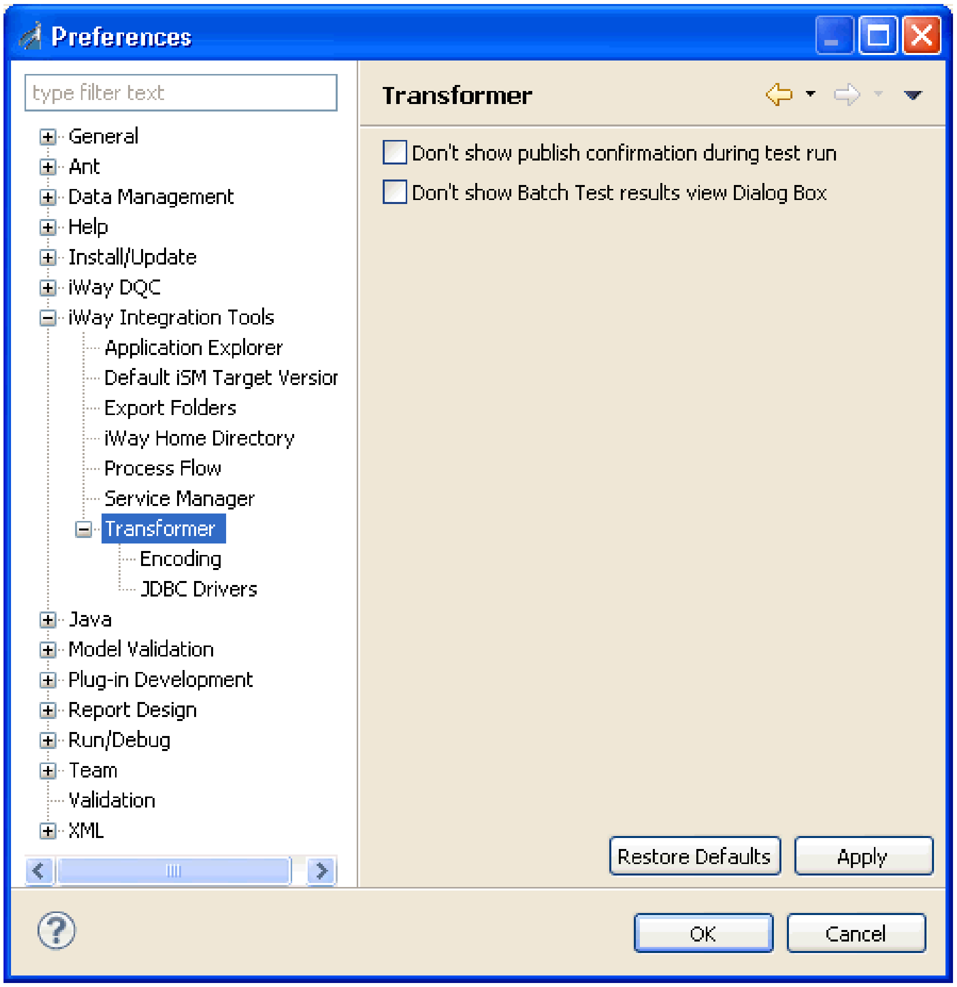
Procedure: How to Verify iWay Transformer Preferences

To verify iWay Transformer preferences:

1. From the menu bar, click *Window*, and select *Preferences*, as shown in the following image.



The Preferences dialog box opens, as shown in the next image.



Preferences

type filter text

- General
- Ant
- Data Management
- Help
- Install/Update
- iWay DQC
- iWay Integration Tools
  - Application Explorer
  - Default iSM Target Versior
  - Export Folders
  - iWay Home Directory
  - Process Flow
  - Service Manager
  - Transformer
    - Encoding
    - JDBC Drivers
- Java
- Model Validation
- Plug-in Development
- Report Design
- Run/Debug
- Team
- Validation
- XML

Transformer

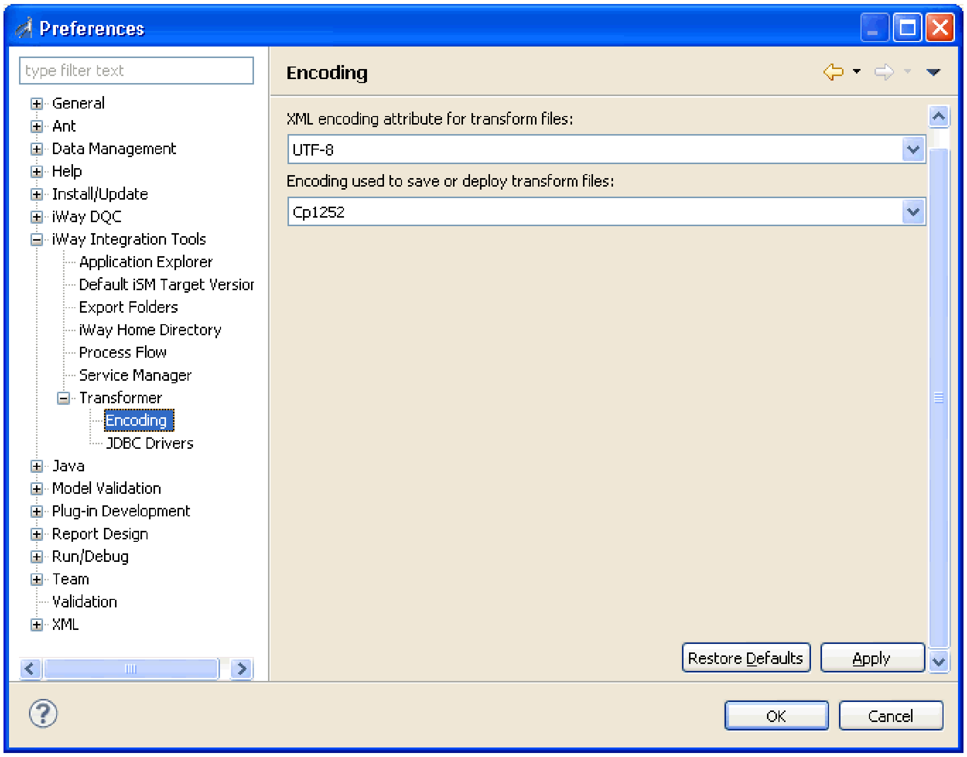
☐ Don't show publish confirmation during test run

☐ Don't show Batch Test results view Dialog Box

2. Expand *Transformer*.

   The Encoding and JDBC Driver categories are listed.

The following image shows the Encoding preference options that are currently set for iWay Transformer.



Encoding

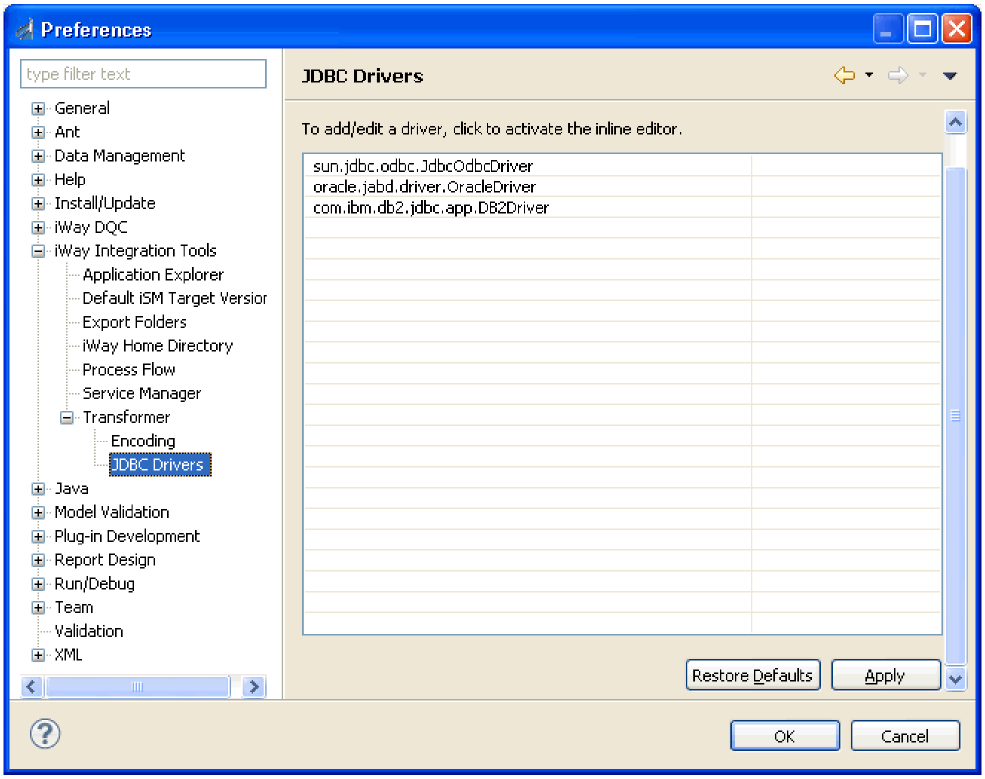
XML encoding attribute for transform files: UTF-8

Encoding used to save or deploy transform files: Cp1252

The Encoding category provides the option of customizing project and template encoding. The following options are available:

- Character encoding. You can select the type of encoding to use for the characters in a file. By default, the character encoding in iWay Transformer is set to UTF-8.
- File encoding. You can select the type of file encoding to use when saving or deploying project and template files to a system that uses a different language format. By default, the file encoding is set to the same value that is used by your operating system.

The following image shows the JDBC drivers that are currently used by iWay Transformer.



JDBC Drivers

To add/edit a driver, click to activate the inline editor.

```
sun.jdbc.odbc.JdbcOdbcDriver
oracle.jabd.driver.OracleDriver
com.ibm.db2.jdbc.app.DB2Driver
```

The JDBC Drivers category allows you to specify JDBC driver configurations that can be shared between applications and used by the @JDBCLOOKUP function and other database-related Transform functions.

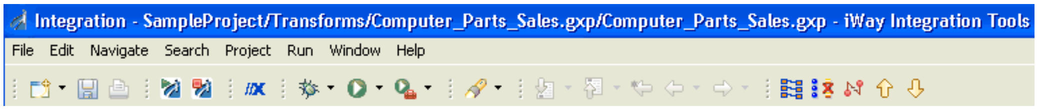
3. Verify or modify the iWay Transformer preferences according to your requirements, and click *OK*.

Transformer Workbench Toolbar and Shortcuts

This topic provides an overview of the Transformer workbench toolbar and shortcuts.

Navigating the Transformer Workbench Toolbar

When you start iWay Transformer, the main toolbar is displayed at the top of the window, as shown in the following image. Some of the toolbar buttons are disabled until you open or create a Transform component.


Integration - SampleProject/Transforms/Computer_Parts_Sales.gxp/Computer_Parts_Sales.gxp - iWay Integration Tools
File Edit Navigate Search Project Run Window Help

The following table includes an image of each toolbar button that is related to the Transformer workbench and describes its function.

Button	Function
🗋	Creates a new Transform component.
💾	Saves a Transform component.
🖶	Prints Transform component test results.
▶	Starts iWay Service Manager.
■	Stops iWay Service Manager.
//X	Launches the XPath Builder.
🐞	Debugs the Transform component.
▶	Runs the Transform component.
▶	Configures external tools to run the Transform component.
🔦	Searches across Transform component mappings and nodes.
▦	Maps the input structure as a root of the output structure. A confirmation dialog box is displayed when you select this option. This option is available only in the Mapping Builder.

Button	Function
	Optimizes mappings by removing all unmapped output nodes including groups with unmapped children. This option is available only in the Mapping Builder.
	Toggles between showing and hiding the mapping lines between the input and output nodes. The mappings signify the relationships between the input and output nodes, where the particular input value is used to construct the value of the output node.
⇧	Moves the selected node up the output structure tree, under the same parent node.
⇩	Moves the selected node down the output structure tree, under the same parent node.

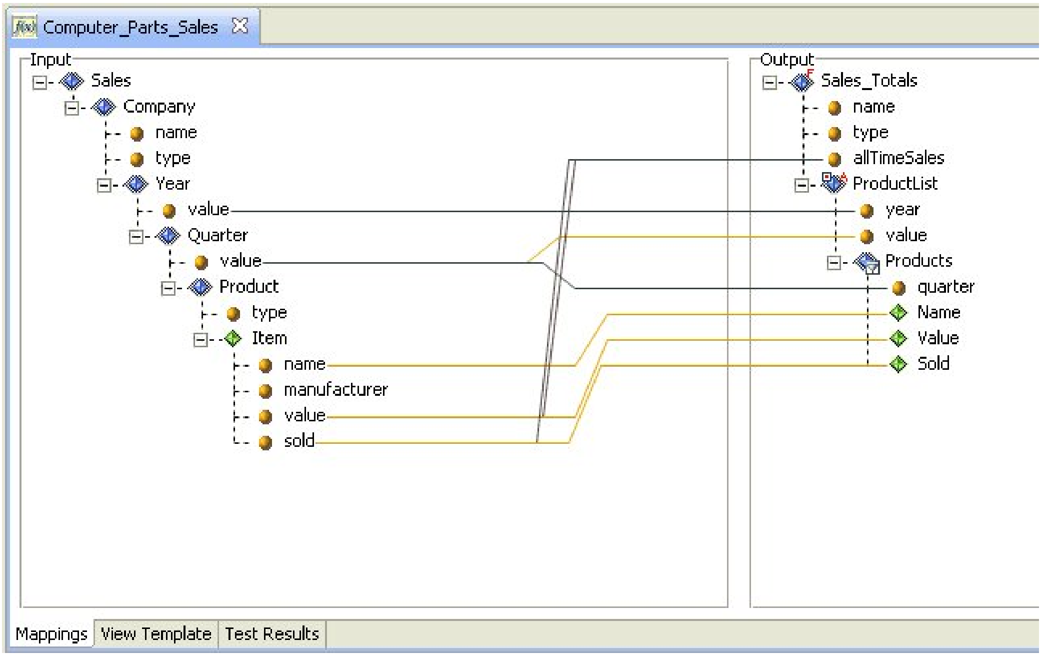
Navigating the Mapping Builder

The Mapping Builder uses mapping types to specify how records and fields of one document structure relate to another.

Mappings Tab

The Mappings tab allows you to design mapping rules for Transform components. You can quickly and easily accomplish mappings between individual nodes contained in structures of incoming and outgoing data, using drag and drop operations. The Mapping Builder interface supports one-to-one, one-to-many, and many-to-one mapping relationships, referring to the fields in the input and output documents, respectively.

The following image shows sample input and output structures on the Mappings tab.

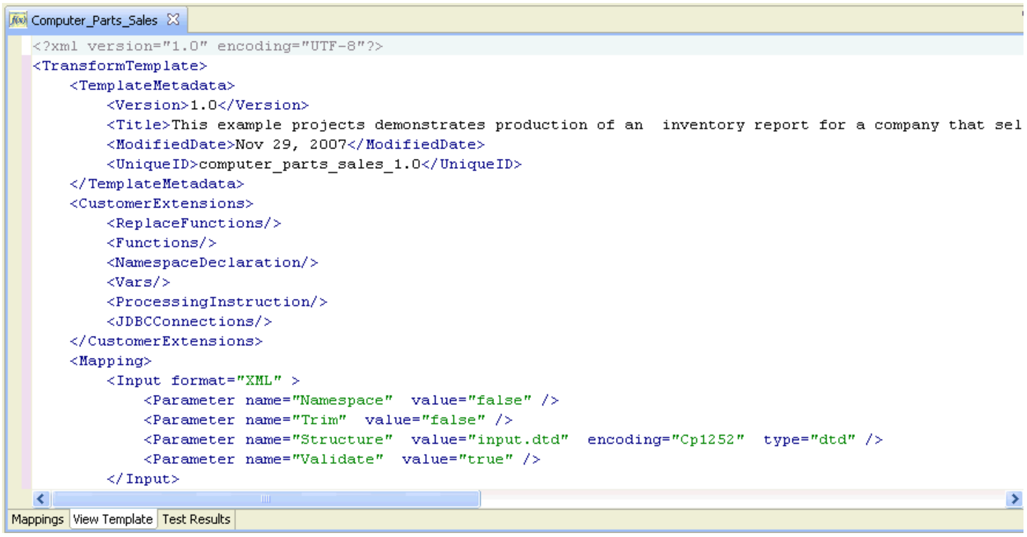


For more information on how to define mappings for Transform components using the Mappings tab, see *Working With the Mapping Builder* on page 224.

View Template Tab

The View Template tab provides a read-only view of a Transform template, as an XML representation, or a serialized view of a Transform component.

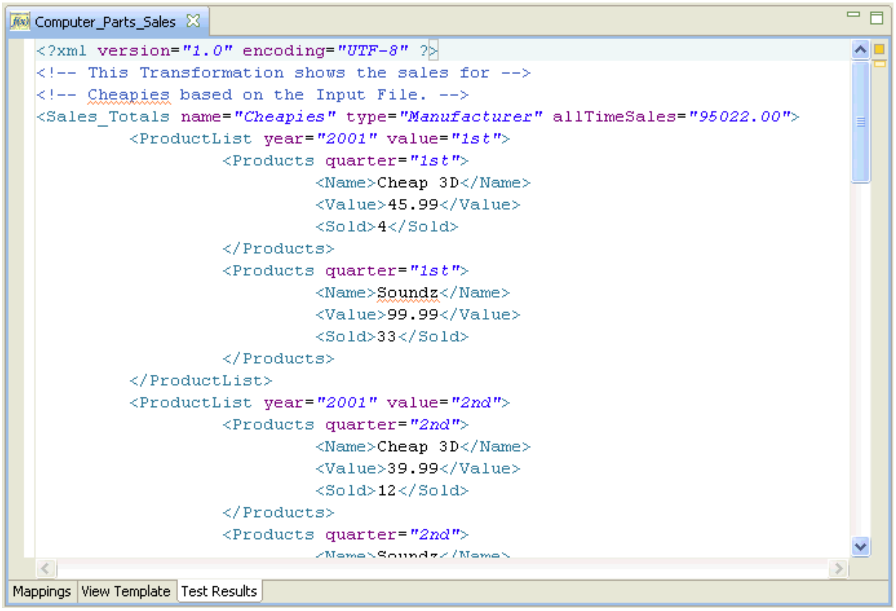
The following image shows a read-only view of a Transform template on the View Template tab.



```
<?xml version="1.0" encoding="UTF-8"?>
<TransformTemplate>
    <TemplateMetadata>
        <Version>1.0</Version>
        <Title>This example projects demonstrates production of an  inventory report for a company that sel
        <ModifiedDate>Nov 29, 2007</ModifiedDate>
        <UniqueID>computer_parts_sales_1.0</UniqueID>
    </TemplateMetadata>
    <CustomerExtensions>
        <ReplaceFunctions/>
        <Functions/>
        <NamespaceDeclaration/>
        <Vars/>
        <ProcessingInstruction/>
        <JDBCConnections/>
    </CustomerExtensions>
    <Mapping>
        <Input format="XML" >
            <Parameter name="Namespace"  value="false" />
            <Parameter name="Trim"  value="false" />
            <Parameter name="Structure"  value="input.dtd"  encoding="Cp1252"  type="dtd" />
            <Parameter name="Validate"  value="true" />
        </Input>
```

Test Results Tab

The Test Results tab displays the output document that is generated by your Transform component, as shown in the following image.



```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This Transformation shows the sales for -->
<!-- Cheapies based on the Input File. -->
<Sales_Totals name="Cheapies" type="Manufacturer" allTimeSales="95022.00">
        <ProductList year="2001" value="1st">
                <Products quarter="1st">
                        <Name>Cheap 3D</Name>
                        <Value>45.99</Value>
                        <Sold>4</Sold>
                </Products>
                <Products quarter="1st">
                        <Name>Soundz</Name>
                        <Value>99.99</Value>
                        <Sold>33</Sold>
                </Products>
        </ProductList>
        <ProductList year="2001" value="2nd">
                <Products quarter="2nd">
                        <Name>Cheap 3D</Name>
                        <Value>39.99</Value>
                        <Sold>12</Sold>
                </Products>
                <Products quarter="2nd">
                        <Name>Soundz</Name>
```

Using Keyboard Shortcuts

This following table lists keyboard shortcuts for commonly used tasks and functions.

Task or Function	Keyboard Shortcut
New	Alt+Shift+N
Close	Ctrl+W
Close All	Ctrl+Shift+W
Save	Ctrl+S
Save All	Ctrl+Shift+S
Undo	Ctrl+Z
Redo	Ctrl+Y
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Delete	Delete
Select All	Ctrl+A
Find and Replace	Ctrl+F
Search	Ctrl+H
Run	Ctrl+F11
Debug	F11
Help	F1

Preparing iWay Integration Tools Suite

For more information on installing, configuring, and starting iWay Integration Tools (iIT) Suite, see the *iWay Integration Tools Suite User's Guide*.

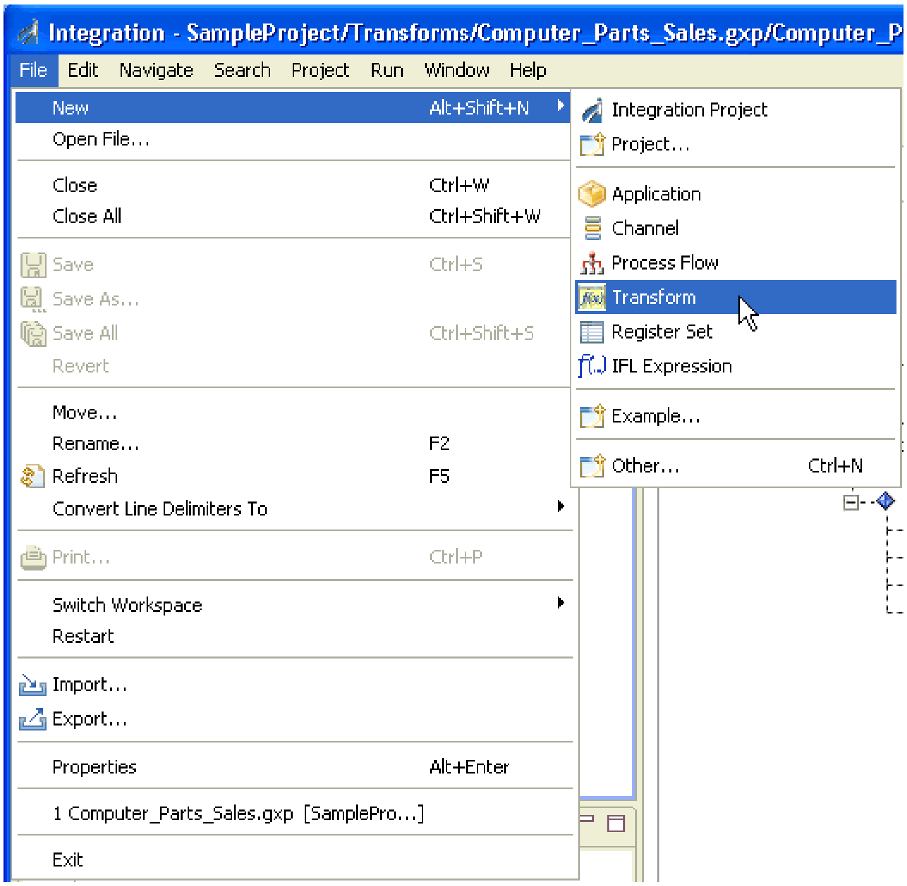
Creating a Transform Component

This topic describes how to create a Transform component using the File menu, Integration Project context menu, or New button.

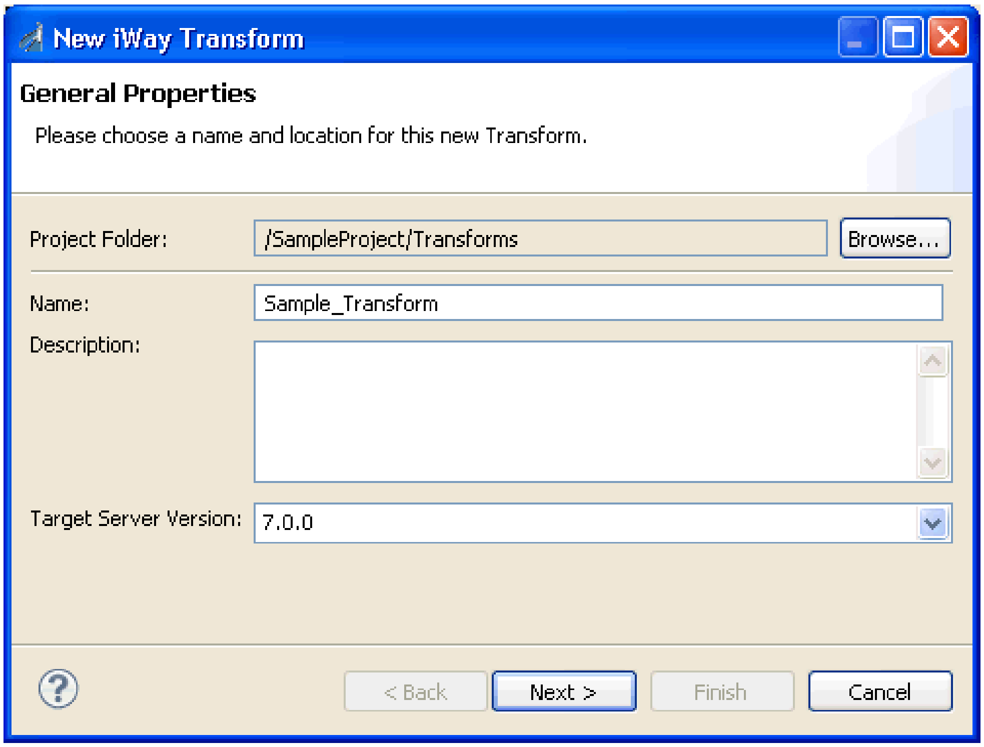
Procedure: How to Create a Transform Component Using the File Menu

To create a Transform component using the File menu:

Click *File*, select *New*, and then click *Transform* from the context menu, as shown in the following image.



The New iWay Transform wizard opens, as shown in the next image.



General Properties

Please choose a name and location for this new Transform.

Project Folder: /SampleProject/Transforms

Name: Sample_Transform

Description:

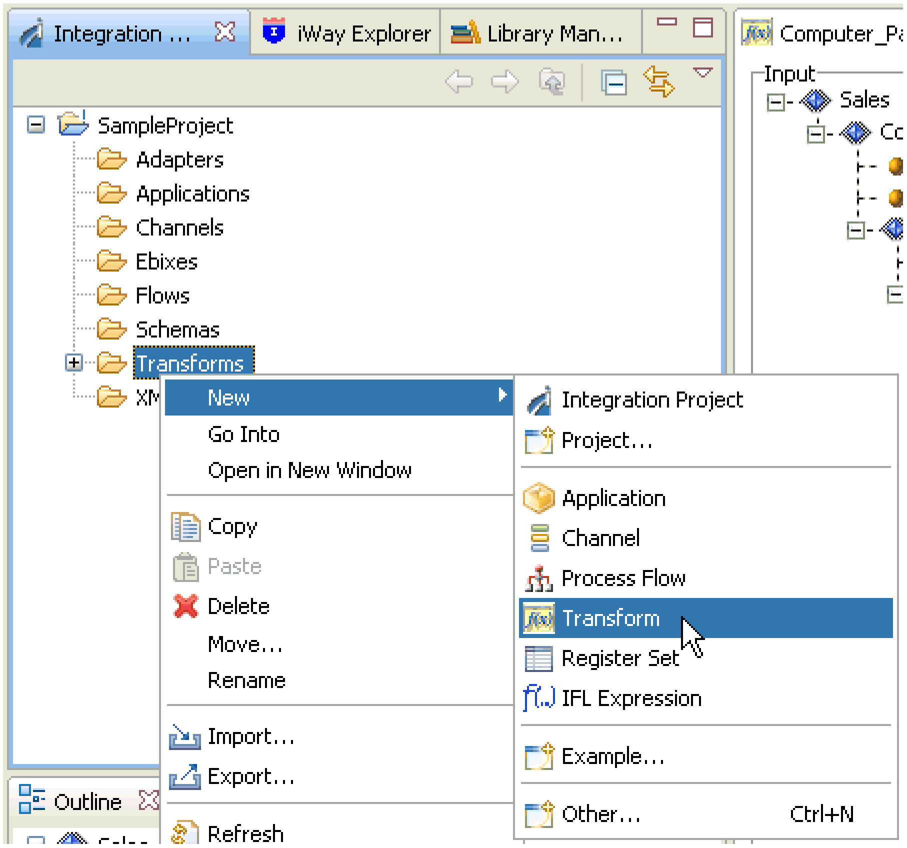
Target Server Version: 7.0.0

For more information on creating a Transform component using the New iWay Transform wizard, see *Configuring a New Transform Component* on page 62.

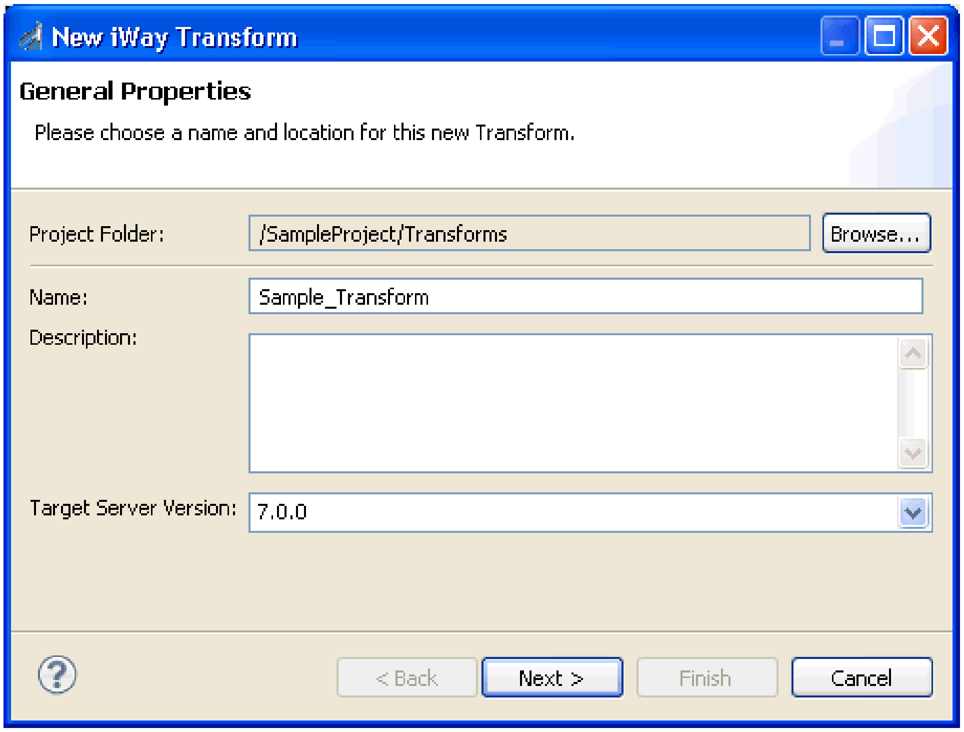
Procedure: How to Create a Transform Component Using the Integration Project Context Menu

To create a Transform component using the Integration Project context menu:

Right-click the *Transforms* folder in your Integration Project, select *New*, and then click *Transform*, as shown in the following image.



The New iWay Transform wizard opens, as shown in the next image.



Please choose a name and location for this new Transform.

Project Folder: /SampleProject/Transforms

Name: Sample_Transform

Description:

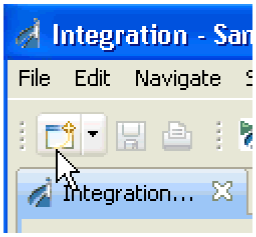
Target Server Version: 7.0.0

For more information on creating a Transform component using the New iWay Transform wizard, see *Configuring a New Transform Component* on page 62.

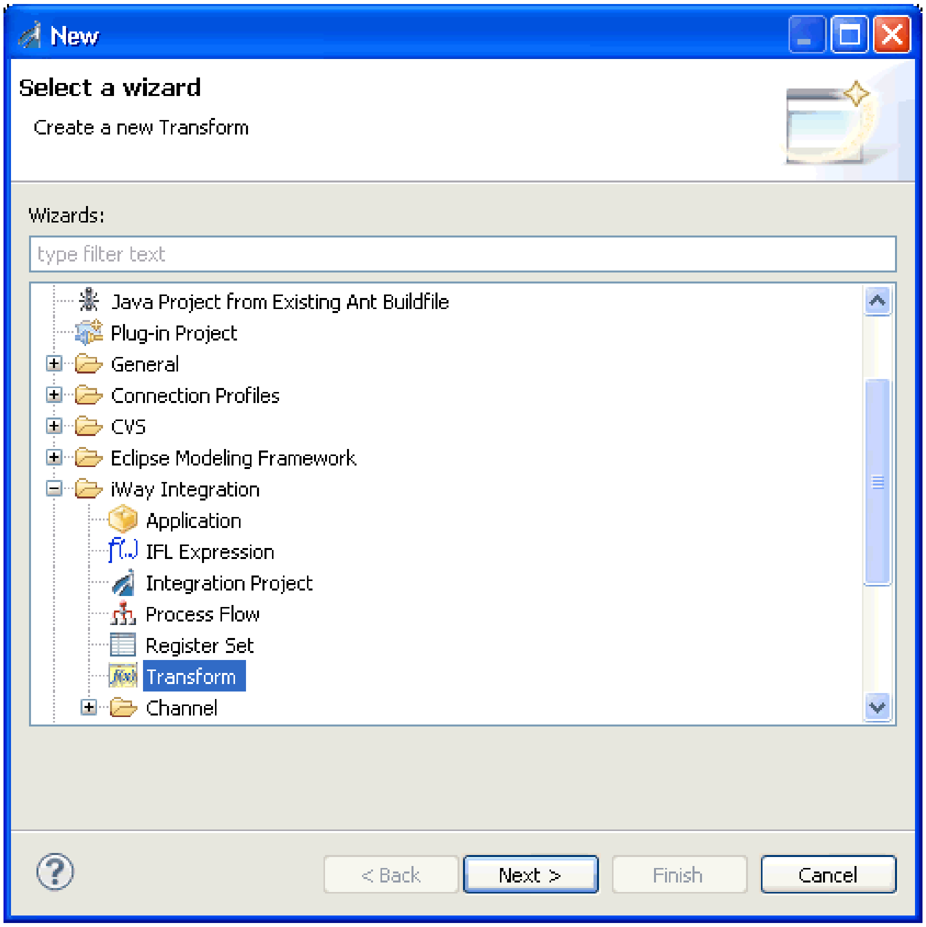
Procedure: How to Create a Transform Component Using the New Button

To create a Transform component using the New button:

1. Click the *New* button in the toolbar, as shown in the following image.

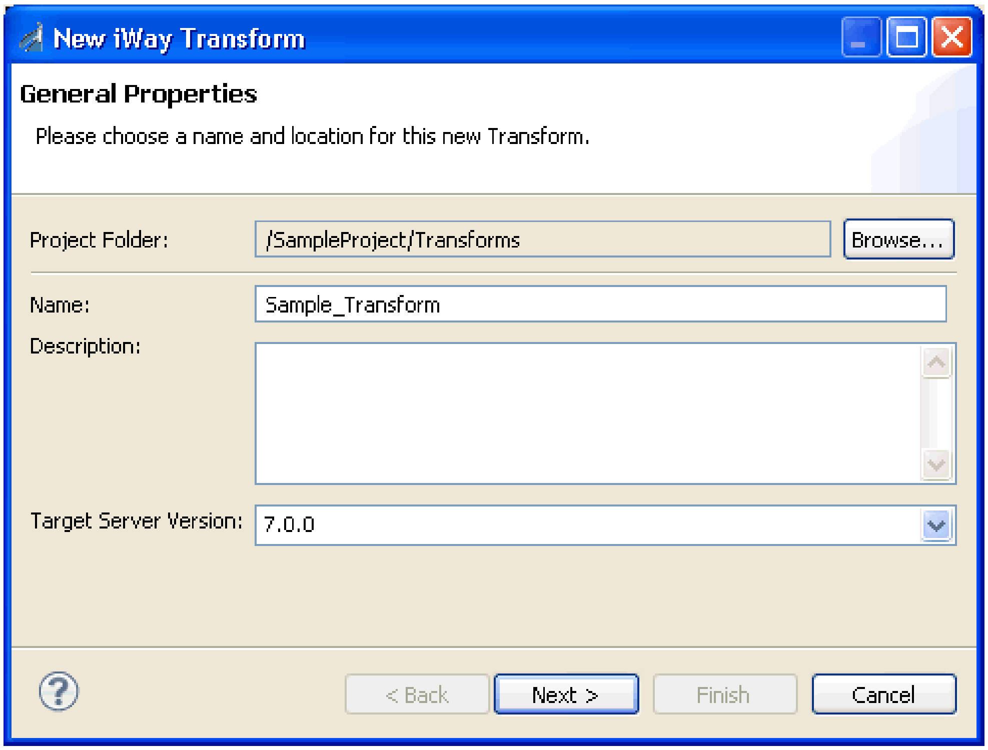


The New dialog box opens, as shown in the following image.



2. Expand *iWay Integration*, and then select *Transform*.

The New iWay Transform wizard opens, as shown in the next image.

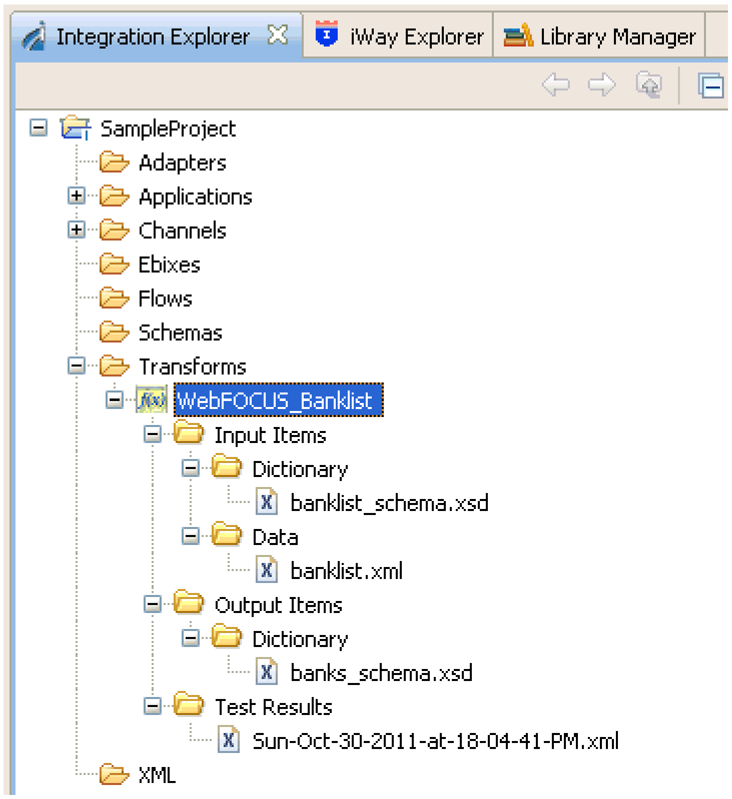

New iWay Transform
General Properties
Please choose a name and location for this new Transform.
Project Folder: /SampleProject/Transforms
Browse...
Name: Sample_Transform
Description:
Target Server Version: 7.0.0
< Back
Next >
Finish
Cancel

You can skip the New iWay Transform wizard steps by clicking the *Finish* button once it is enabled. You can then manually alter the Transform project properties at your convenience.

For more information on creating a Transform component using the New iWay Transform wizard, see *Configuring a New Transform Component* on page 62.

Navigating Resources

You can navigate Transform project resources in the Integration Explorer tab, as shown in the following image.



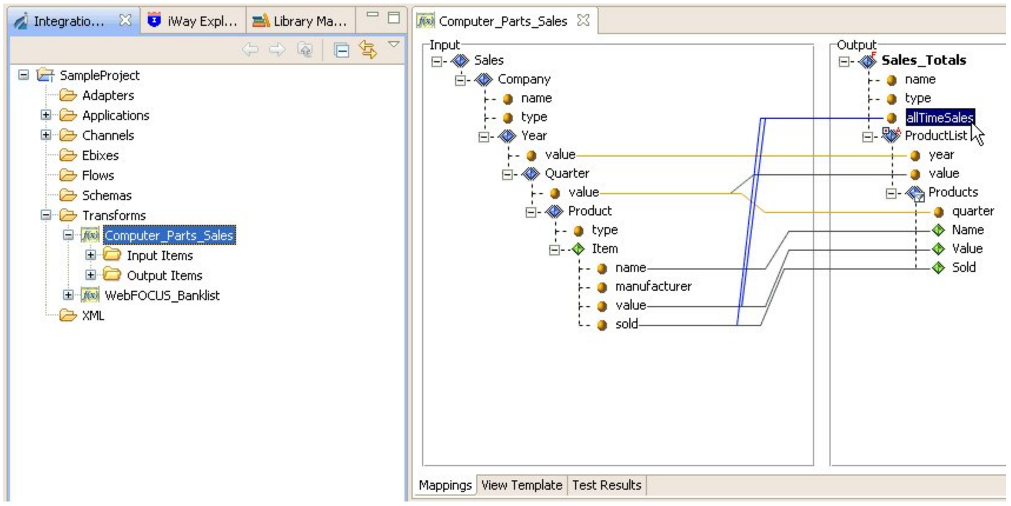
Browsing Mappings

This topic describes how to browse the mappings for a Transform project.

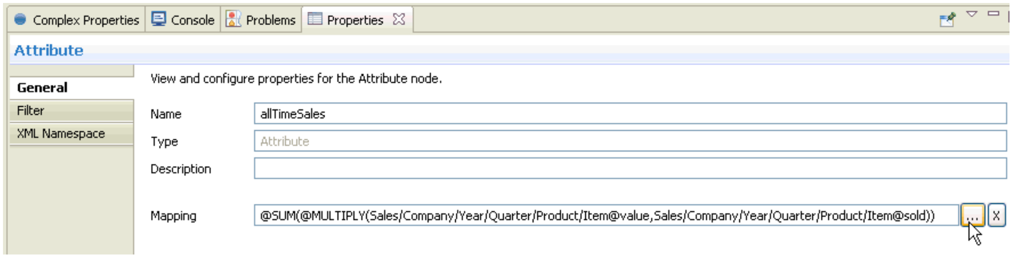
Procedure: How to Browse Mappings

To browse the mappings for a Transform project:

1. Click the *Mappings* tab for your Transform project (for example, Computer_Parts_Sales), as shown in the following image.



2. Select an output node in the Output pane (for example, the *allTimeSales* attribute). Properties of the selected output node are displayed in the Properties tab, as shown in the following image.

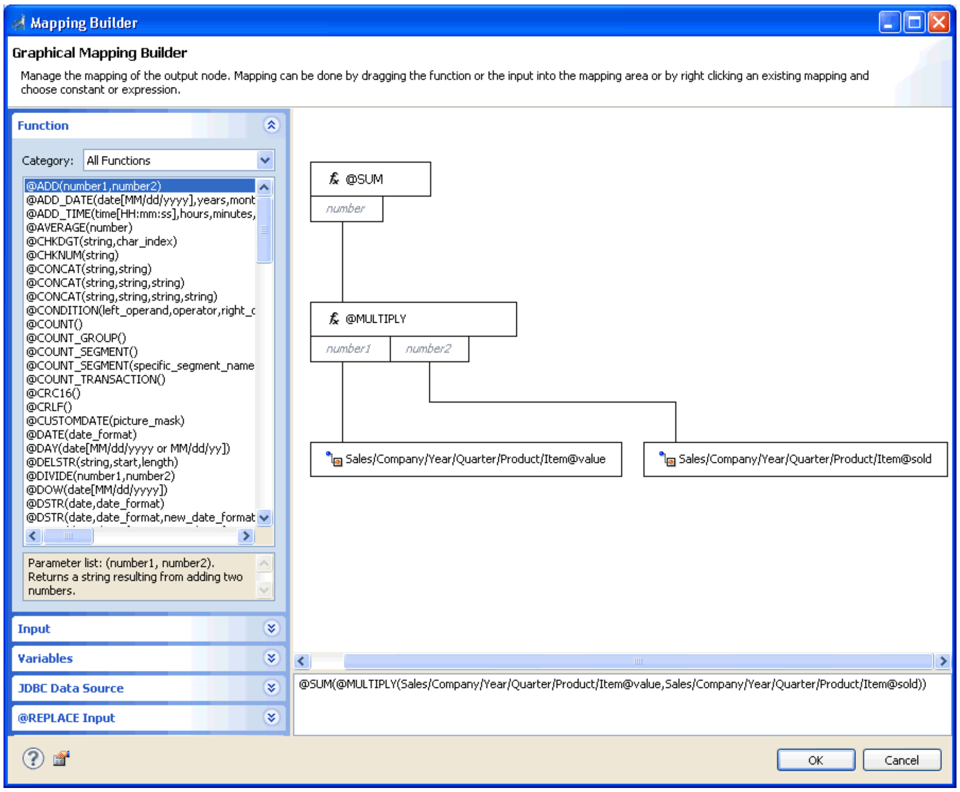


View and configure properties for the Attribute node.

Name	allTimeSales
Type	Attribute
Description	
Mapping	@SUM(@MULTIPLY(Sales/Company/Year/Quarter/Product/Item@value,Sales/Company/Year/Quarter/Product/Item@sold))

3. Click the ellipsis button to the right of the Mapping field.

The Mapping Builder opens, as shown in the following image.


Mapping Builder
Graphical Mapping Builder
Function
Input
Variables
JDBC Data Source
@REPLACE Input
@SUM
number
@MULTIPLY
number1
number2
Sales/Company/Year/Quarter/Product/Item@value
Sales/Company/Year/Quarter/Product/Item@sold
OK
Cancel

Manage the mapping of the output node. Mapping can be done by dragging the function or the input into the mapping area or by right clicking an existing mapping and choose constant or expression.

Category: All Functions

```
@ADD(number1,number2)
@ADD_DATE(date[MM/dd/yyyy],years,mont
@ADD_TIME(time[HH:mm:ss],hours,minutes,
@AVERAGE(number)
@CHKDGT(string,char_index)
@CHKNUM(string)
@CONCAT(string,string)
@CONCAT(string,string,string)
@CONCAT(string,string,string,string)
@CONDITION(left_operand,operator,right_c
@COUNT()
@COUNT_GROUP()
@COUNT_SEGMENT()
@COUNT_SEGMENT(specific_segment_name
@COUNT_TRANSACTION()
@CRC16()
@CRLF()
@CUSTOMDATE(picture_mask)
@DATE(date_format)
@DAY(date[MM/dd/yyyy or MM/dd/yy])
@DELSTR(string,start,length)
@DIVIDE(number1,number2)
@DOW(date[MM/dd/yyyy])
@DSTR(date,date_format)
@DSTR(date,date_format,new_date_format
```

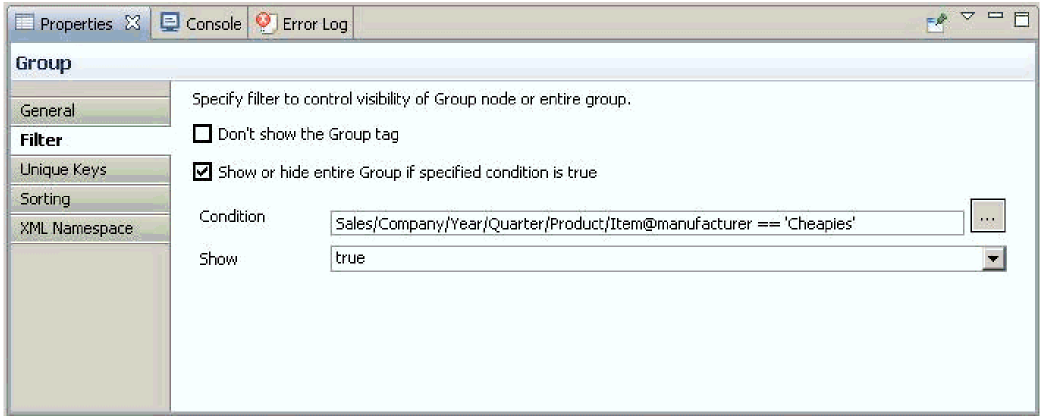
Parameter list: (number1, number2). Returns a string resulting from adding two numbers.

```
@SUM(@MULTIPLY(Sales/Company/Year/Quarter/Product/Item@value,Sales/Company/Year/Quarter/Product/Item@sold))
```

Editing Mappings

The example in this topic demonstrates how to edit output mappings to add a Summary node at the end of the root subtree to summarize the quantity and the amount of a sale. If you review the test results for the sample Computer_Parts_Sales Transform component, the filter set on the Product node ensures that the document contains only information on the Cheapies manufacturing brand. As a result, the summaries will reflect only Cheapies brand information.

The following image shows the filter that is set.



Specify filter to control visibility of Group node or entire group.

- [ ] Don't show the Group tag
- [✓] Show or hide entire Group if specified condition is true

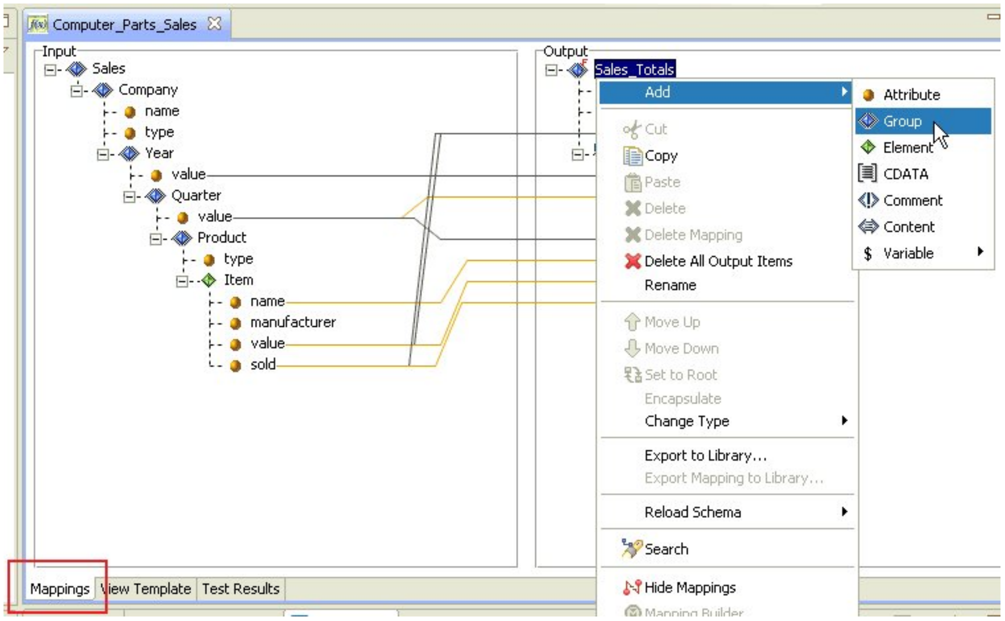
Condition: Sales/Company/Year/Quarter/Product/Item@manufacturer == 'Cheapies'

Show: true

Procedure: How to Edit Mappings

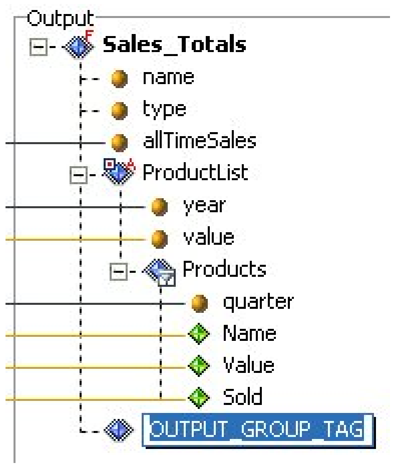
To edit mappings for a Transform component:

1. Click the *Mappings* tab, as shown in the following image.

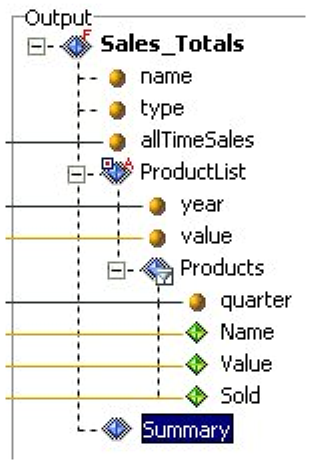


2. In the Output pane, right-click the *Sales_Totals* root group, select *Add*, and then click *Group* from the context menu.

A new group node is added to the output structure, as shown in the following image.



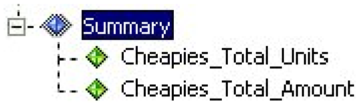
3. Change the default name for the group node to *Summary*, as shown in the following image.



4. Add two elements to the Summary group, using the following names:

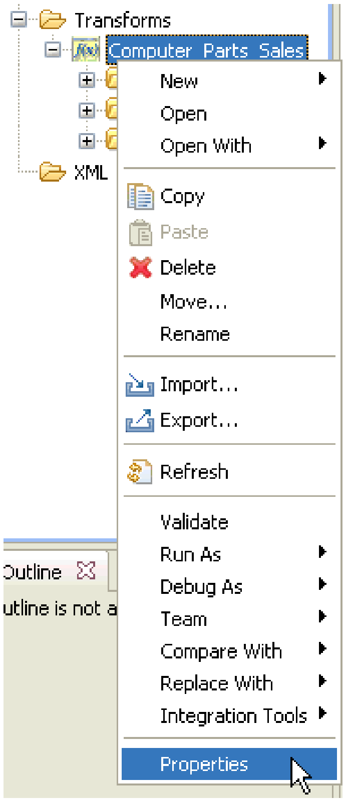
   - Cheapies_Total_Units
   - Cheapies_Total_Amount

   The following image shows how the Summary group will look.

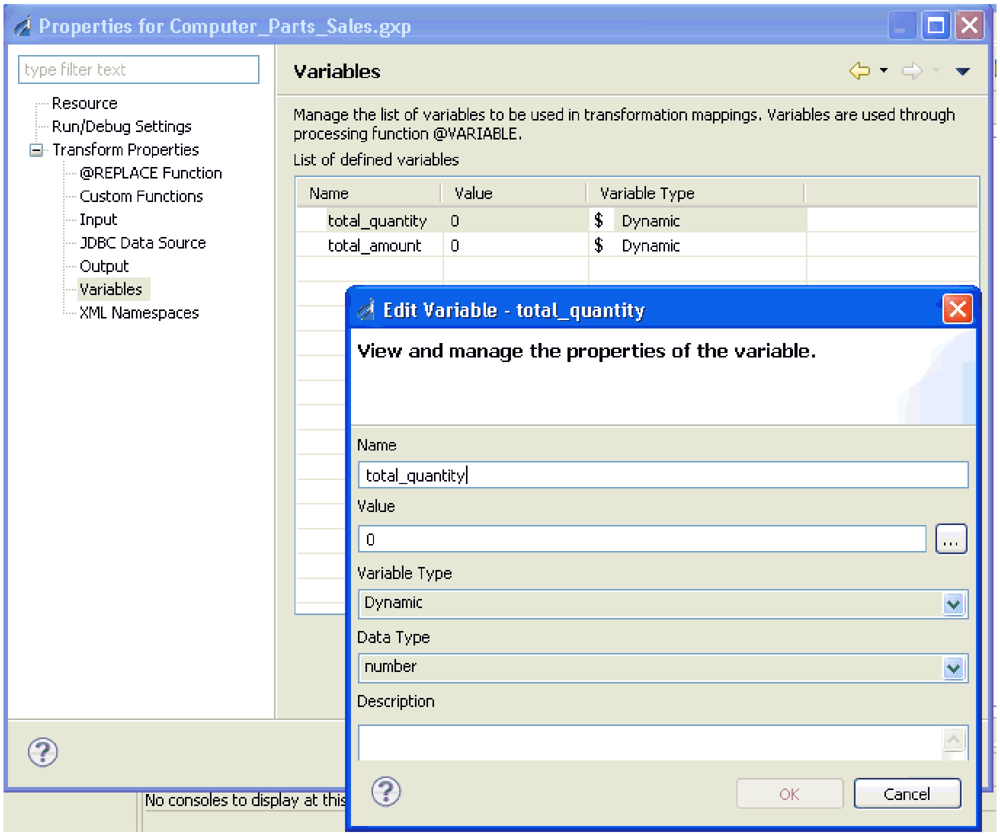


Now that the new output structure is configured, you need to configure variables to calculate the total values of the new elements (Cheapies_Total_Units and Cheapies_Total_Amount).

5. Right-click the *Computer_Parts_Sales* Transform component in the Integration Explorer tab, and select *Properties*, as shown in the following image.



The Properties dialog box for the Computer_Parts_Sales Transform component opens, as the next image shows.



Manage the list of variables to be used in transformation mappings. Variables are used through processing function @VARIABLE.

List of defined variables

Name	Value	Variable Type
total_quantity	0	$ Dynamic
total_amount	0	$ Dynamic

No consoles to display at this

Edit Variable - total_quantity

View and manage the properties of the variable.

Name: total_quantity

Value: 0

Variable Type: Dynamic

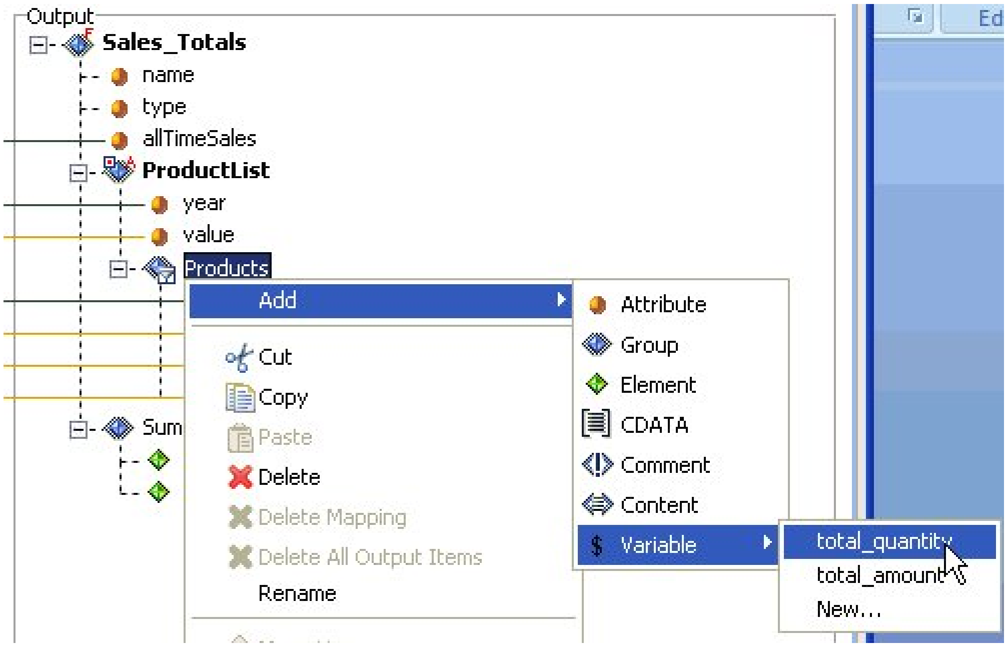
Data Type: number

Description:

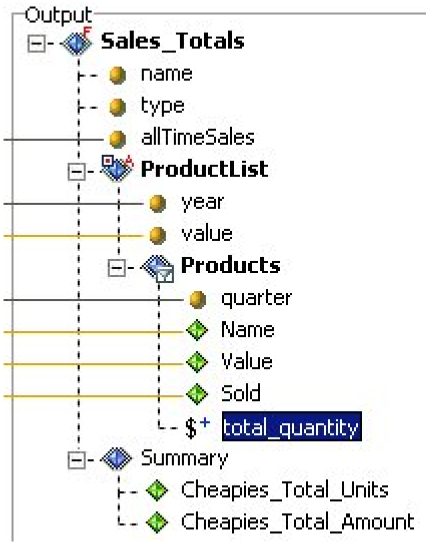
6. Expand *Transform Properties* in the left pane, and select the *Variables* category.
7. Configure two numeric dynamic variables (total_amount and total_quantity). Ensure that the value for both variables is 0.

   You are now ready to add calculations within each product node, which will update the variables appropriately as the document is parsed.

8. In the Output pane, right-click the *Products* group node, select *Add*, *Variable*, and then click *total_quantity*, as shown in the following image.

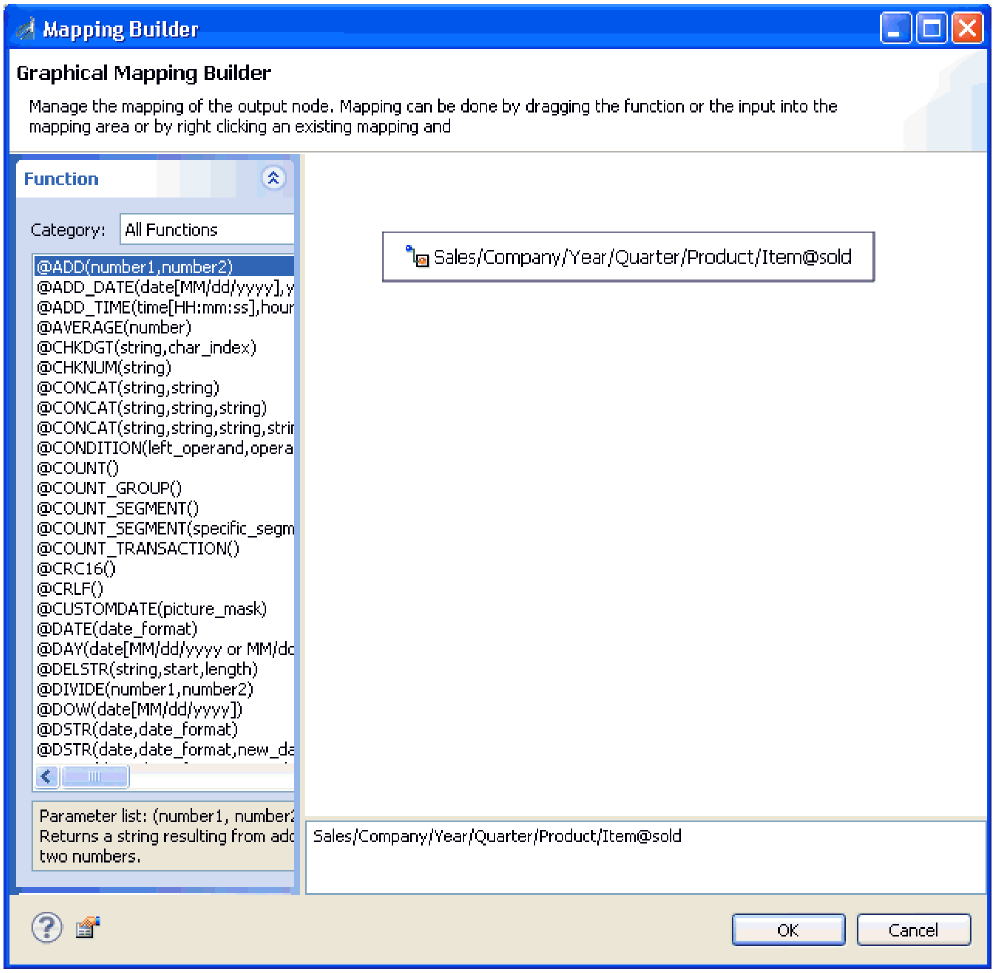


The total_quantity variable is added to the Products group node, as shown in the next image.



By default, the increment by 1 is the variable node that is created for the total_quantity variable. To calculate the total_quantity variable, you need to change it to increment by the quantity specified in a given node.

The following image shows the functions available in the Mapping Builder.



Manage the mapping of the output node. Mapping can be done by dragging the function or the input into the mapping area or by right clicking an existing mapping and

@ADD(number1,number2)
@ADD_DATE(date[MM/dd/yyyy],y
@ADD_TIME(time[HH:mm:ss],hour
@AVERAGE(number)
@CHKDGT(string,char_index)
@CHKNUM(string)
@CONCAT(string,string)
@CONCAT(string,string,string)
@CONCAT(string,string,string,strir
@CONDITION(left_operand,opera
@COUNT()
@COUNT_GROUP()
@COUNT_SEGMENT()
@COUNT_SEGMENT(specific_segm
@COUNT_TRANSACTION()
@CRC16()
@CRLF()
@CUSTOMDATE(picture_mask)
@DATE(date_format)
@DAY(date[MM/dd/yyyy or MM/dc
@DELSTR(string,start,length)
@DIVIDE(number1,number2)
@DOW(date[MM/dd/yyyy])
@DSTR(date,date_format)
@DSTR(date,date_format,new_da

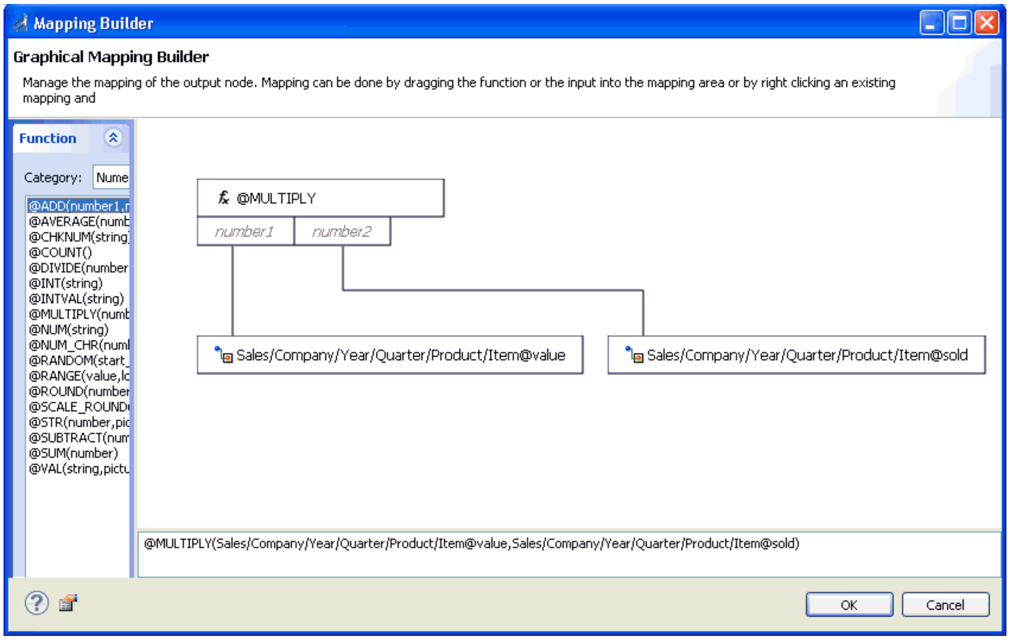
Parameter list: (number1, number2
Returns a string resulting from adc
two numbers.

Sales/Company/Year/Quarter/Product/Item@sold

Sales/Company/Year/Quarter/Product/Item@sold

9. To calculate the total_amount variable, create a formula that multiplies the number of units sold by the cost of each unit, and use the ADD function to create a sum for the total.

In the following image, the formula is shown, and the ADD function is selected from the list of functions on the left.

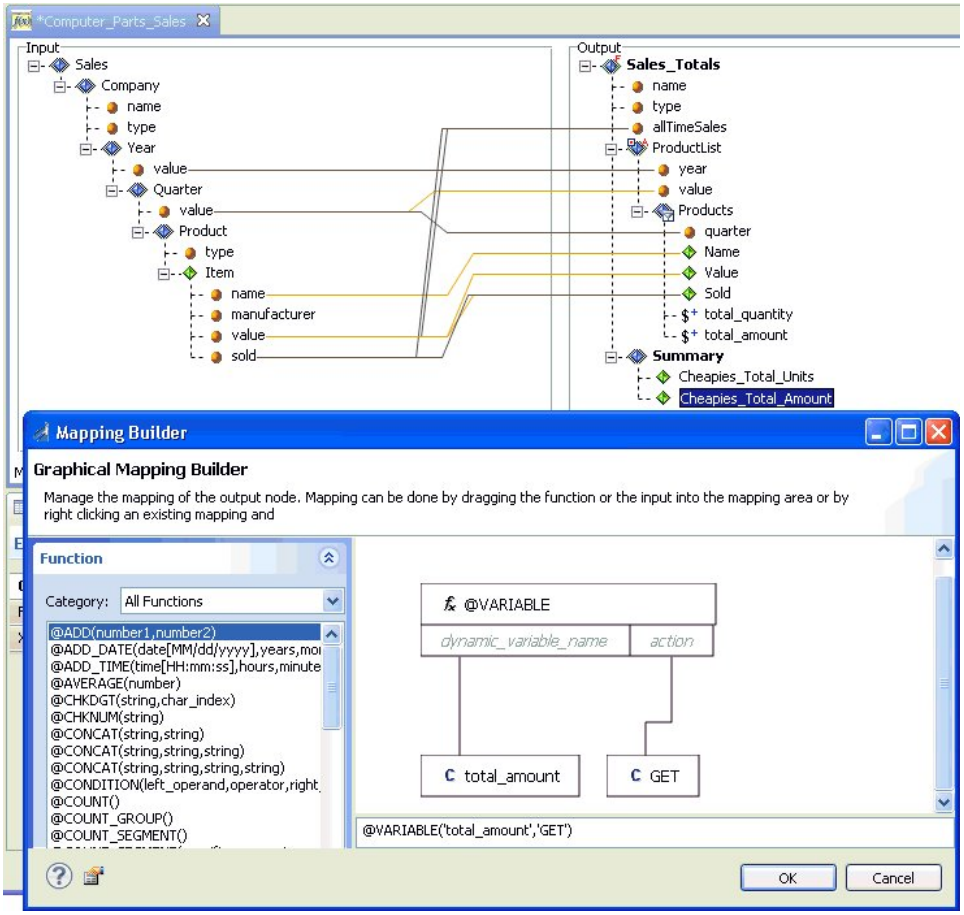

Mapping Builder
Graphical Mapping Builder
Function
Category: Nume
@MULTIPLY
number1
number2
Sales/Company/Year/Quarter/Product/Item@value
Sales/Company/Year/Quarter/Product/Item@sold
OK
Cancel

Manage the mapping of the output node. Mapping can be done by dragging the function or the input into the mapping area or by right clicking an existing mapping and

@ADD(number1,r
@AVERAGE(numb
@CHKNUM(string)
@COUNT()
@DIVIDE(number
@INT(string)
@INTVAL(string)
@MULTIPLY(numb
@NUM(string)
@NUM_CHR(numl
@RANDOM(start_
@RANGE(value,lc
@ROUND(number
@SCALE_ROUND(
@STR(number,pic
@SUBTRACT(num
@SUM(number)
@VAL(string,pictu

```
@MULTIPLY(Sales/Company/Year/Quarter/Product/Item@value,Sales/Company/Year/Quarter/Product/Item@sold)
```

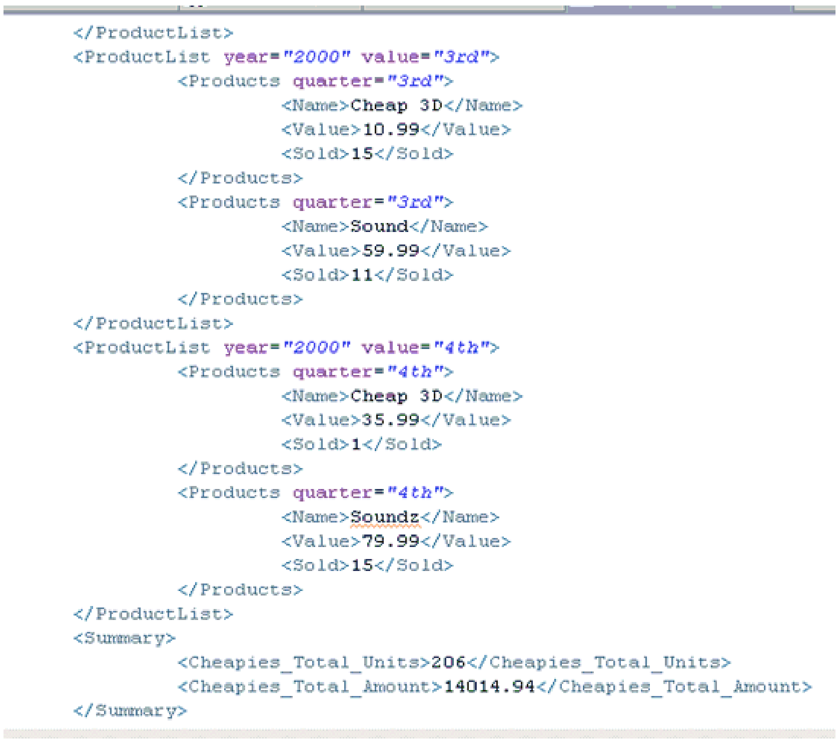
10. Assign the final values for both variables to their respective total nodes at the bottom of the output structure, as shown in the following image.



Manage the mapping of the output node. Mapping can be done by dragging the function or the input into the mapping area or by right clicking an existing mapping and

```
@VARIABLE('total_amount','GET')
```

11. Test the updated Computer_Parts_Sales Transform component, and review the results, which are shown in the following image.



```
      </ProductList>
      <ProductList year="2000" value="3rd">
               <Products quarter="3rd">
                        <Name>Cheap 3D</Name>
                        <Value>10.99</Value>
                        <Sold>15</Sold>
               </Products>
               <Products quarter="3rd">
                        <Name>Sound</Name>
                        <Value>59.99</Value>
                        <Sold>11</Sold>
               </Products>
      </ProductList>
      <ProductList year="2000" value="4th">
               <Products quarter="4th">
                        <Name>Cheap 3D</Name>
                        <Value>35.99</Value>
                        <Sold>1</Sold>
               </Products>
               <Products quarter="4th">
                        <Name>Soundz</Name>
                        <Value>79.99</Value>
                        <Sold>15</Sold>
               </Products>
      </ProductList>
      <Summary>
               <Cheapies_Total_Units>206</Cheapies_Total_Units>
               <Cheapies_Total_Amount>14014.94</Cheapies_Total_Amount>
      </Summary>
```

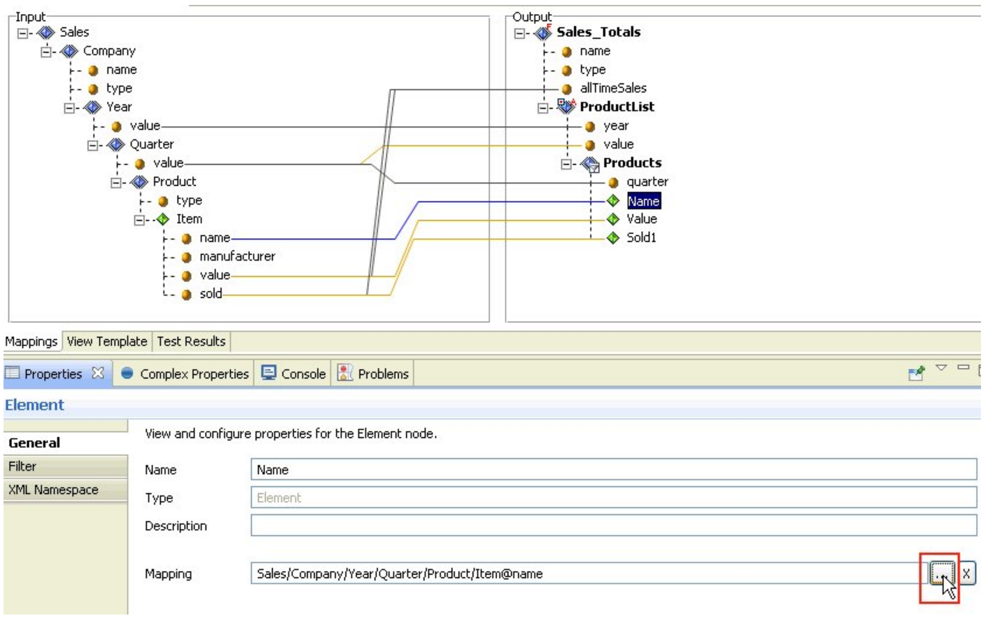
Using the Mapping Builder

The Mapping Builder is a tool available in iWay Transformer as part of the Mapping Builder facility. It allows you to construct an output node using various methods and formulas implemented in iWay Transformer as functions. You can access the predefined functions and the custom functions that you define, or build a statement containing several functions, if needed.

Procedure: How to Open the Mapping Builder

To open the Mapping Builder:

1. In the Output pane of the Mappings tab, select an output node, as shown in the following image.



View and configure properties for the Element node.

Name: Name

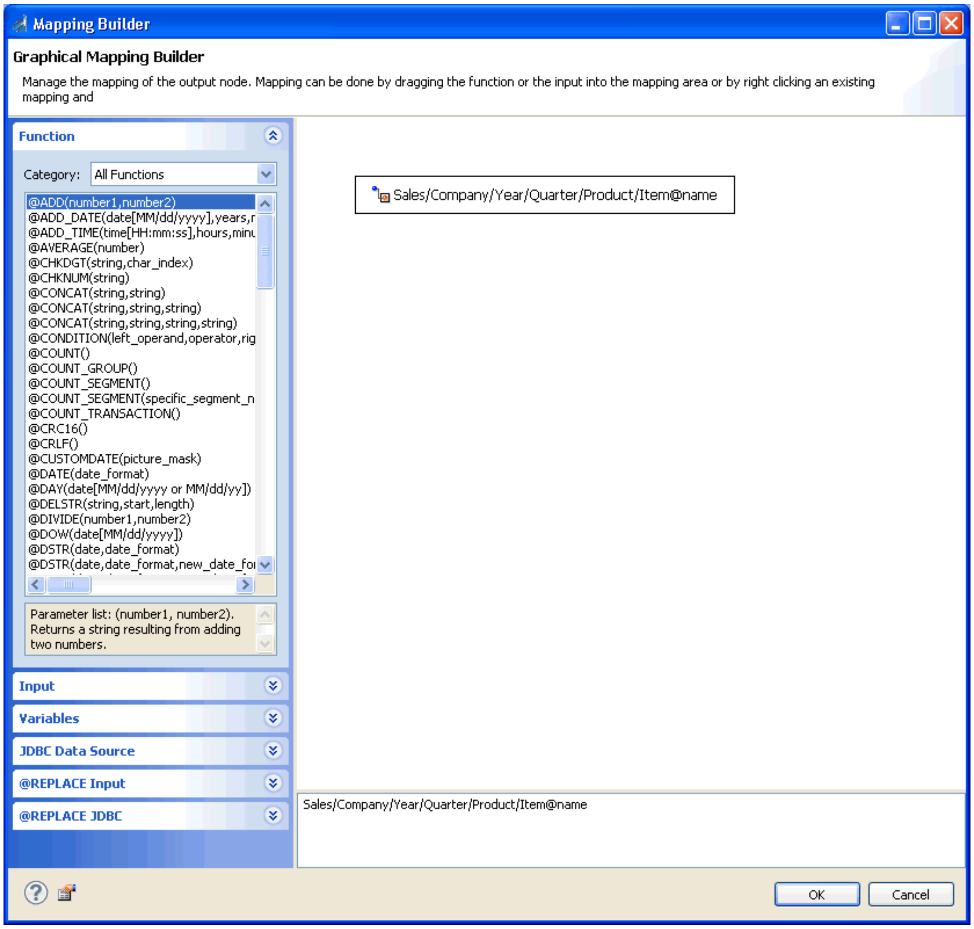
Type: Element

Description:

Mapping: Sales/Company/Year/Quarter/Product/Item@name

2. Click the ellipsis button on the Properties tab.

The Mapping Builder opens, as shown in the following image.


Mapping Builder
Graphical Mapping Builder
Manage the mapping of the output node. Mapping can be done by dragging the function or the input into the mapping area or by right clicking an existing mapping and
Function
Category: All Functions
@ADD(number1,number2)
@ADD_DATE(date[MM/dd/yyyy],years,r
@ADD_TIME(time[HH:mm:ss],hours,minu
@AVERAGE(number)
@CHKDGT(string,char_index)
@CHKNUM(string)
@CONCAT(string,string)
@CONCAT(string,string,string)
@CONCAT(string,string,string,string)
@CONDITION(left_operand,operator,rig
@COUNT()
@COUNT_GROUP()
@COUNT_SEGMENT()
@COUNT_SEGMENT(specific_segment_n
@COUNT_TRANSACTION()
@CRC16()
@CRLF()
@CUSTOMDATE(picture_mask)
@DATE(date_format)
@DAY(date[MM/dd/yyyy or MM/dd/yy])
@DELSTR(string,start,length)
@DIVIDE(number1,number2)
@DOW(date[MM/dd/yyyy])
@DSTR(date,date_format)
@DSTR(date,date_format,new_date_fo
Parameter list: (number1, number2). Returns a string resulting from adding two numbers.
Input
Variables
JDBC Data Source
@REPLACE Input
@REPLACE JDBC
Sales/Company/Year/Quarter/Product/Item@name
Sales/Company/Year/Quarter/Product/Item@name
OK
Cancel

For more information, see *Using the Mapping Builder* on page 269.

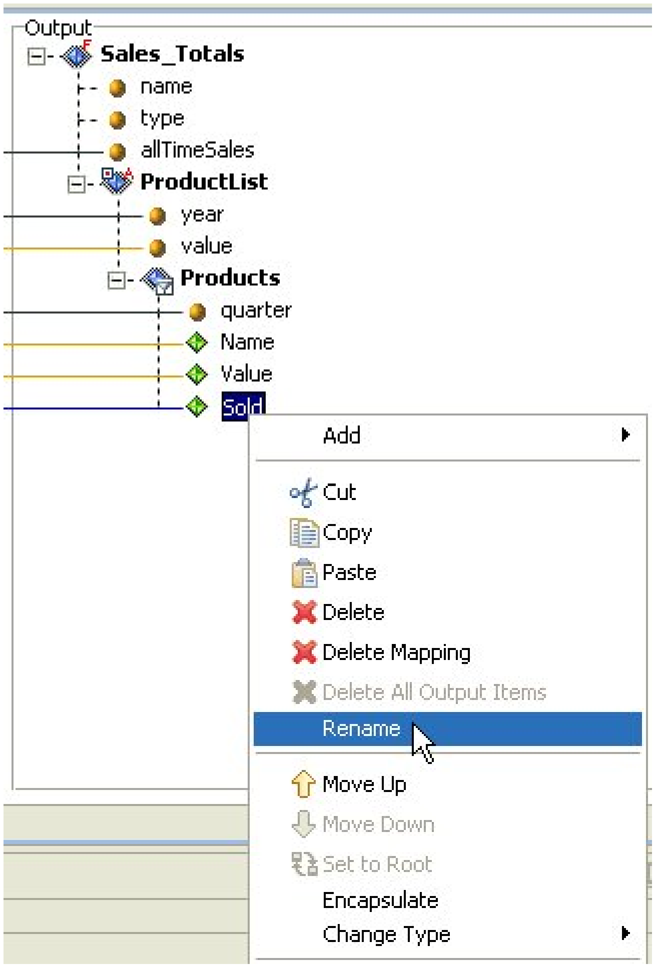
Renaming Mappings

To customize the output of your transform, you can rename an output structure node in your Transform component, if required.

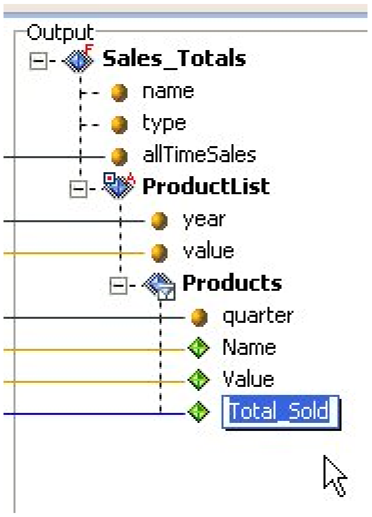
Procedure: How to Rename Mappings

To rename mappings:

1. Right-click an output structure node in the Output pane of the Mapping Builder, and select *Rename* from the context menu, as shown in the following image.



2. Type a new name for the output structure node, as shown in the next image.



Tip: The rename operation is also available if you double-click the name of the output structure node.

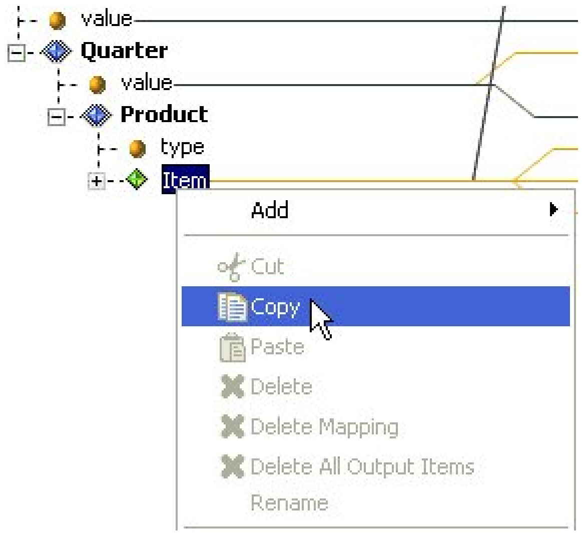
Copying and Moving Mappings

To modify the output structure of your transform, you can copy and move structure nodes in your Transform component, if required.

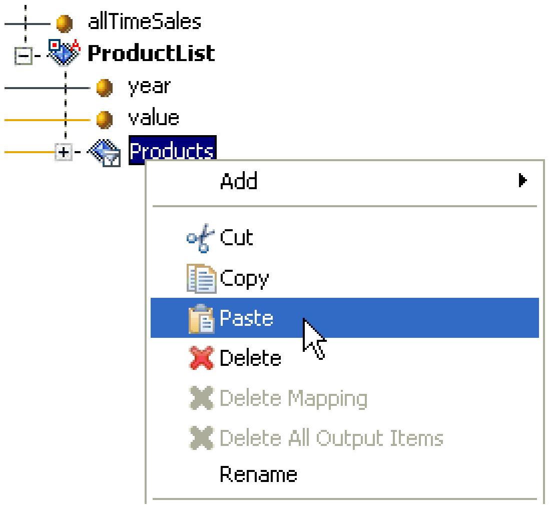
Procedure: How to Copy Mappings

Perform the following steps to copy an input structure node and paste it to the output structure:

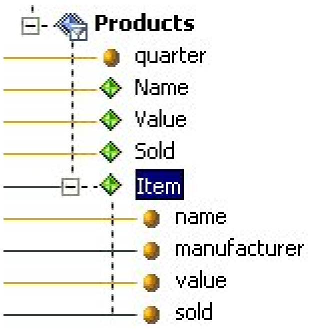
1. Right-click an input structure node (for example, the *Item* element) in the Input pane of the Mapping Builder, and select *Copy* from the context menu, as shown in the following image.



2. In the Output pane of the Mapping Builder, right-click an output structure node (for example, the *Products* group node) to which you want to add the copied input, and select *Paste* from the context menu, as shown in the following image.



A copy of the input structure node (for example, the Item element) is added to the output structure, as shown in the following image.

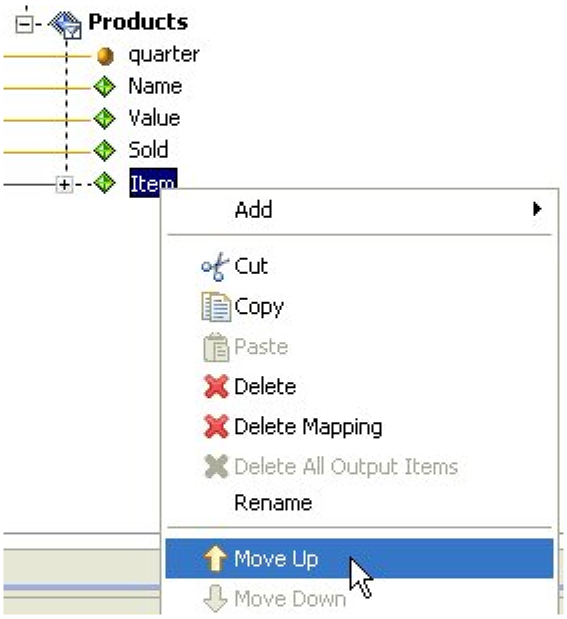


Procedure: How to Move Mappings

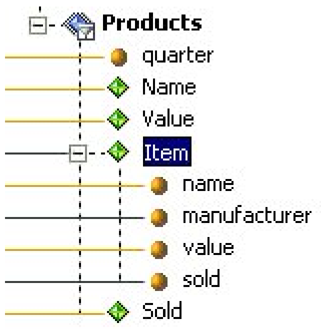
iWay Transformer provides a number of options for changing the order of output mappings.

Perform the following steps to rearrange an output node within the output structure:

Right-click an output structure node (for example, the *Item* element) in the Output pane of the Mapping Builder, and select *Move Up* from the context menu, as shown in the following image.



The output structure node is repositioned up by one level, as shown in the following image.



To reposition a selected output structure node down by one level, right-click the node, and select *Move Down* from the context menu. You can also click the *Move Up* and *Move Down* buttons on the toolbar to reposition nodes accordingly.

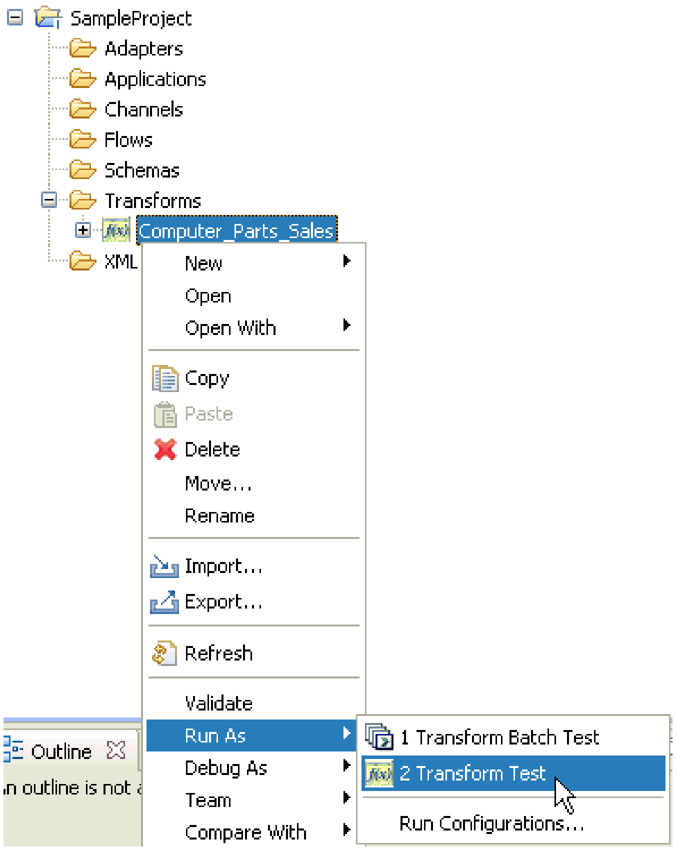
Running a Transform Component

Running a Transform component allows you to analyze the results of the transform.

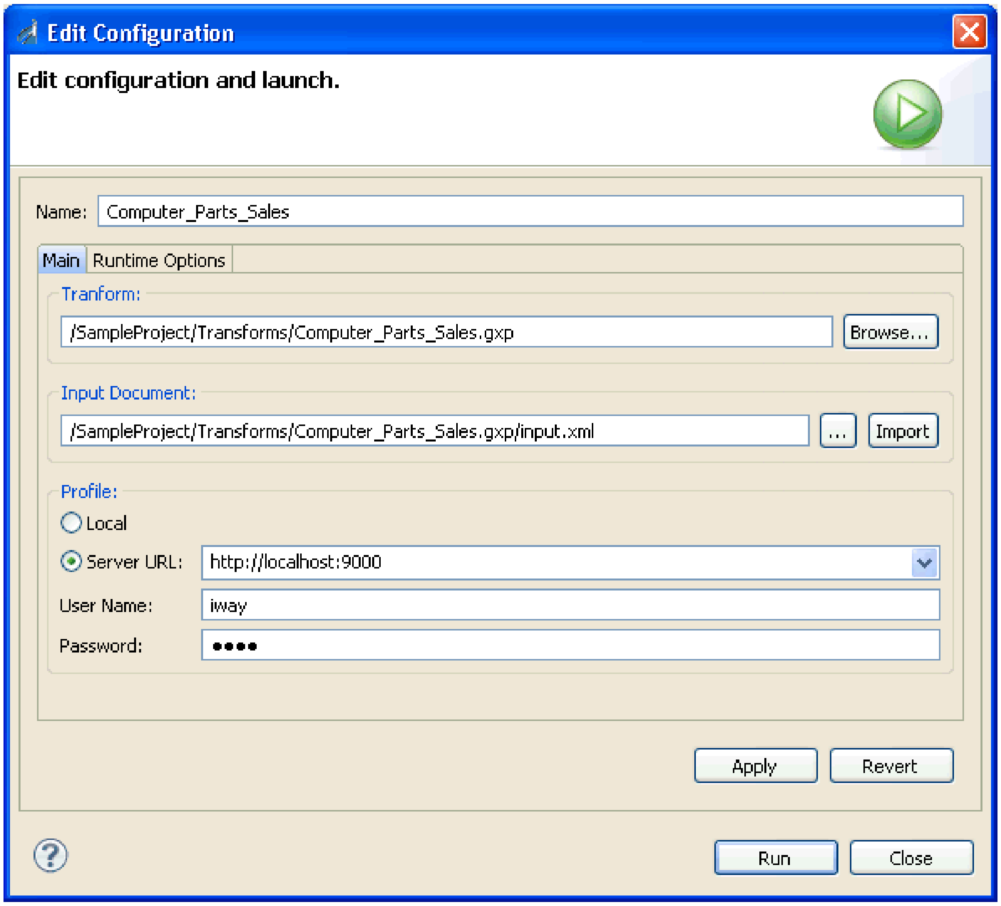
Procedure: How to Run a Transform Component

To run a Transform component:

1. Right-click the Transform component (for example, *Computer_Parts_Sales*) in the Transforms folder of your integration project, select *Run As*, and then click *Transform Test*, as shown in the following image.



The Edit Configuration dialog box opens, as shown in the next image.



Name: Computer_Parts_Sales

Tranform: /SampleProject/Transforms/Computer_Parts_Sales.gxp

Input Document: /SampleProject/Transforms/Computer_Parts_Sales.gxp/input.xml

Profile:

Local

[✓ Server URL:] http://localhost:9000

User Name: iway

Password: ●●●●

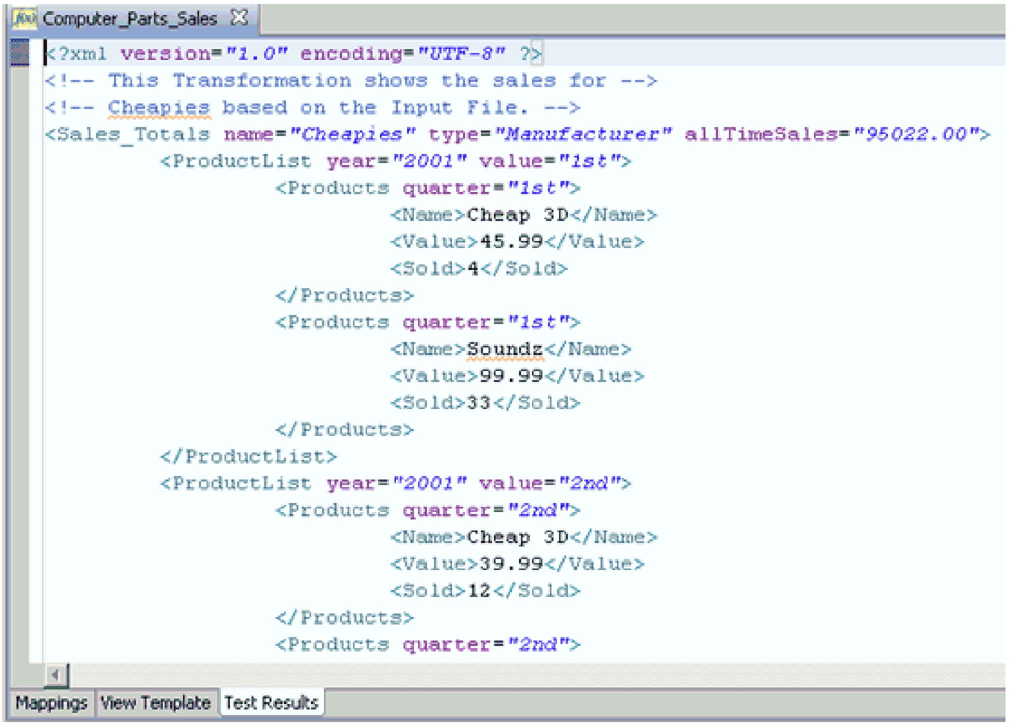
2. Select the *Server URL* option in the Profile area, and then select an available URL that is used by iWay Explorer from the drop-down list.

3. Click *Apply*.

   When the Apply button is grayed out, the option is applied.

4. Click *Run*.

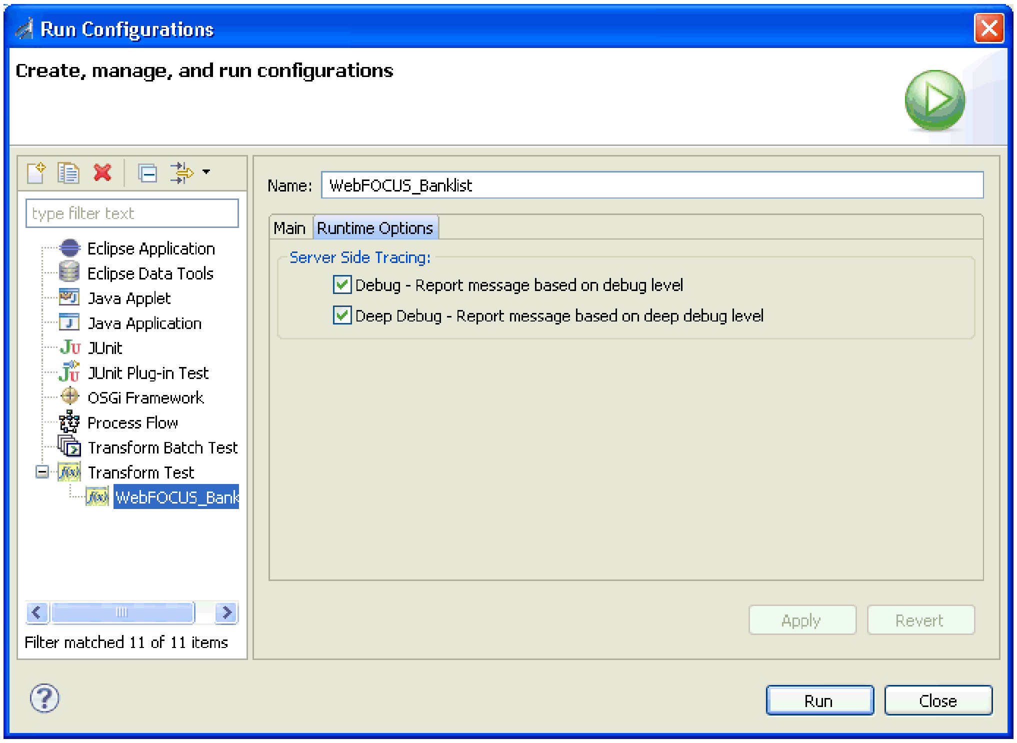
The results are displayed in the Test Results tab, as shown in the following image.



```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This Transformation shows the sales for -->
<!-- Cheapies based on the Input File. -->
<Sales_Totals name="Cheapies" type="Manufacturer" allTimeSales="95022.00">
         <ProductList year="2001" value="1st">
                  <Products quarter="1st">
                           <Name>Cheap 3D</Name>
                           <Value>45.99</Value>
                           <Sold>4</Sold>
                  </Products>
                  <Products quarter="1st">
                           <Name>Soundz</Name>
                           <Value>99.99</Value>
                           <Sold>33</Sold>
                  </Products>
         </ProductList>
         <ProductList year="2001" value="2nd">
                  <Products quarter="2nd">
                           <Name>Cheap 3D</Name>
                           <Value>39.99</Value>
                           <Sold>12</Sold>
                  </Products>
                  <Products quarter="2nd">
```

Debugging a Transform Component

The Runtime Options tab allows you to debug a Transform component, as shown in the following image.



Name: WebFOCUS_Banklist

Server Side Tracing:

- [✓] Debug - Report message based on debug level
- [✓] Deep Debug - Report message based on deep debug level

Working With Other Editors

For more information, see *Dictionary Builder Tutorial* on page 80.

Project Configuration Tutorial

This topic provides a project configuration tutorial for iWay Transformer.

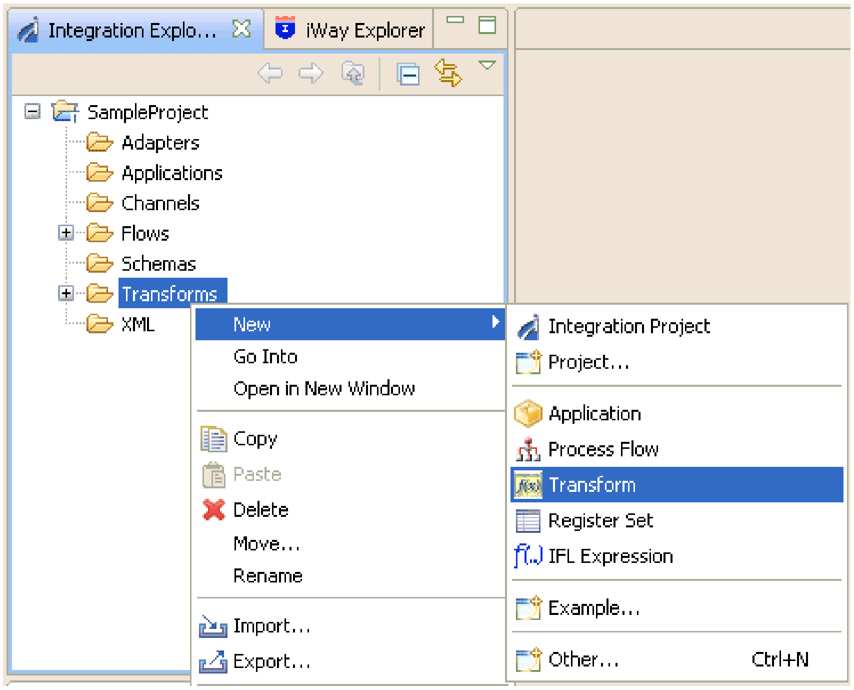
Configuring a New Transform Component

This topic describes how to configure a new Transform component.

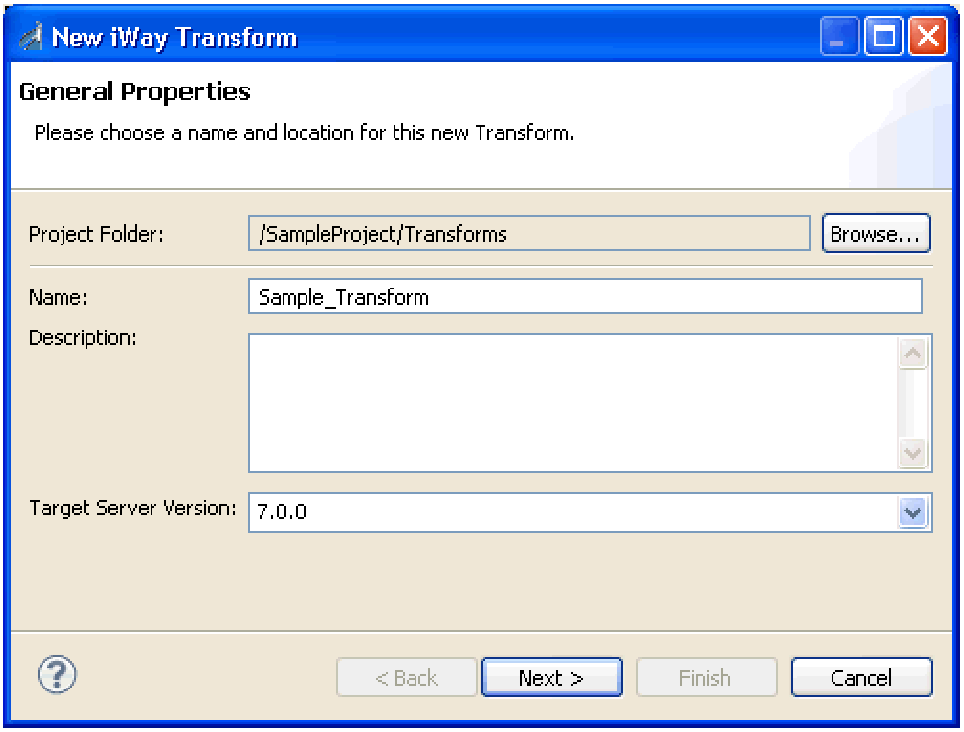
Procedure: How to Configure a New Transform Component

To configure a new Transform component:

1. Right-click the *Transforms* folder in your Integration Project, select *New*, and then click *Transform*, as shown in the following image.



The New iWay Transform dialog box opens, as shown in the following image.


New iWay Transform

General Properties

Please choose a name and location for this new Transform.

Project Folder: /SampleProject/Transforms

Name: Sample_Transform

Description:

Target Server Version: 7.0.0

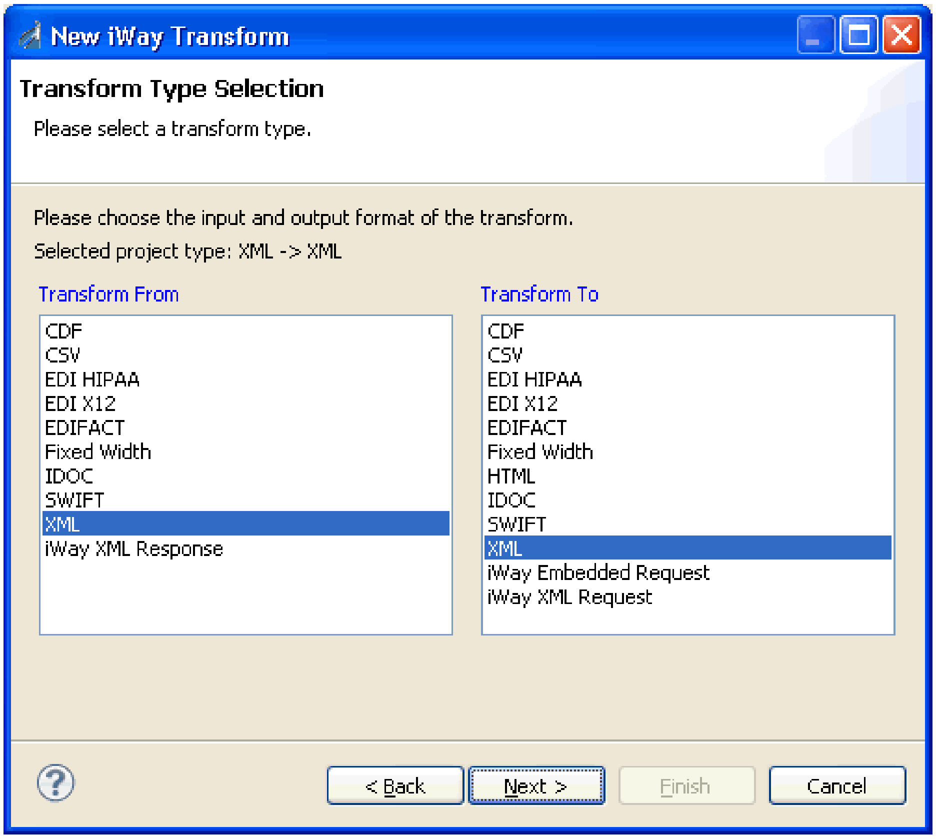
2. Type a name and a description (optional) for your new Transform component.

   If required, you can change the preselected project folder by clicking *Browse* next to the Project Folder field.

3. Select a target version of iWay Service Manager (iSM) from the Target Server Version drop-down list.

4. Click *Next*.

The Transform Type Selection pane opens, as shown in the following image.



Transform Type Selection

Please select a transform type.

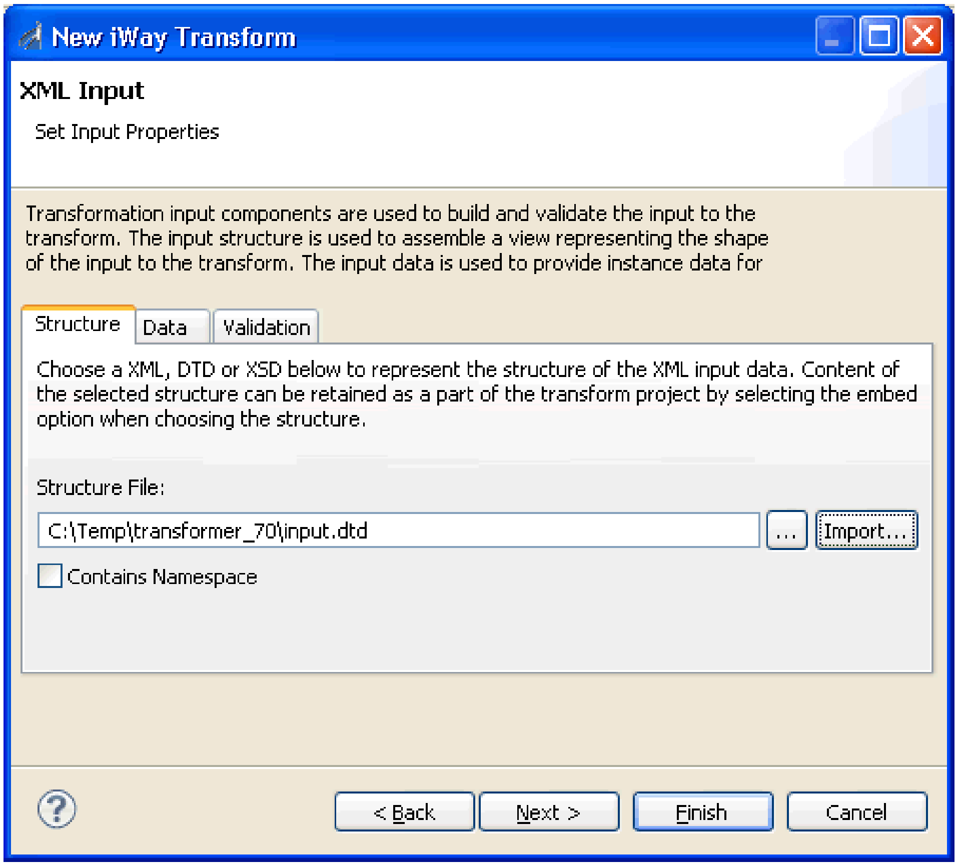
Please choose the input and output format of the transform.

Selected project type: XML -> XML

Transform From	Transform To
CDF	CDF
CSV	CSV
EDI HIPAA	EDI HIPAA
EDI X12	EDI X12
EDIFACT	EDIFACT
Fixed Width	Fixed Width
IDOC	HTML
SWIFT	IDOC
XML	SWIFT
iWay XML Response	XML
	iWay Embedded Request
	iWay XML Request

5. From the list in the Transform From section, select the format of your input, for example, *XML*.
6. From the list in the Transform To section, select the format of your output data, for example, *XML*.
7. Click *Next*.

The XML Input pane opens with the Structure tab selected, as shown in the following image.



Transformation input components are used to build and validate the input to the transform. The input structure is used to assemble a view representing the shape of the input to the transform. The input data is used to provide instance data for

Choose a XML, DTD or XSD below to represent the structure of the XML input data. Content of the selected structure can be retained as a part of the transform project by selecting the embed option when choosing the structure.

Structure File: C:\Temp\transformer_70\input.dtd ... Import...

☐ Contains Namespace

Tip: You can skip the remaining steps by clicking the *Finish* button if it is enabled. You can then manually alter the Transform component properties at your convenience from the Properties dialog box.

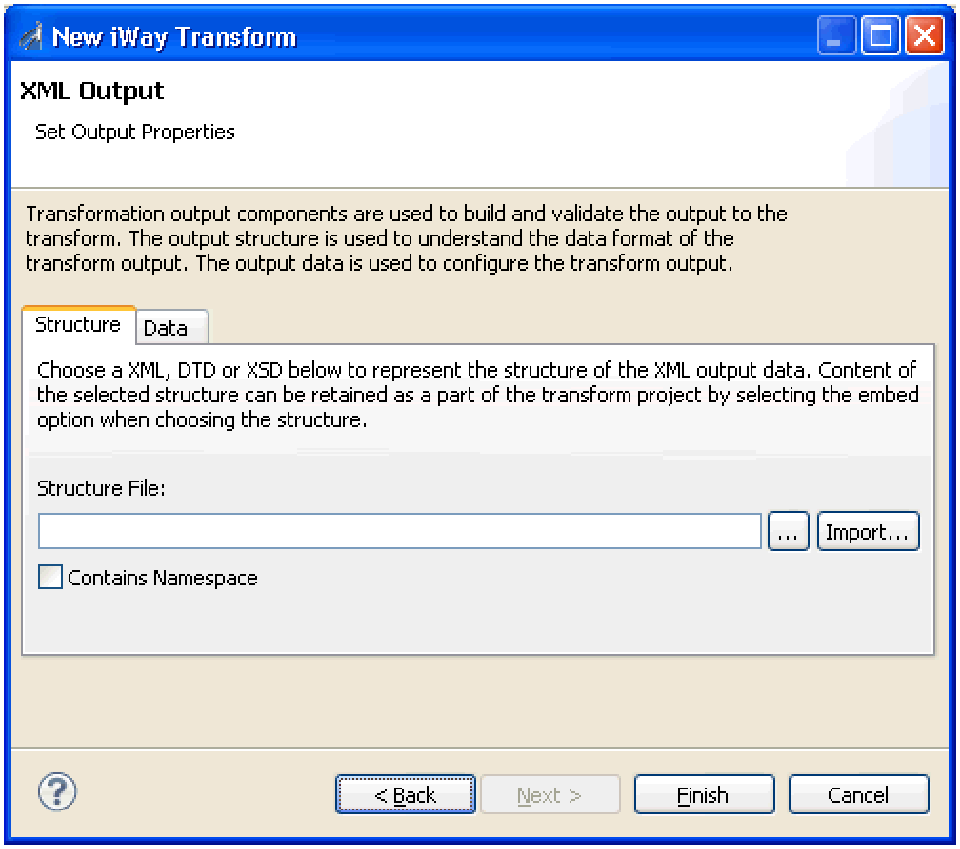
8. In the Structure File field, type the name of a DTD, XSD (schema), or XML file that represents the input dictionary.

   ❑ Click the ellipsis button to the right of the field to browse the Eclipse workspace for the structure file.

   ❑ Click the *Import* button to the right of the field to browse your file system for the structure file.

9. Click *Next*.

The XML Output pane opens with the Structure tab selected, as shown in the following image.



Transformation output components are used to build and validate the output to the transform. The output structure is used to understand the data format of the transform output. The output data is used to configure the transform output.

Choose a XML, DTD or XSD below to represent the structure of the XML output data. Content of the selected structure can be retained as a part of the transform project by selecting the embed option when choosing the structure.

Structure File:

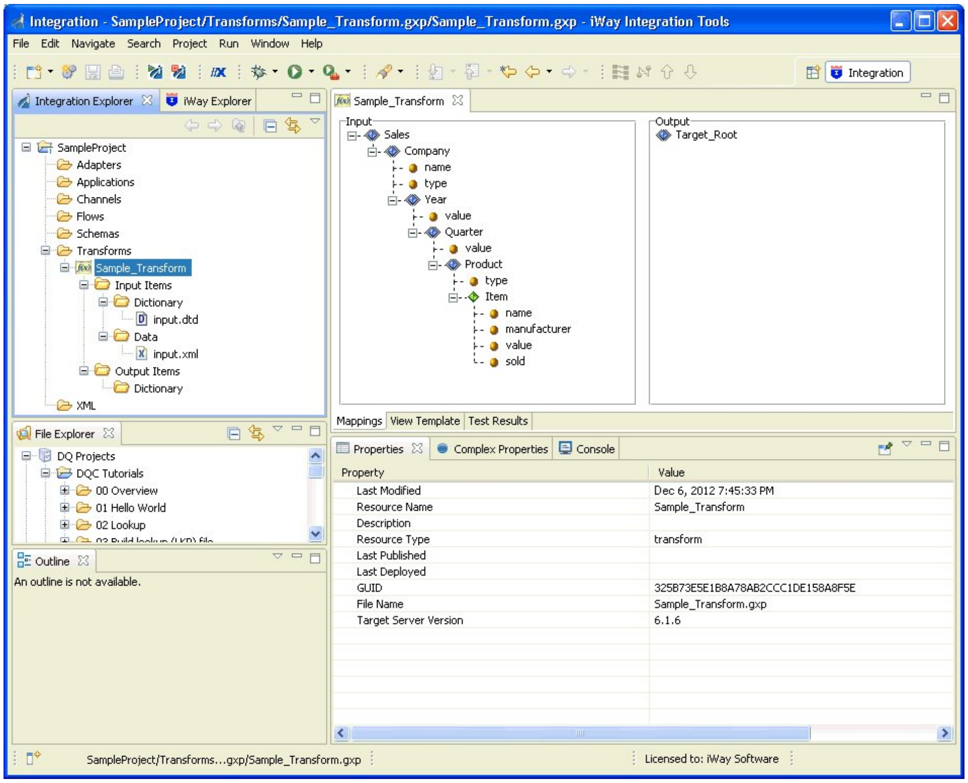
10. In the Structure File field, type the name of the data dictionary that represents the structure of the output data (optional).

    - ❑ Click the ellipsis button to the right of the field to browse the Eclipse workspace for the structure file.

    - ❑ Click the *Import* button to the right of the field to browse your file system for the structure file.

11. Click *Finish*.

Your new Transform component is displayed in the iIT Eclipse workspace, as shown in the following image.



Property	Value
Last Modified	Dec 6, 2012 7:45:33 PM
Resource Name	Sample_Transform
Description	
Resource Type	transform
Last Published	
Last Deployed	
GUID	325B73E5E1B8A78AB2CCC1DE158A8F5E
File Name	Sample_Transform.gxp
Target Server Version	6.1.6

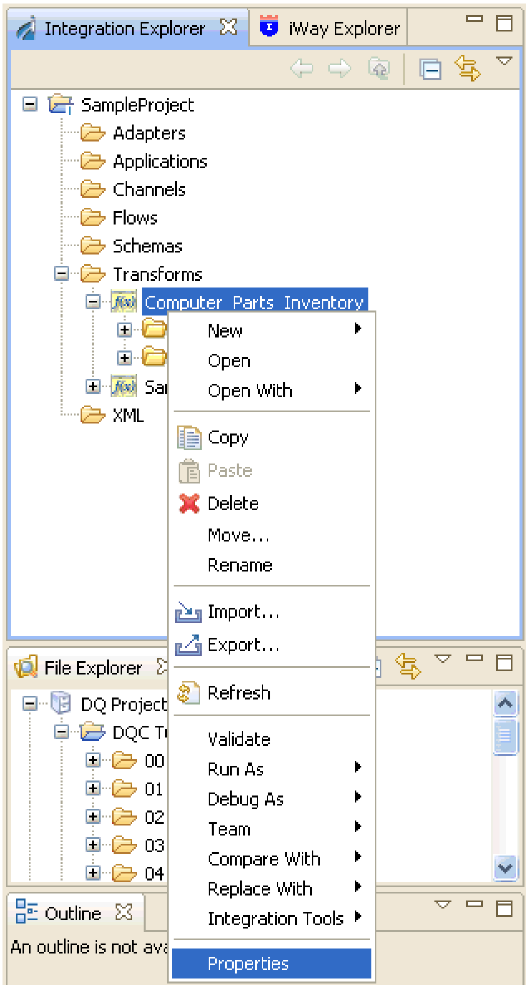
The Transform component and any related resources (for example, structure files and input data) exist in the workspace subdirectory that you named during the creation process. For example:

`iIT_Home\workspace\SampleProject\Transforms\Sample_Transform.gxp`

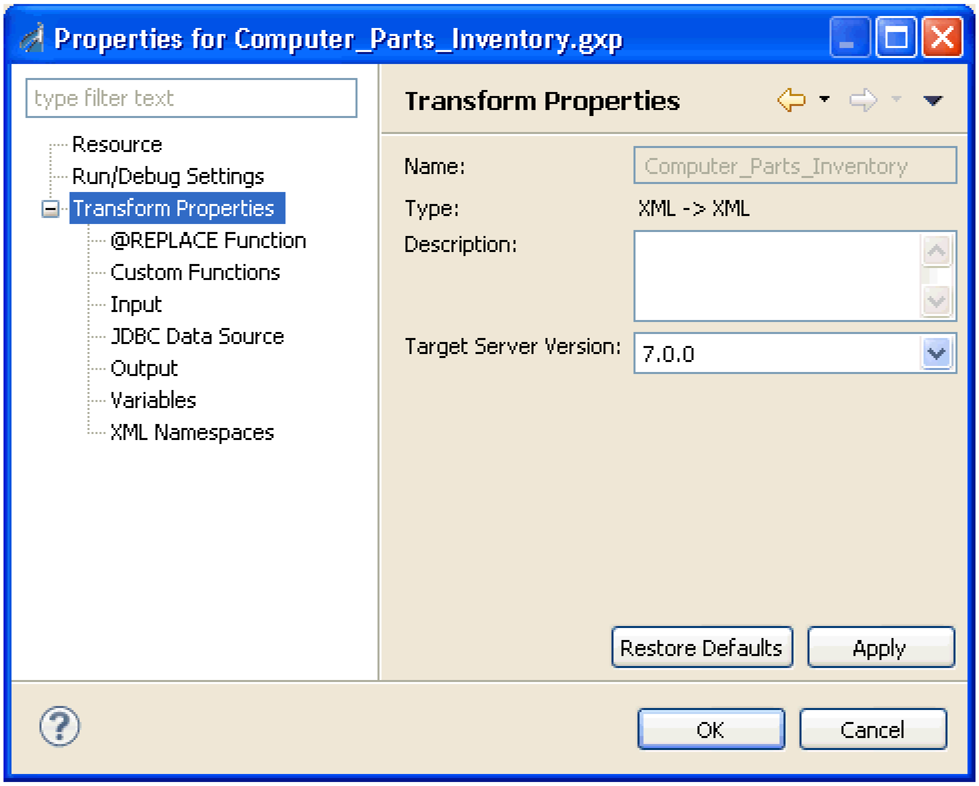
Configuring Transform Component Properties

The Transform component Properties dialog box enables you to view or modify the set of properties defined for a Transform component, such as project creation date and custom functions.

To access the Transform component Properties dialog box, right-click a Transform component project name in the Integration Explorer pane, and select *Properties* from the context menu, as shown in the following image.



The Properties dialog box for the selected Transform component opens, as shown in the following image.


Properties for Computer_Parts_Inventory.gxp

type filter text

- Resource
- Run/Debug Settings
- Transform Properties
  - @REPLACE Function
  - Custom Functions
  - Input
  - JDBC Data Source
  - Output
  - Variables
  - XML Namespaces

Transform Properties

Name: Computer_Parts_Inventory

Type: XML -> XML

Description:

Target Server Version: 7.0.0

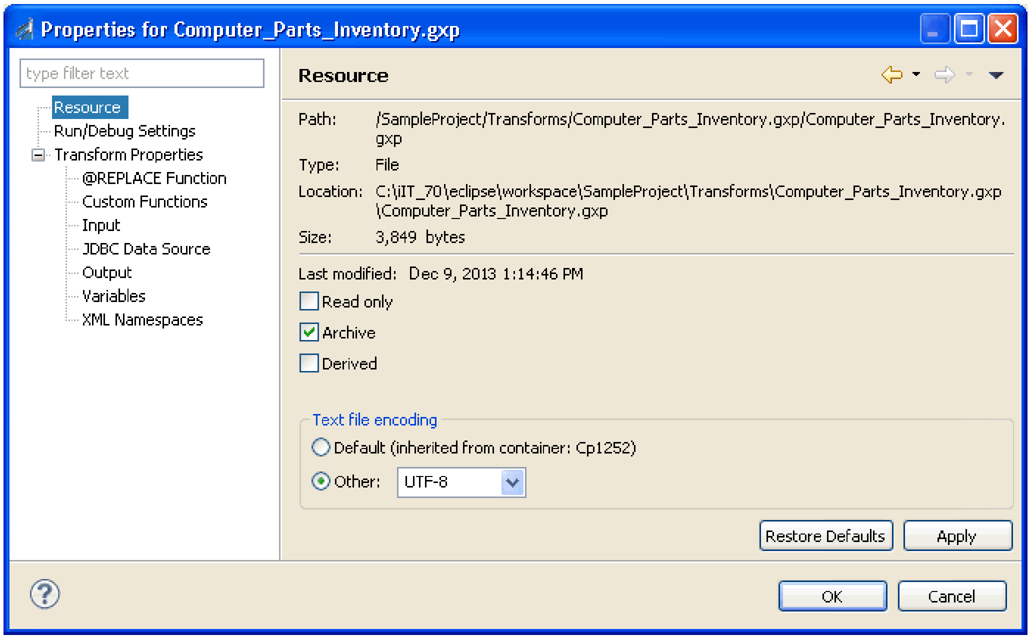
The Properties dialog box includes the following categories:

- **Resource.** For more information, see *Resource* on page 71.
- **Run/Debug Settings.** For more information, see *Run/Debug Settings* on page 72.
- **Transform Properties.** For more information, see *Transform Properties* on page 72.
  - **@REPLACE Function.** For more information, see *@REPLACE Function* on page 73.
  - **Custom Functions.** For more information, see *Custom Functions* on page 74.
  - **Input.** For more information, see *Input* on page 75.
  - **JDBC Data Source.** For more information, see *JDBC Data Source* on page 76.
  - **Output.** For more information, see *Output* on page 77.

- **Variables.** For more information, see *Variables* on page 78.
- **XML Namespaces.** For more information, see *XML Namespaces* on page 79.

Resource

The Resource category displays system-level information about the Transform component resource file, as shown in the following image.



Resource

Path: /SampleProject/Transforms/Computer_Parts_Inventory.gxp/Computer_Parts_Inventory.gxp

Type: File

Location: C:\iIT_70\eclipse\workspace\SampleProject\Transforms\Computer_Parts_Inventory.gxp\Computer_Parts_Inventory.gxp

Size: 3,849 bytes

Last modified: Dec 9, 2013 1:14:46 PM

- [ ] Read only
- [✓] Archive
- [ ] Derived

Text file encoding

Default (inherited from container: Cp1252) [✓ Other: UTF-8]

The following information is available:

- Path
- Type
- Location
- Size
- Last Modified

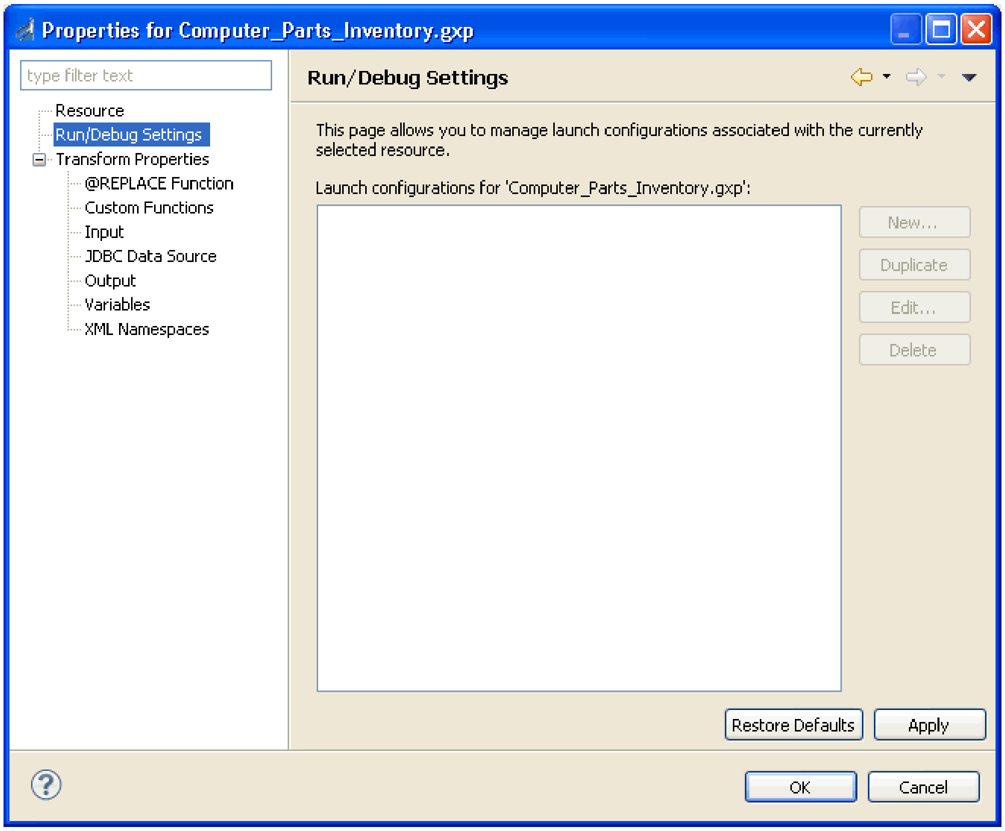
You can set Read only, Archive, and Derived file access options for the resource if required. By default, the Archive option is set.

The Text file encoding section allows you to specify the type of encoding that is used for the characters in a file. By default, the character encoding is set to UTF-8.

Run/Debug Settings

The Run/Debug Settings category allows you to manage launch configurations that are associated with the resource that is currently selected.

The following image shows the Run/Debug Settings pane.



Properties for Computer_Parts_Inventory.gxp

Run/Debug Settings

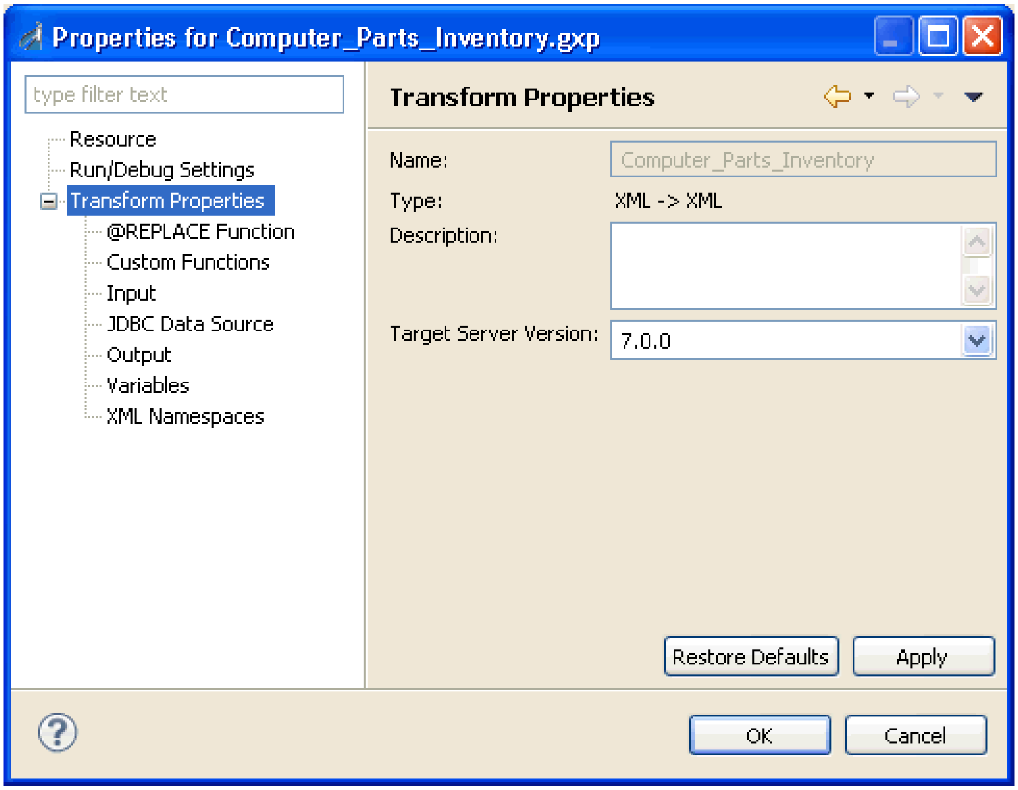
This page allows you to manage launch configurations associated with the currently selected resource.

Launch configurations for 'Computer_Parts_Inventory.gxp':

Transform Properties

The Transform Properties category includes basic information about the Transform component.

The following image shows the Transform Properties pane.



Name: Computer_Parts_Inventory

Type: XML -> XML

Description:

Target Server Version: 7.0.0

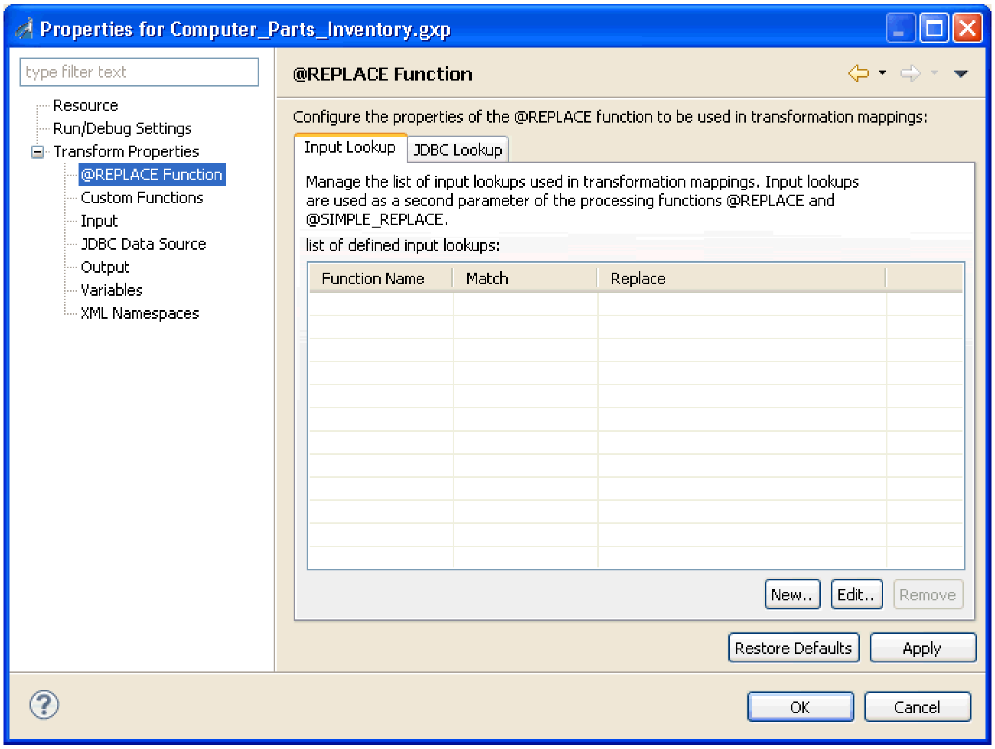
The following properties are listed:

- **Name.** Name of your Transform component.
- **Type.** Input and output format of your Transform component. If the format of the input or output is changed, the Type property will reflect those changes.
- **Description.** Custom description for your Transform component.
- **Target Server Version.** The target version of iWay Service Manager (iSM) that is selected for this Transform component.

@REPLACE Function

The @REPLACE Function category provides a way to instantaneously match and replace certain input data values. Each individually configured replace function works similarly to custom functions, in that you must first define the function and then apply it in the output node mapping value definition that you want to affect.

The following image shows the @REPLACE Function pane.



Configure the properties of the @REPLACE function to be used in transformation mappings:

Manage the list of input lookups used in transformation mappings. Input lookups are used as a second parameter of the processing functions @REPLACE and @SIMPLE_REPLACE.

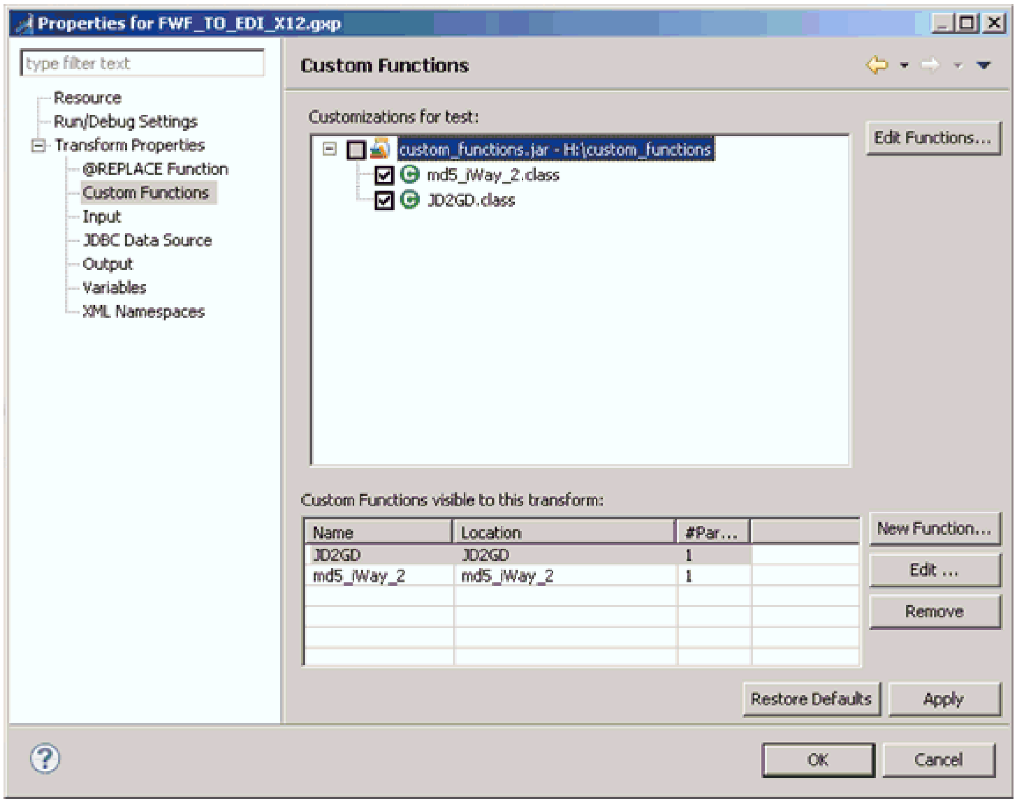
list of defined input lookups:

Function Name	Match	Replace

Custom Functions

The Custom Functions category enables you to manage the list of custom functions that you can build on-demand, when a predefined iWay Transformer function does not exist to perform the task that you require. You must write custom functions in Java and store them on your system so that they are available for use with iWay Transformer during design time. You must configure custom functions differently for use at run time.

The following image shows the Custom Functions pane.

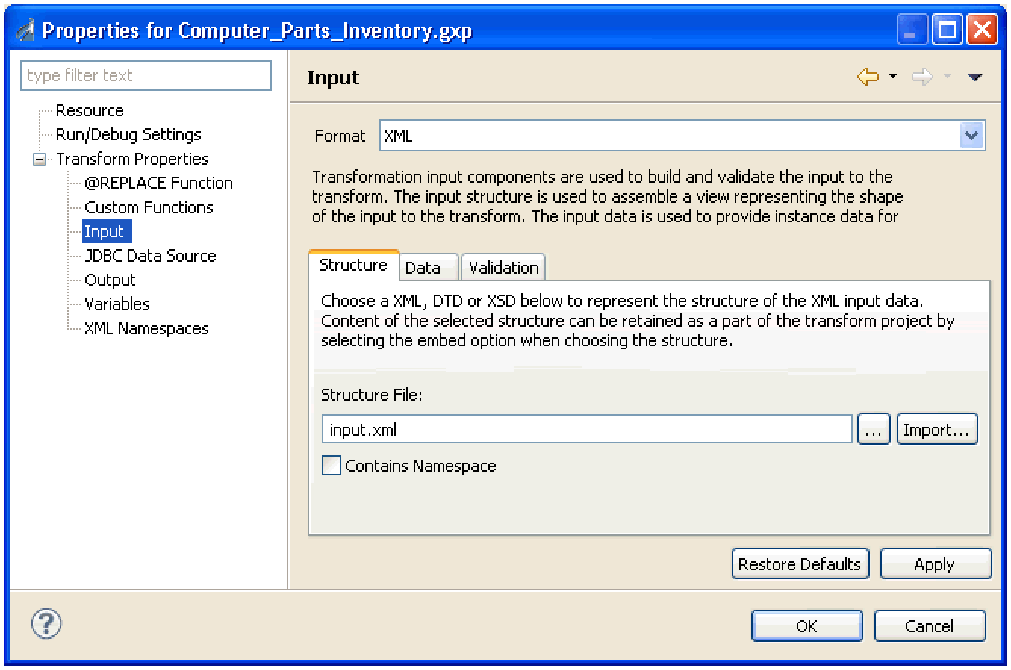


Name	Location	#Par...
JD2GD	JD2GD	1
md5_iWay_2	md5_iWay_2	1

Input

The Input category allows you to configure the input properties for the Transform component.

The following image shows the Input pane.


Properties for Computer_Parts_Inventory.gxp

type filter text

- Resource
- Run/Debug Settings
- Transform Properties
  - @REPLACE Function
  - Custom Functions
  - Input
  - JDBC Data Source
  - Output
  - Variables
  - XML Namespaces

Input

Format: XML

Transformation input components are used to build and validate the input to the transform. The input structure is used to assemble a view representing the shape of the input to the transform. The input data is used to provide instance data for

Structure | Data | Validation

Choose a XML, DTD or XSD below to represent the structure of the XML input data. Content of the selected structure can be retained as a part of the transform project by selecting the embed option when choosing the structure.

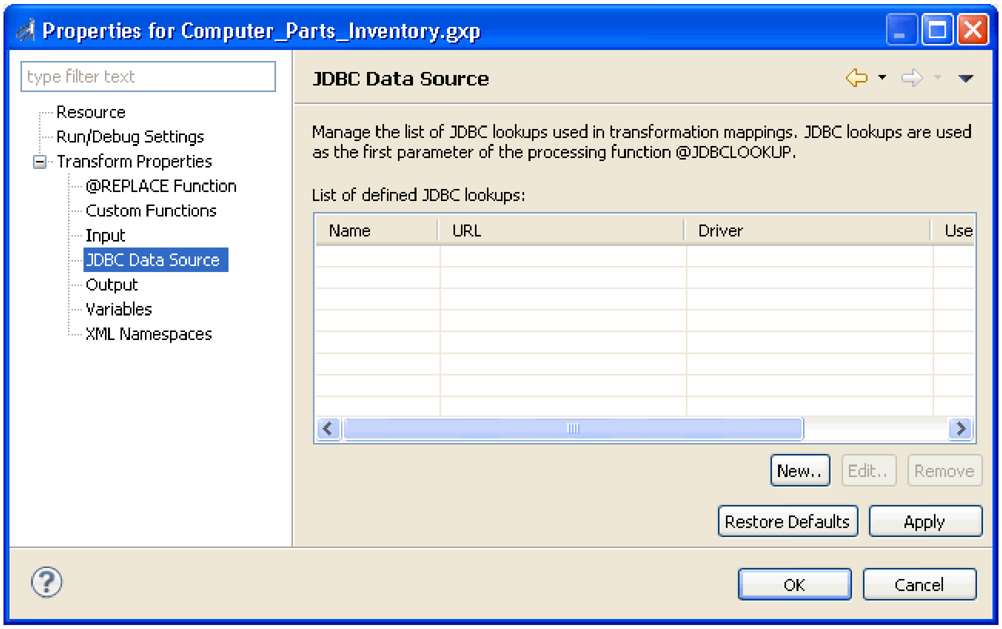
Structure File: input.xml

Contains Namespace

JDBC Data Source

The JDBC Data Source category enables you to manage the list of JDBC lookups used by the @JDBCLOOKUP function during transformation mappings. You can define multiple JDBC connections or add a connection from an existing project. In addition, you can specify any predefined function (for example, @SREG) as the URL.

The following image shows the JDBC Data Source pane.



JDBC Data Source

Manage the list of JDBC lookups used in transformation mappings. JDBC lookups are used as the first parameter of the processing function @JDBCLOOKUP.

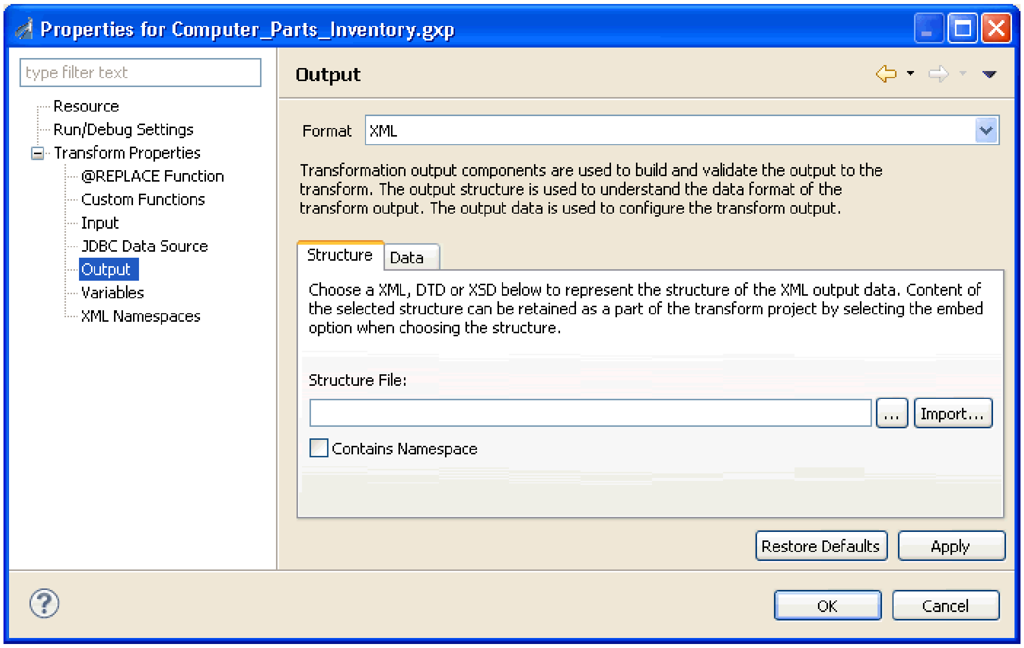
List of defined JDBC lookups:

Name	URL	Driver	Use

Output

The Output category allows you to configure the output properties for the Transform component.

The following image shows the Output pane.



Format: XML

Transformation output components are used to build and validate the output to the transform. The output structure is used to understand the data format of the transform output. The output data is used to configure the transform output.

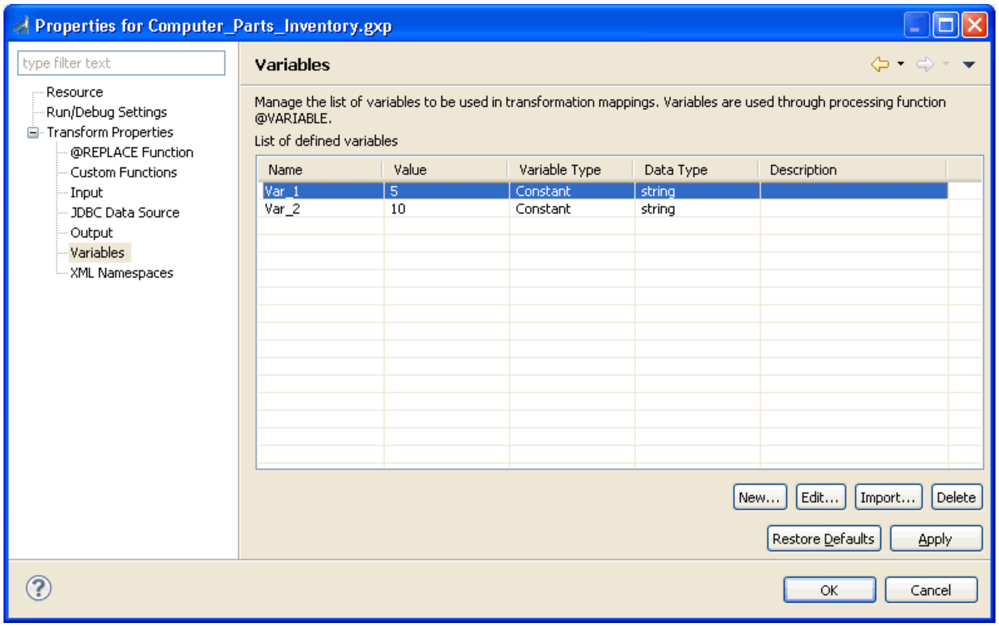
Choose a XML, DTD or XSD below to represent the structure of the XML output data. Content of the selected structure can be retained as a part of the transform project by selecting the embed option when choosing the structure.

☐ Contains Namespace

Variables

The Variables category enables you to manage the list of variables that can be used for your output node values. Using variables improves readability and usability of the output. You can use a defined variable in the output nodes mapping value through the @GETCONSTANT or @VARIABLE function.

The following image shows the Variables pane.



Manage the list of variables to be used in transformation mappings. Variables are used through processing function @VARIABLE.

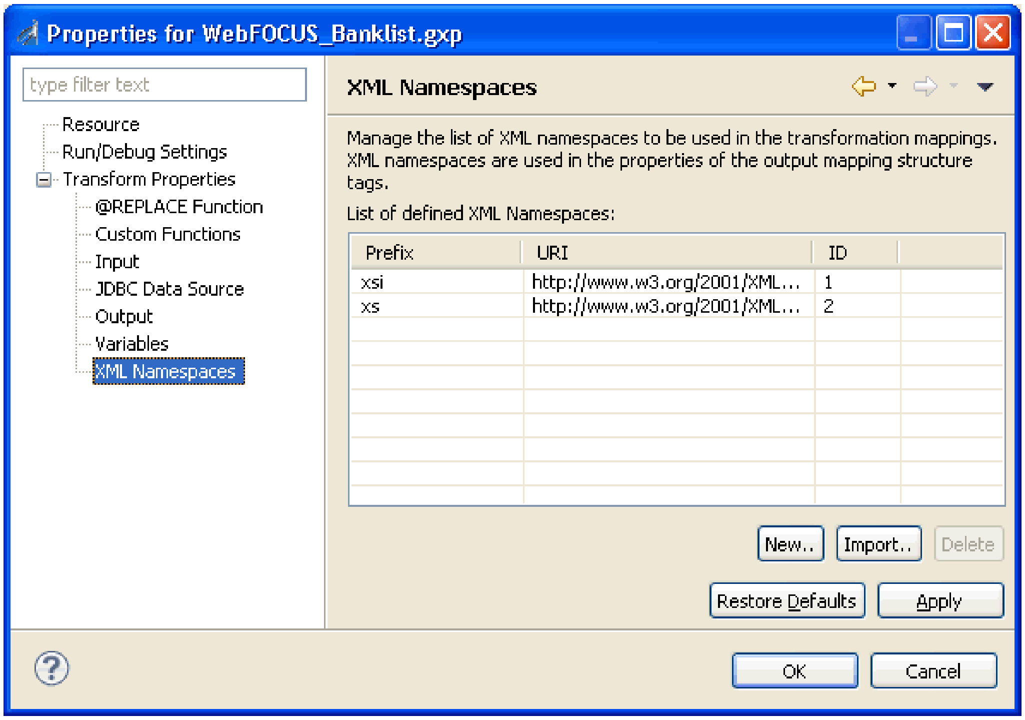
List of defined variables

Name	Value	Variable Type	Data Type	Description
Var_1	5	Constant	string	
Var_2	10	Constant	string	

XML Namespaces

The XML Namespaces category enables you to load XML namespaces from other projects or to create your own.

The following image shows the XML Namespaces pane.



Manage the list of XML namespaces to be used in the transformation mappings. XML namespaces are used in the properties of the output mapping structure tags.

List of defined XML Namespaces:

Prefix	URI	ID
xsi	http://www.w3.org/2001/XML...	1
xs	http://www.w3.org/2001/XML...	2

Dictionary Builder Tutorial

This topic provides a Dictionary Builder tutorial for iWay Transformer. Dictionary Builder is a tool within iWay Transformer, which provides a graphical interface for creating and modifying e-business metadata.

Overview

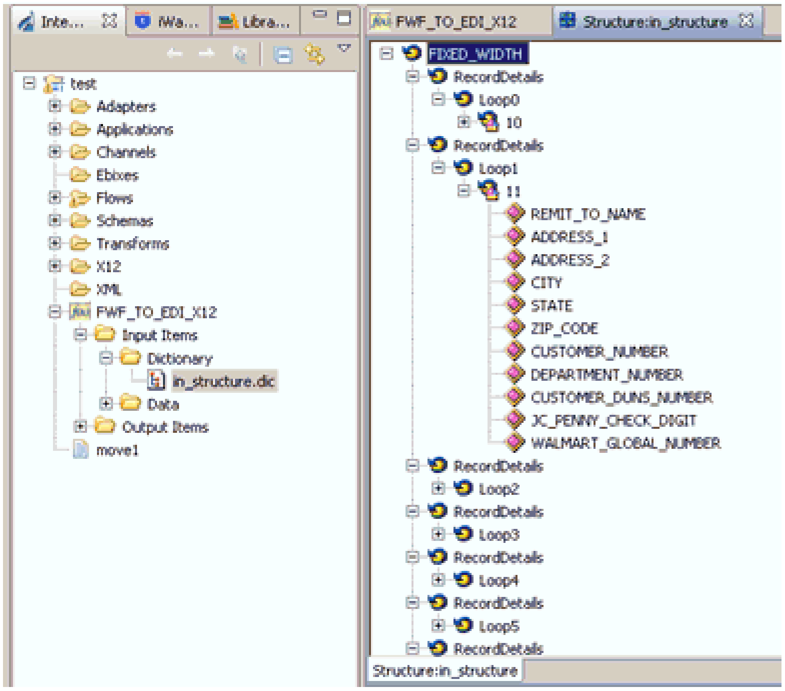
Dictionary Builder provides metadata management in EDI, SWIFT, or proprietary projects within iWay Transformer or iWay Service Manager (iSM). It allows composing, editing, and distributing e-business metadata in the form of Ebix components containing data dictionaries, and certain data dictionaries within Transform project components. Dictionary Builder is composed of two major facilities:

- Data Dictionary Editor (Metadata Management Facility)
- Ebix Entry Projects (Ebix Management Facility)

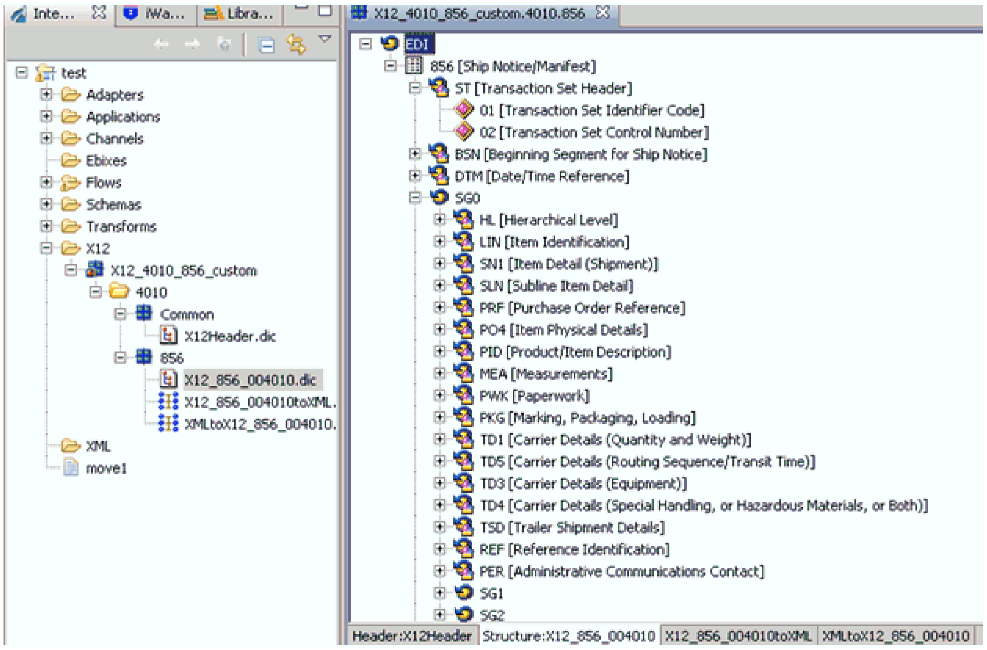
A dictionary is a representation that describes the structure of a transaction or a document that is used in EDI, SWIFT, or proprietary projects.

You can use the Dictionary Builder interface to visualize and browse, in an orderly way, all the components of e-business data dictionaries. You can access the Dictionary Builder interface in iWay Transformer in either of two ways:

- Open a Transform project (.gxp) that uses a dictionary of supported data formats for its input or output, as shown in the following image.



❑ Open an Ebix Entry project (.ebx), as shown in the following image.


Inte...
iWa...
Libra...
test
Adapters
Applications
Channels
Ebixes
Flows
Schemas
Transforms
X12
X12_4010_856_custom
4010
Common
X12Header.dic
856
X12_856_004010.dic
X12_856_004010toXML.
XMLtoX12_856_004010.
XML
move1
X12_4010_856_custom.4010.856
EDI
856 [Ship Notice/Manifest]
ST [Transaction Set Header]
01 [Transaction Set Identifier Code]
02 [Transaction Set Control Number]
BSN [Beginning Segment for Ship Notice]
DTM [Date/Time Reference]
SG0
HL [Hierarchical Level]
LIN [Item Identification]
SN1 [Item Detail (Shipment)]
SLN [Subline Item Detail]
PRF [Purchase Order Reference]
PO4 [Item Physical Details]
PID [Product/Item Description]
MEA [Measurements]
PWK [Paperwork]
PKG [Marking, Packaging, Loading]
TD1 [Carrier Details (Quantity and Weight)]
TD5 [Carrier Details (Routing Sequence/Transit Time)]
TD3 [Carrier Details (Equipment)]
TD4 [Carrier Details (Special Handling, or Hazardous Materials, or Both)]
TSD [Trailer Shipment Details]
REF [Reference Identification]
PER [Administrative Communications Contact]
SG1
SG2
Header:X12Header
Structure:X12_856_004010
X12_856_004010toXML
XMLtoX12_856_004010

Metadata Management Facility

In iWay Transformer, you can build, browse, and edit standard or custom data dictionaries through the Graphical User Interface (GUI) driven by the Metadata Management Facility. You can export or publish these dictionaries afterwards as Ebix components on an iSM server, or incorporate them as input or output structures for Transform projects. The purpose of a data dictionary is to provide a single global source for custom business requirements for processing your e-business message.

iWay Format Adapters are used to convert EDI or SWIFT formatted messages to XML and vice versa. Data dictionaries are used to define the structure of e-business messages. Data dictionaries contain structural and content information.

To ensure correct processing of the e-business message, the supporting e-business metadata must be supplied in the form of data dictionaries. Data dictionaries are stored as entries within the archives, known as Ebix components.

Currently, the Metadata Management Facility supports the following data formats:

❑ EDI X12

- EDIFACT
- Fixed Width
- SWIFT

Ebix Management Facility

The Ebix Management Facility enables you to create, update, and browse multiple Ebix entries or export them to Ebix archives. Ebix archives provide a way to efficiently store multiple dictionary components and other related metadata as one executable file. These archive files have an extension of .ebx. In this guide, they are referred to as Ebix archives. The design of Ebix archives advances the concept of iWay packages by providing seamless integration with iSM components.

In design time, you can import each of the data dictionaries and its related metadata components in iWay Transformer as one unit, called an Ebix Entry. An Ebix Entry is managed in iWay Transformer through an Ebix Entry project, a type of project that supports Dictionary Builder views. You can then export single or multiple Ebix Entry projects opened in iWay Transformer as Ebix archives or publish them on the server as Ebix components.

The Ebix Management Facility supports the following data formats:

- EDI HIPAA
- EDI X12
- EDIFACT
- HL7
- SWIFT

Navigating Dictionary Builder

This topic provides an overview of the supported iSM dictionary components and describes how to navigate these components in Dictionary Builder.

Overview

A data dictionary is an iWay representation of e-business metadata that describes the layout and grammar of a transaction or a document. The layout and grammar are stored in one or more files that contain dictionary components, such as headers or structures.

Dictionaries are required when the input or output data is not structural, for example, in EDI or SWIFT formats. Most dictionaries are converted into XML from the specifications of particular message types, which are managed and published by organizations. iWay Transformer supports the following dictionary component types:

- **Header.** Provides the envelope details for the message, such as trading partner information and message layout.
- **Structure.** Contains the layout and grammar of the document contained in the message, such as a transaction set of an EDI message.
- **Message.** Represents a structural definition of an HL7 transaction set in XML format. Messages consist of segments and segment groups in sequential order.
- **Field.** Represents a basic building block of an HL7 message. A field consists of syntax or data type.
- **Datatype.** Represents a syntax rule for a field, for example, "ASCII String".

The following table lists the component types that are supported for each data format.

Format	Header	Structure	Message	Segment	Datatype	Field
EDIFACT	Yes	Yes	No	No	No	No
EDI X12	Yes	Yes	No	No	No	No
Fixed Width	No	Yes	No	No	No	No
IDOC	No	Yes	No	No	No	No
SWIFT	No	Yes	No	No	No	No
CSV	No	Yes	No	No	No	No
XML	No	Yes	No	No	No	No

In iWay Transformer, you typically use dictionaries as a starting point for designing your mapping.

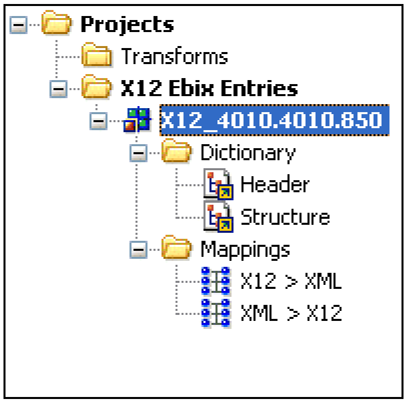
You can build, browse, and edit dictionary components of the supported formats using the Dictionary Builder interface.

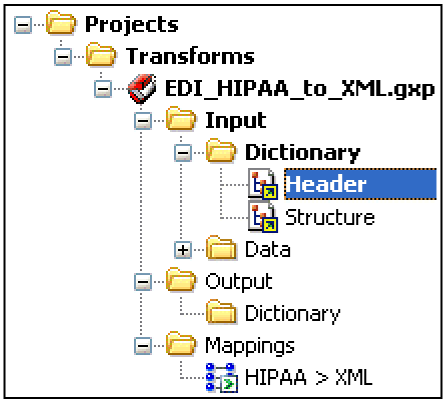
Navigating the Dictionary Builder Interface

You can access the workspace environment of the Dictionary Builder interface in three different ways:

- Select an Ebix Entry project name or any of its components in the Project Navigator pane, as shown in the following image on the left.
- Select the input or output dictionary of a Transform project, or any of the components inside (for example, header or structure), in the Project Navigator pane, as shown in the following image on the right.

  The data format of the dictionary must be supported by the Metadata Management Facility.





Located on the left side of the iWay Transformer window in the default view, the Project Navigator pane displays the references to Ebix Entry and Transform projects that are currently open. Transform projects that are currently open can be found in the Transforms category. When an Ebix Entry project is open, an Ebix Entry category is created per each data format. For example, in the preceding image, all open EDI X12 Ebix Entry projects are located in the X12 Ebix Entries subfolder.

However, if you open a SWIFT Ebix Entry, the new project category (SWIFT Ebix Entries) will appear in the Project Navigator, containing the references to your SWIFT Ebix Entry project.

The Ebix Entry project name is displayed as a compilation of the data name format being used, the format version it belongs to, and the message name that it describes. This notation provides quick and easy resource organization, which is especially useful when you are working with multiple Ebix Entry projects at once.

For example, the EDI X12 Ebix Entry project referred to in the preceding image is displayed by the Dictionary Builder interface as

```
X12_4010.4010.850
```

where:

`X12_4010`

Is the naming convention for a standard Ebix that begins with *Data FormatName_Version*.

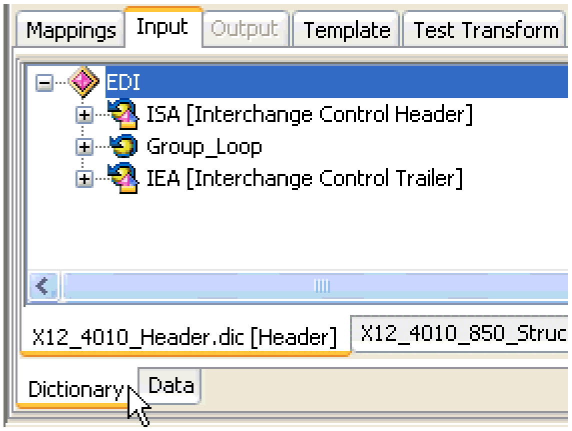
`4010`

Is the name for a standard version within that data format.

`850`

Is the message type (Purchase Order) represented by the dictionary.

❑ Navigate to the Dictionary tab for the project in the project workspace pane, as shown in the following image.

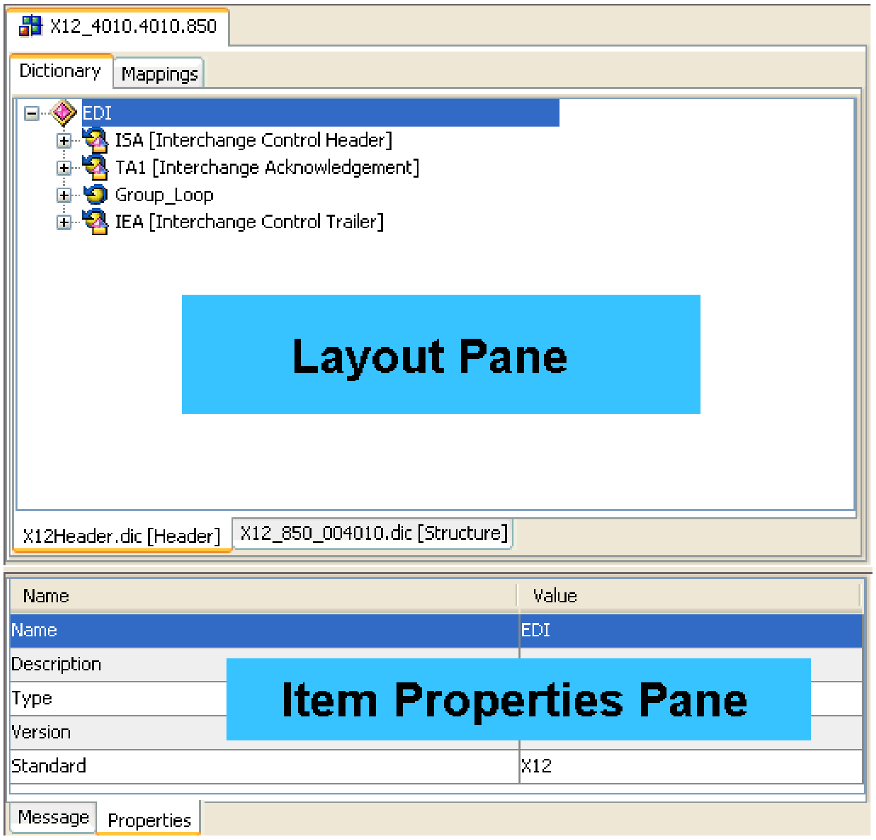

Mappings
Input
Output
Template
Test Transform
EDI
ISA [Interchange Control Header]
Group_Loop
IEA [Interchange Control Trailer]
X12_4010_Header.dic [Header]
X12_4010_850_Struc
Dictionary
Data

The Dictionary Builder view consists of two panes:

❑ Layout pane

❑ Item Properties pane

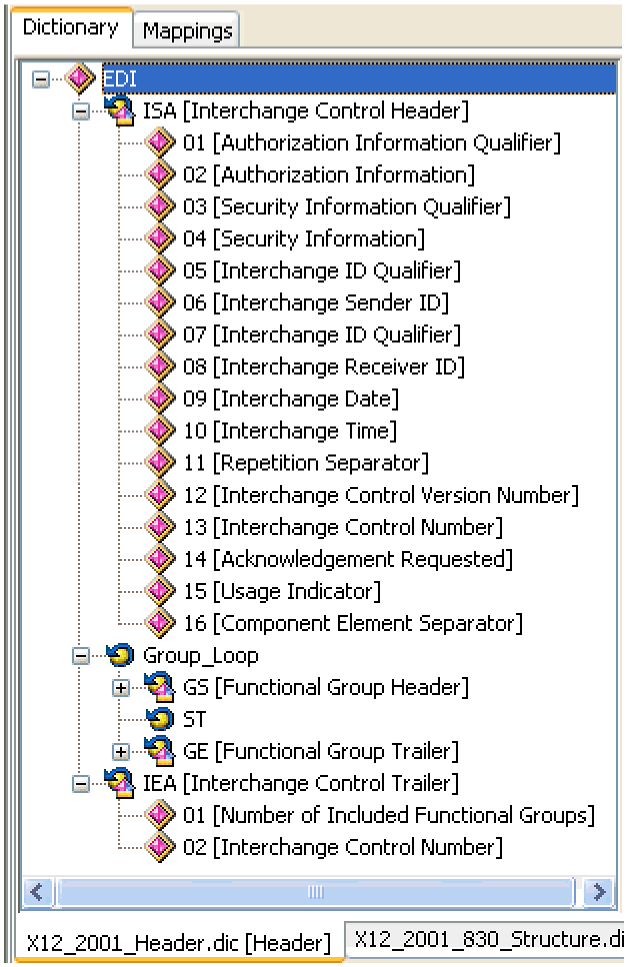
The following image shows the two panes.


X12_4010.4010.850
Dictionary
Mappings
EDI
ISA [Interchange Control Header]
TA1 [Interchange Acknowledgement]
Group_Loop
IEA [Interchange Control Trailer]
Layout Pane
X12Header.dic [Header]
X12_850_004010.dic [Structure]
Item Properties Pane
Message
Properties

Name	Value
Name	EDI
Description	
Type	
Version	
Standard	X12

Layout pane

The following image shows the Layout pane.



EDI
ISA [Interchange Control Header]
01 [Authorization Information Qualifier]
02 [Authorization Information]
03 [Security Information Qualifier]
04 [Security Information]
05 [Interchange ID Qualifier]
06 [Interchange Sender ID]
07 [Interchange ID Qualifier]
08 [Interchange Receiver ID]
09 [Interchange Date]
10 [Interchange Time]
11 [Repetition Separator]
12 [Interchange Control Version Number]
13 [Interchange Control Number]
14 [Acknowledgement Requested]
15 [Usage Indicator]
16 [Component Element Separator]
Group_Loop
GS [Functional Group Header]
ST
GE [Functional Group Trailer]
IEA [Interchange Control Trailer]
01 [Number of Included Functional Groups]
02 [Interchange Control Number]

The Layout pane is the core working area used by the Dictionary Builder interface. Each dictionary component is displayed on its own tab, such as the Header or Structure. Dictionary components are graphically presented in iWay Transformer as tree structures of the lesser structure units that make up the dictionary, such as segments and loops.

Item descriptions are provided in square brackets next to each EDI dictionary structure segment. For example, the ISA segment node in the preceding image (EDI X12 Version 2001 Message Type 830) includes the following description:

```
[Interchange Control Header]
```

Similarly, the 01 element node includes the following description:

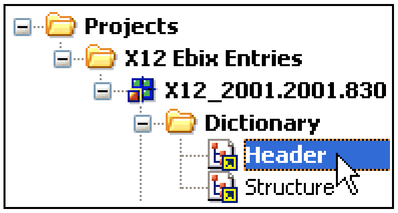
```
[Authorization Information Qualifier]
```

These descriptions allow you to map and identify EDI segments more efficiently.

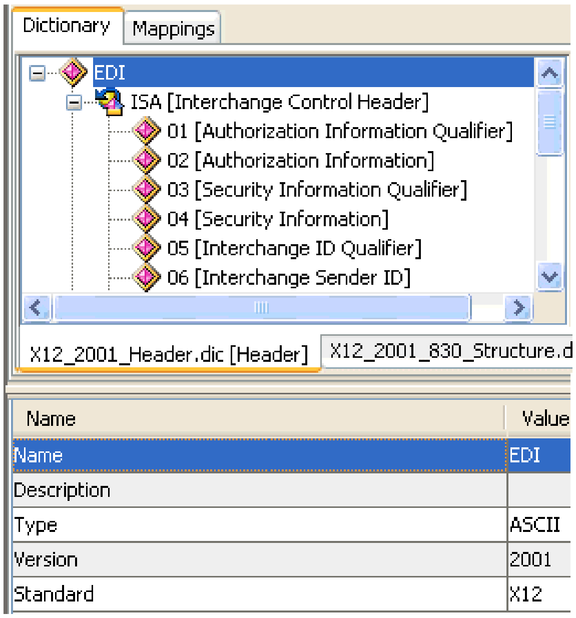
To quickly delete a dictionary component from the Layout pane, select the item and press the Delete key. When the Confirm dialog box opens, press the Enter key, or click *Yes*.

The tree structure of the dictionary metadata on the Layout pane is easy to navigate. The manipulation that is performed on the data on the Layout pane drives the editing and processing of the dictionary data in the Item Properties pane. The Item Properties pane displays the set of properties that belong to the item highlighted in the Layout pane.

If the dictionary has more than one component (for example, it is an EDI X12 dictionary that consists of header and structure components), you can toggle between the tabs that are labeled by component name (for example, Header and Structure). The tabs are located at the bottom of the Layout pane. You can also select the components from the Dictionary category in the Project Navigator pane, as shown in the following image.



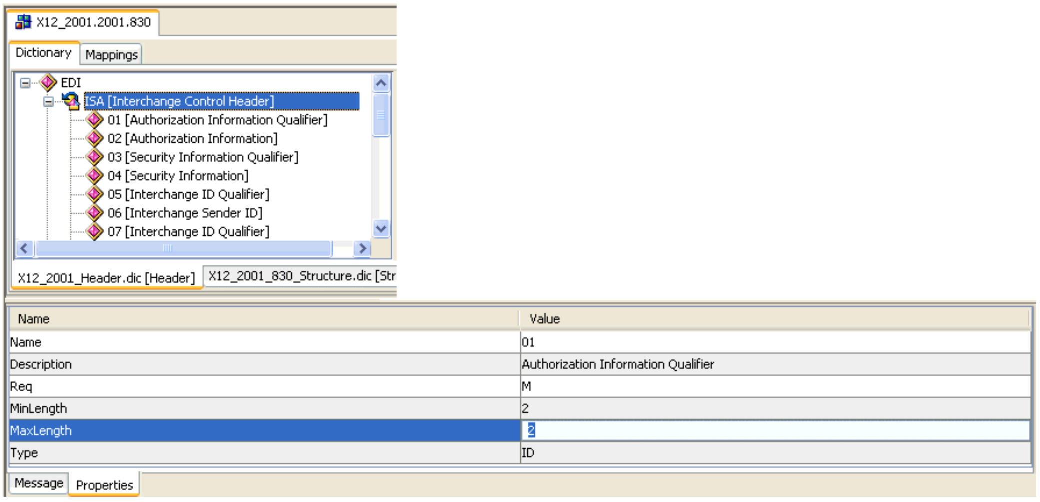
The data format type, such as EDI, typically serves as the root of the component structure. You can use this feature to look up the standard properties for this project from the Item Properties pane when the root element of the dictionary component tree is highlighted on the Layout pane, as shown in the following image.


Dictionary
Mappings
EDI
ISA [Interchange Control Header]
01 [Authorization Information Qualifier]
02 [Authorization Information]
03 [Security Information Qualifier]
04 [Security Information]
05 [Interchange ID Qualifier]
06 [Interchange Sender ID]
X12_2001_Header.dic [Header]
X12_2001_830_Structure.d

Name	Value
Name	EDI
Description	
Type	ASCII
Version	2001
Standard	X12

Item Properties pane

The following image shows the Item Properties pane.


X12_2001.2001.830
Dictionary
Mappings
EDI
ISA [Interchange Control Header]
01 [Authorization Information Qualifier]
02 [Authorization Information]
03 [Security Information Qualifier]
04 [Security Information]
05 [Interchange ID Qualifier]
06 [Interchange Sender ID]
07 [Interchange ID Qualifier]
X12_2001_Header.dic [Header]
X12_2001_830_Structure.dic [Str
Message
Properties

Name	Value
Name	01
Description	Authorization Information Qualifier
Req	M
MinLength	2
MaxLength	2
Type	ID

The default location of the Item Properties pane is at the bottom of the Dictionary Builder workspace. It displays the set of properties for the structural item that is currently highlighted in the Layout pane. You can edit item properties by double-clicking the value of the property, as shown in the preceding image.

Chapter 3

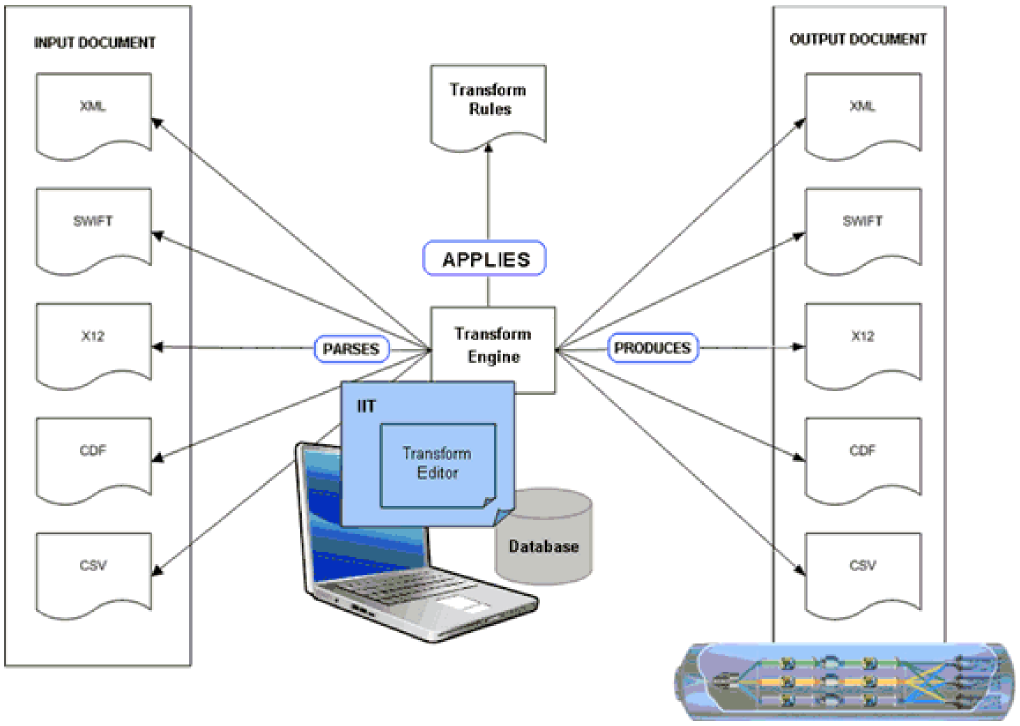
iWay Transformer Concepts

This section describes concepts related to iWay Transformer, such as transformation and mappings.

In this chapter:

Transformation Process

Transformation is the means by which an incoming document is converted from one data format or structure to another.



Transform Component

A Transform component represents one configured transformation. It contains information on the dependencies and the rules of a particular transform. The iWay Transformation Engine processes input data and builds the output data according to these rules. The serialized form of a Transform component is called a Transform template.

Transform Template

A Transform template contains a blueprint of a transformation. It provides an XML representation of a Transform component, referencing its various parameters and rules. It is typically distinguished by a .gxp extension file format.

Configuration of a Transform Component

The main modules within a Transform component are Dictionary and Mapping rules.

Dictionary

A data dictionary is an iWay representation of e-business metadata that describes the layout and grammar of a transaction or a document. The layout and grammar are stored in one or more files that contain dictionary components, such as headers or structures.

Dictionaries are required when the input or output data is not structural, for example, in EDI or SWIFT formats. Most dictionaries are converted to XML from the specifications of particular message types, which are managed and published by organizations. iWay Transformer supports the following dictionary component types:

- **Header.** Provides the envelope details for the message, such as trading partner information and message layout.
- **Structure.** Contains the layout and grammar of the document contained in the message, such as a transaction set of an EDI message.
- **Message.** Represents a structural definition of an HL7 transaction set in XML format. Messages consist of segments and segment groups in sequential order.
- **Segment.** Represents a business entity, for example, Patient Identifier, in an HL7 message. Segments consist of fields in sequential order, and are contained within the message component.
- **Field.** Represents a basic building block of an HL7 message. A field consists of syntax or data type.
- **Datatype.** Represents a syntax rule for a field, for example, "ASCII String".

The following table lists the component types that are supported for each data format.

Format	**Header**	**Structure**	**Message**	**Segment**	**Datatype**	**Field**
EDIFACT	Yes	Yes	No	No	No	No
EDI HIPAA	Yes	Yes	No	No	No	No
EDI X12	Yes	Yes	No	No	No	No
Fixed Width	No	Yes	No	No	No	No

Format	Header	Structure	Message	Segment	Datatype	Field
IDOC	No	Yes	No	No	No	No
SWIFT	No	Yes	No	No	No	No
HL7	No	No	Yes	Yes	Yes	Yes
CSV	No	Yes	No	No	No	No
XML	No	Yes	No	No	No	No

You can edit the majority of non-XML Transformer dictionaries using the Dictionary Builder.

In iWay Transformer, dictionaries are typically used as a starting point for designing your mappings.

iWay Transformer Mappings

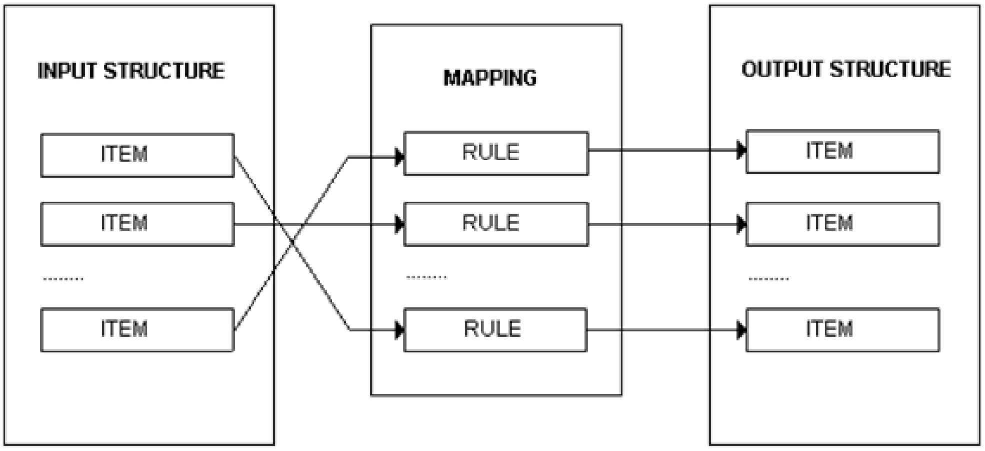
Mappings consist of rules that define how a transformation is performed for a project, based on an input and output configuration. They are managed using Mapping Builder.

Mapping Rules

Mapping rules define a set of requirements performed in the course of a transformation, based on particular input and output configurations. A mapping rule specifies how records and fields of an output document structure are calculated in relation to the input structure. Each mapping rule:

- Points to a single node of an output structure.
- Contains a formula for obtaining data for a value of an output node.
- Can map one or more of the input structure items to an output structure item, depending on the type of the mapping value used.

The following diagram shows the mapping rule process.



Mapping Structure

A mapping structure represents a structure (layout) of a document. It consists of interconnected nodes. Each mapping node has:

- A mapping type. For more information, see *Mapping Types* on page 236.
- A mapping value. For more information, see *Mapping Values* on page 256.
- A specific place in the structure that defines its mapping relationships.

For more information, see *Working With the Mapping Builder* on page 224.

iWay Transformer Supported Data Formats

Transform components support the following data formats:

- CSV (Comma-separated Values).
- EDI (Electronic Data Interchange).
- EDI HIPAA.
- EDI X12.
- EDIFACT.
- Fixed Width.
- HTML.

- ❏ IDOC (SAP Intermediate Document).
- ❏ SWIFT (International Financial).
- ❏ XML.
- ❏ iWay XML Response.
- ❏ iWay XML Embedded Request.
- ❏ iWay XML Request.
- ❏ CDF (Column-defined Format). CDF is deprecated. iWay Software recommends that you use Fixed Width.

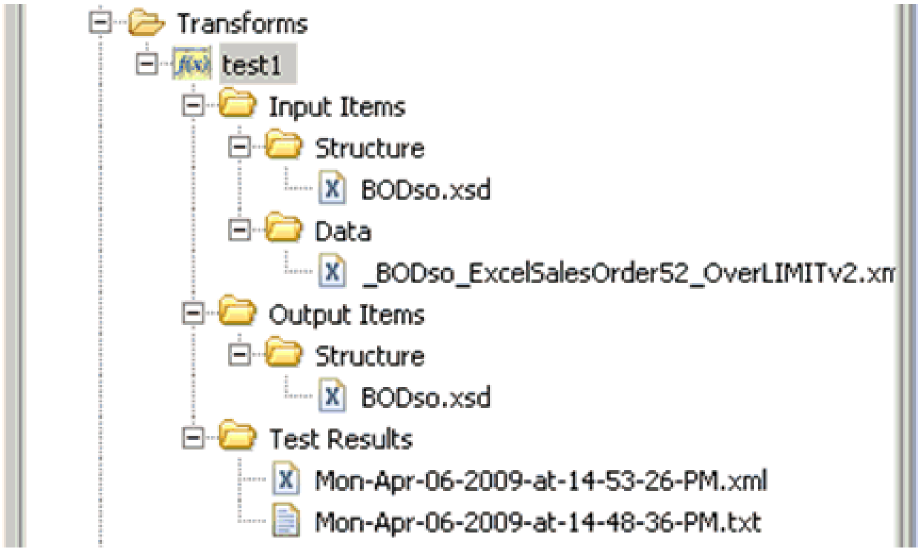
Transform Component Dependencies

There are a number of optional and required dependencies that you must define for a Transform component to function properly. You can view and browse a project dependencies using Integration Explorer.

A Transform component resides under the Transforms folder. It is identified by its name and yellow f(x) icon. Typically, it contains a folder for Input Items and a folder for Output Items, each containing respective dependencies.

The Input Structure file is required for the initiation of your project. The Output Structure file is optional, depending on the specific requirements of the transformation. The Input Data file is required when you wish to test the Transform component. You need a valid input document to undergo the transformation.

After you execute the test run, the Test Results folder containing the test data is displayed with the dependencies, as shown in the following image.


Transforms
test1
Input Items
Structure
BODso.xsd
Data
_BODso_ExcelSalesOrder52_OverLIMITv2.xm
Output Items
Structure
BODso.xsd
Test Results
Mon-Apr-06-2009-at-14-53-26-PM.xml
Mon-Apr-06-2009-at-14-48-36-PM.txt

Using Namespaces in iWay Transformer

An XML namespace is a collection of element types and attribute names. XML namespaces provide a way to distinguish between duplicate element types and attribute names. A duplication may occur, for example, in an XSLT stylesheet or in a document that contains element types and attributes from two different Document Type Definitions (DTDs). In an XML namespace, an element type or attribute name is uniquely identified by a two-part name property: the name of its XML namespace and its local name.

You can configure XML namespaces globally or separately for each node. An individual XML namespace configuration enables you to accomplish more advanced goals based on your requirements.

Default namespaces are supported through the overwrite Default Namespace option.

iWay Transformer also supports the mapping of multiple namespaces to the same URI and the application of a namespace prefix to a subtree.

The following example presents XML namespaces as a convenient solution to distinguish between two different XML element types named Address.

```
<Department>
<Name>DVS1</Name>
<addr:Address xmlns:addr="http://www.tu-darmstadt.de/ito/addresses">
<addr:Street>Wilhelminenstr. 7</addr:Street> <addr:City>Darmstadt</
addr:City>
<addr:State>Hessen</addr:State>
<addr:Country>Germany</addr:Country>
<addr:PostalCode>D-64285</addr:PostalCode>
</addr:Address >
<serv:Server xmlns:serv="http://www.tu-darmstadt.de/ito/servers">
<serv:Name>OurWebServer</serv:Name> <serv:Address>123.45.67.8</
serv:Address>
</serv:Server>
</Department>
```

Note the first Address element type. Its name belongs to the http://www.tu-darmstadt.de/ito/addresses XML namespace. It has a universal (two-part) name of “http://www.tu-darmstadt.de/ito/addresses” and “Address”. The second element type with the same first name “Address” belongs to the http://www.tu-darmstadt.de/ito/servers XML namespace. It has a universal name of {http://www.tu-darmstadt.de/ito/servers}Address. Thus, each universal name is preserved as unique, meeting the requirement that each element type in an XML document needs to have a unique name.

You can also configure your Transform component to use XML namespaces by loading a set of namespaces from an existing Transform component. Alternatively, you can opt to create your own set. Either way, ensure that you select the *Contains Namespace* check box when configuring the input.

In addition, ensure that all namespaces from your input data file are described in your XML or schema structure, and that the prefixes are consistent (that is, they contain the same URI).

To better understand namespace mapping rules, you can draw the parallel between namespace mapping in iWay Transformer and long distance telephone calling using your land line.

As an example, imagine that you need to phone your aunt, who currently resides in France, from New York, USA. In order to complete the call to your aunt, your long distance calling plan must first have the country code for France on the calling plan list of permitted country codes. Otherwise, the call to France will not be completed.

In the same manner, the namespaces from the input data file can be compared to a list of country codes to be dialed. You cannot map the namespace from your input data file successfully, unless it is included in your schema or XML structure namespace list, which in this example, is similar to a long distance calling plan.

While designing projects with namespaces, check the schema and input data for consistency. Your mapping values will not appear in the output if the same namespace prefix from the incoming data has a different URI in your structure file.

The namespaces with different prefixes that have the same URI are treated as the same namespace, according to the rules outlined in XML Namespace Specifications.

For example, the following namespaces are described in your dictionary:

```
<a1:getMessages xmlns:a1="uri:wsdl:org.iway.sp2.customer.v1"
xmlns:internal="uri:wsdl:org.iway.sp2.internal.v1">
.
.
.
```

In addition, your input data has the following namespaces:

```
<customer:getMessages xmlns:customer="uri:wsdl:org.iway.sp2.customer.v1"
xmlns:internal="uri:wsdl:org.iway.sp2.internal.v1">
.
.
.
```

The namespaces *customer* and *a1* are treated as the same namespace while mapping the output values.

If you are dealing with dynamic namespaces for incoming data, for which their URIs cannot be determined during design time, we recommend that you deselect the *Contains Namespace* check box. This will ensure that incoming data values appear in the output. For more information, see *Working With Namespaces* on page 265.

Transform Functions

Functions provide various algorithms that calculate output mapping values in iWay Transformer.

iWay Transformer provides numerous predefined functions in the following major categories:

- **EDI Functions**. For more information, see *EDI Functions* on page 102.
- **Numerical Functions**. For more information, see *Numerical Functions* on page 103.
- **Processing Functions**. For more information, see *Processing Functions* on page 112.
- **Run-Time Functions**. For more information, see *Run-Time Functions* on page 118.
- **Security Functions**. For more information, see *Security Functions* on page 120.
- **String Functions**. For more information, see *String Functions* on page 121.

- **Time Functions**. For more information, see *Time Functions* on page 134.

Predefined Functions

To find more information about a function or its parameters, navigate to an appropriate function category within this topic. Functions and categories are arranged in alphabetical order.

EDI Functions

The EDI functions available in iWay Transformer are described in the following table.

Format-specific functions, such as EDI, do not appear in the Functions pane of the Mapping Builder unless the output is using the exact format.

EDI Function	Description
@COUNT_GROUP()	Returns the occurrences of the group within an EDI message. **Parameters:** None.
@COUNT_SEGMENT()	Returns the occurrences of the segment within a transaction set. **Parameters:** None.
@COUNT_SEGMENT (*param1*)	Returns the occurrences of the specified segment within a transaction set. **Parameter:** *param1*. Name of the segment. It must be a constant.
@COUNT_TRANSACTION()	Returns the occurrences of the transaction within a group. **Parameters:** None.
@CRC16()	Is specific to the UCS 4010 894 document. It is used on the 866 element (CRC 16 checksum) of the G8501 segment. The value of this function is generated using a Cyclic Redundancy Code (CRC) algorithm. It has a fixed length of four characters with no zero suppression. It applies to the contents of the entire transaction that is configured for segments ST through G86, inclusive.

EDI Function	Description
@LINE_COUNTER (*param1*)	Returns the occurrences of the specified segment within the same loop instance. The counter is reset when a new loop starts. **Parameter:** *param1*. Name of the segment. It must be a constant. It is based on the LN segment description.

Numerical Functions

The numerical functions available in iWay Transformer are described in the following table.

Numerical Function	Description
@ADD (*param1*, *param2*)	Returns the result of adding the two numbers specified as parameters. **Parameters:** *param1*. Number to add to *param2*. *param2*. Number to add to *param1*. **Example:** @ADD(24.01, 12.02) returns 36.03.
@AVERAGE (*param1*)	Returns the average of the specified parameter values within the same parent instance. This function is used within the group in combination with the Agg looping property. You must carefully set the looping settings of the parent and grandparent of the attribute or element that uses the @AVERAGE function. For more information, see *Group Properties* on page 241. **Parameter:** *param1*. Number that represents the value to average. Usually, it is mapped from the ancestor node to the input document node.

Numerical Function	Description

	Example:
	In the following example, the output node ValueAverage has the value @AVERAGE(Sales/Company/Year/Quarter/Product/Item@value), where the output obtained is:
	`<Sales_Totals>`
	`  <companyName>Video and Sound Card Express</companyName>`
	`  <Statistics>`
	`     <itemValueAverage>139.7407</itemValueAverage>`
	`     <allTimeSales>95022.02</allTimeSales>`
	`  </Statistics>`
	`</Sales_Totals>`
	In this XML-to-XML transformation example, the Sales_Totals output group has the looping property set to False. The Statistics group has the looping property set to Agg. The desired output value is also obtained with the Sales_Totals looping property set to Agg, but not when it is set to True.

@CHKNUM (*param1*)	Returns the string true if the specified parameter is a valid number. Use this function to determine if the parameter is a number (either integer or decimal). Returns true or false. **Parameter:** *param1*. Number to validate. **Examples:** @CHKNUM('1.1') returns true. @CHKNUM('abcd') or @CHKNUM('1,234.55') returns false.
@COUNT	Counts the input items processed and returns the next sequential value. The first loop through @COUNT initializes the counter to 1. Subsequent passes increment the counter to the next integer number. **Parameters:** None.

Numerical Function	Description
@DIVIDE (*param1*, *param2*)	Returns the decimal result of dividing two numbers specified as parameters. **Parameters:** *param1*. Number that represents the dividend. *param2*. Number that represents the divisor. **Example:** @DIVIDE('24.02', '12.01') returns 2.0.
@INT (*param1*)	Returns the first occurrence of the integer value found in the specified parameter. If the integer value cannot be found, the number 0 (zero) is returned. **Parameter:** *param1*. String to be checked. **Examples:** @INT('45.60 or 65.60') returns 45. @INT('in the summer of 1998 and 1999') returns 1998. @INT('time is 23:11:56') returns 23. @INT('last year') returns 0.
@INTVAL (*param1*)	Returns the specified parameter value if it is a valid integer. Otherwise, it returns the number 0 (zero). **Parameter:** *param1*. String to be checked. **Examples:** @INTVAL('45') returns 45. @INTVAL('45.12') or @INTVAL('dollars 1.23') returns 0.

Numerical Function	**Description**
@MULTIPLY (*param1*, *param2*)	Returns the decimal result of multiplying two members specified as parameters. **Parameters:** *param1*. Number to multiply by *param2*. *param2*. Number to multiply by *param1*. **Example:** @MULT(12.01, 2.00) returns 24.02.
@NUM (*param1*)	Converts an integer string to a numeric value. Use this function to determine if a parameter is an integer (whole number). The function returns the parameter or an error. **Parameter:** *param1*. Numeric string to be converted to an integer value. **Example:** @NUM('45') returns 45.
@NUM_CHR (*param1*)	Returns the ASCII character that corresponds to a number. **Parameter:** *param1*. Integer number between 1 and 127 that represents the order of an ASCII character. **Example:** @NUM_CHR ('66') returns 'B'.
@RANDOM (*param1*)	Returns a pseudo random number. **Parameter:** *param1*. Number used to seed the random number generator. If *param1* = -1, the seed is initialized randomly. If *param1* = *n* (where *n* is not 0 or -1), the seed is initialized to *n*. If *param1* = 0, the same random number value that is generated for the previous output item is used as the result for the next output item. For example, if the @RANDOM value of an output element is '-12345', and you use @RANDOM('0') for the next element, the same '-12345' value is returned for this element.

Numerical Function	Description
@RANGE (*param1*, *param2*, *param3*)	Determines if a number falls within a range, and returns the string value "true" or "false". The range is given by *param2* to *param3*, inclusive. **Parameters:** *param1*. Value that is checked. *param2*. Value that represents the lower limit of the range. *param3*. Value that represents the upper limit of the range. **Example:** @RANGE('10', '5', '15') evaluates to true.
@ROUND (*param1*, *param2*, *param3*)	Extracts a specified part of a number and rounds the result to an integer. **Parameters:** *param1*. Number subjected to the operation. *param2*. Number of digits to be extracted from the integer part of *param1*. Digits are counted from the left of the decimal separator. *param3*. Number of digits to be extracted from the decimal part of *param1*. Digits are counted from the right of the decimal separator. **Example:** @ROUND(345.995, 2, 2) returns 46.00.

Numerical Function	**Description**
@SCALE_ROUND (*param1*, *param2*, *param3*)	Returns the specified parameter rounded in a specified fashion. Enables you to scale a number and specify the way that it is rounded during scaling. **Parameters:** *param1*. Number subjected to the operation. *param2*. Number of digits to the right of the decimal place to keep. The value must be a non-negative integer. *param3*. Type of rounding required. The following are possible values: ROUND_CEILING: Round towards positive infinity. ROUND_DOWN: Round towards zero. ROUND_FLOOR: Round towards negative infinity. ROUND_HALF_DOWN: Round towards nearest number. If equidistant to two numbers, round down. ROUND_HALF_EVEN: Round towards nearest number. If equidistant to two numbers, round towards even number. ROUND_HALF_UP: Round towards nearest number. If equidistant to two numbers, round up. ROUND_UNNECESSARY: No rounding is necessary. Use this rounding mode to make sure that the output value has the exact number of decimals specified in *param2*. If the number of decimals is different, an error is returned. ROUND_UP: Round away from zero. **Example:** @SCALE_ROUND(5.111, 2, ROUND_UP) returns 5.12.

Numerical Function	**Description**
@STR (*param1*, *param2*)	Converts a number to an alpha string according to a picture mask. Use this function to format numbers. **Parameters:** *param1*. Number to be converted into an alpha string. *param2*. Picture mask format for the string. For more information, see *Numeric Pictures* on page 140. **Examples:** @STR('45.12', '##.#') returns '45.1'. @STR('75','#.00') returns 75.00. @STR('567.1', '#.00') returns 567.10. Since a picture mask is used, the function may cause rounding. For example, @STR(39.999,'#.00') returns '40.00'.
@SUBTRACT (*param1*, *param2*)	Returns the result of subtracting two numbers specified as parameters. **Parameters:** *param1*. Float number from which to subtract *param2*. *param2*. Float number to subtract from *param1*. **Example:** @SUBTRACT(24.02, 11.01) returns 13.01.

Numerical Function	Description

@SUM (*param1*)	Returns the sum of the numeric values of a specified input node.
	
	You must carefully set the looping settings of the parent and grandparent of the node that uses the @SUM function. For more information, see *Group Properties* on page 241.
	
	Parameter: *param1*. Input node that has a numeric value.
	
	Example: In the following example, the output node allTimeSales has the value @SUM(@MULTIPLY(Sales/Company/Year/Quarter/Product/Item @value, Sales/Company/Year/Quarter/Product/Item@sold)), where the output obtained is:
	
	`<Sales_Totals>`
	`<companyName>Video and Sound Card Express</companyName>`
	`  <Statistics>`
	`     <itemValueAverage>139.7407</itemValueAverage>`
	`     <allTimeSales>95022.02</allTimeSales>`
	`  </Statistics>`
	`</Sales_Totals>`
	
	In this XML-to-XML transformation example, the Sales_Totals output group has the looping property set to False. The Statistics group has the looping property set to Agg. The desired output value is also obtained with the Sales_Totals looping property set to Agg, but not when it is set to True.

Numerical Function	Description
@VAL (*param1*, *param2*)	Returns the number that matches the specified picture mask. This function returns a value only if the picture mask matches the input format. For example, if the picture mask is # and the input is 7.3, the function returns an error, not 7 as might be expected. Use this function to retrieve the numeric value from a string. Do not use this function to format a number. To format a number, use the @STR function. **Parameters:** *param1*. String that contains the number to be retrieved. *param2*. Format in which the number is stored in the string. Must be specified as a constant in the Mapping Builder. For more information, see *Numeric Pictures* on page 140. **Examples:** @VAL('30.11 dollars plus a fee of 40 dollars','##') returns 40. @VAL('30.11 dollars plus a fee of 40 dollars','##.##') returns 30.11. To format a number with commas, such as <X>123,456,789</X>, you can remove the commas by using the @CONCAT and @SUBSTR functions. For example: @CONCAT(@SUBSTR(X,'1','3'),@SUBSTR(X,'5','3'),@SUBSTR(X,'9','3'))

Processing Functions

The processing functions available in iWay Transformer are described in the following table.

Processing Function	Description

@CONDITION
 	
(*param1*)
	The @CONDITION function is a new processing function that supports embedded conditions. This function also includes support for AND / OR operators.
	
	Note: The @CONDITION function is designed to be used within the @IF function to support a more refined handling of multiple conditions instead of a series of nested @IF statements.
	
	Parameters:
	
	Param1: A boolean expression of a defined syntax (<*left_operand*><*operator*><*right_operand*>). The *left_operand* and *right_operand* can be any supported mapped values. Supported arguments for the *operator* include all arguments of the current @IF processing function (for example: ==, !=, <, >, <=, >=, AND, OR). By default, the *operator* is set to AND in the Output Node Mapping Builder.
	
	For AND or OR operators, if the left or right operands are mapped to an input context, it will be assumed that the @EXISTS function is used. For example:
	
	`@CONDITION(a/b/c AND a/b/e)`
	`@IF(a/b/c AND a/b/e,'true','false')`
	
	Will be the same as:
	
	`@CONDITION(@EXISTS(a/b/c) AND @EXISTS(a/b/e))`
	`@IF (@EXISTS(a/b/c) AND @EXISTS(a/b/`
	`e) ,'true','false')`
	
	The @CONDITION function returns a string:
	
	❑ true - If condition results to true, Boolean.TRUE is returned.
	
	❑ false - If condition results to false, Boolean.FALSE is returned.

Processing Function	Description
@EDIT (*param1*, *param2*, *param3*)	Returns a value formatted using the specified picture mask and formatter class name. For more information, see *Numeric Pictures* on page 140. **Parameters:** *param1*. String representing the formatter class name. Supports the DecimalFormat class. *param2*. String representing the picture mask to be applied to *param3*. *param3*. Number to be formatted. **Examples:** @EDIT('DecimalFormat','#,###.##','123456.78) returns 123,456.78. @EDIT('DecimalFormat','000','12') returns 012.

Processing Function	**Description**

@EXISTS *(context_to_be_checked)*	Returns a string with a Boolean value that indicates whether or not the context is null.
	
	Use this processing function to distinguish between an element that is missing from a document structure from an element that is present, but has a value of an empty string.
	
	Examples:
	
	`<Statistics>`
	`   <itemValueAverage>139.7407</itemValueAverage>`
	`   <allTimeSales>95022.02</allTimeSales>`
	`</Statistics>`
	
	@EXISTS (Statistics/ itemValueAverage) returns true.
	
	`<Statistics>`
	`   <itemValueAverage></itemValueAverage>`
	`   <allTimeSales>95022.02</allTimeSales>`
	`</Statistics>`
	
	@EXISTS (Statistics/ itemValueAverage) returns true.
	
	`<Statistics>`
	`   <allTimeSales>95022.02</allTimeSales>`
	`</Statistics>`
	
	@EXISTS(Statistics/ itemValueAverage) returns false.

@GETCONSTANT (*param1*)	Returns the value of the global constant specified by name in the parameter. **Parameter:** *param1*. Global constant defined in the Global Constant section of the Project Properties pane. **Example:** Assume that you defined the following global constant in project properties: Name: COMPANY_ADDRESS Value: 1234 1st Avenue As a result, @GETCONSTANT('COMPANY_ADDRESS') returns '1234 1st Avenue'.

Processing Function	Description
@IF (*param1*, *param2*, *param3*, *param4*, *param5*)	Allows for a conditional selection statement defined by the first three parameters. If the condition evaluates to true, it returns the contents of the fourth parameter. Otherwise, it returns the contents of the fifth parameter. **Parameters:** *param1*. Left operand of the condition statement. *param2*. Conditional operator. The possible arguments are: ❑ = = equal to ❑ != not equal to ❑ >= equal to or greater than ❑ <= equal to or less than ❑ > greater than ❑ < less than *param3*. Right operand of the condition statement. *param4* (true_option). The string to be returned if the condition statement is true. *param5* (false_option). The string to be returned if the condition statement is false. **Example:** The variable X has a value that fluctuates between 4 and 8. @IF(X<'10', 'ABC', 'DEF') always returns 'ABC'.
@INPUT_CONTENT	Returns the contents of the entire input data file as a string. **Parameters:** None.

Processing Function	**Description**

@JDBCLOOKUP (*param1*, *param2*)	Returns a value retrieved from a database using an SQL statement. The SQL statement can be dynamically based on the input from other transform functions. If more than one value is retrieved by the SQL statement, the last value in the sequence is returned.
	
	Parameters:
	
	param1. String that represents the name of a globally defined JDBC connection configuration.
	
	param2. SQL statement that is defined using SQL Builder.
	
	Example:
	
	`@JDBCLOOKUP('LOOKUP_TEST', {'SELECT field1 FROM LOOKUP_TABLE WHERE field2 ' = '+@QUOTE(Customer/Person/Name)})`
	
	where:
	
	`LOOKUP_TEST`
	
	Is the name of the specific JDBC connection configuration.
	
	`{'SELECT field1 FROM LOOKUP_TABLEWHERE field2 ' = '+@QUOTE(Customer/Person/Name)}`
	
	Is the SQL statement that is constructed dynamically.

@NULL	Returns a null output, that is, no output. Useful, for example, as either the true_option or false_option in an @IF function. **Parameters:** None.

Processing Function	Description
@REPLACE (*param1*, *param2*)	Calls the predefined replace function specified by name in the second parameter of @REPLACE, which replaces every matched string specified in the first parameter. If the match is not found, the first parameter is returned. For more information on how to define replace functions, see *Using the Mapping Builder* on page 269. **Parameters:** *param1*. Input node, value, or constant in which to make changes. *param2*. Replace function name as defined. The @REPLACE processing function supports spaces and unprintable characters (for example, \t, \r, or \n) as parameter values. **Example:** @REPLACE('11009333009', 'REPLACE_009_WITH_Add') returns 11Add333Add.
@SIMPLE_REPLACE (*param1*, *param2*)	Calls the predefined replace function, which replaces the first string that matches the value of the first parameter of this function. If a match is not found, the first parameter is returned. **Parameters:** *param1*. Input node value in which to make changes. *param2*. Replace function name as defined. **Example:** @SIMPLE_REPLACE('009', 'REPLACE_009_WITH_Add') returns Add.

Run-Time Functions

The run-time functions available in iWay Transformer are described in the following table.

Run-Time Function	Description
@IWENCR (*param1*)	Returns the encrypted value of the parameter, based on the iWay Service Manager internal encryptor. **Parameter:** *param1*. String to be encrypted. **Example:** @IWENCR('1234') returns the value in a format similar to: @ENCR(32533101323532123122319631833137).
@SREG (*param1*, *param2*)	Returns some of the iWay Service Manager special registers defined by the specified parameter. This functionality is supported for run-time purposes only. If the function is tested in design time, *param2* is always returned. **Parameters:** *param1*. String that represents the name of the special register whose value is returned. *param2*. String that represents the default value to return if the special register identified by *param1* is not found. The second parameter of the @SREG() function must consist of a mapped dynamic value. A static default value as the second parameter is not allowed. When you troubleshoot @SREG() evaluation problems, we recommend that you map a dynamic value to the second parameter instead of using a static default value.

Run-Time Function	Description
@SET_SREG (*param1*, *param2*, *param3*, *param4*)	Sets the specified iWay Service Manager special register to the specified value, and returns *param4*. This functionality is supported for run-time purposes only. The special register name is specified in *param1*. The value to be assigned is specified in *param2*. The type of the special register is specified in *param3*. The last parameter, *param4*, contains the value to be returned upon successful execution. **Parameters:** *param1*. String that represents the name of the special register. *param2*. String that represents the new value of the special register. *param3*. Number that represents the type of the special register. Possible values include: ❏ 2: user-defined variable ❏ 3: user-defined emit header *param4*. String that represents the expected return value of this function. **Example:** @SET_SREG ('custom_functions_location', 'tools/transformer/custom_functions','2','custom functions location is set') returns 'custom function location is set'.
@SREG_EXISTS (*param1*)	Determines if an iWay Service Manager special register with the name specified in the parameter is already defined, and returns a true or false response. This functionality is supported for run-time purposes only. **Parameters:** *param1*. String that represents the name of the special register. **Example:** @SREG_EXISTS('custom_functions_location') returns true if the register is already defined. Returns false if the register is not defined.

Run-Time Function	Description
@REMOVE_SREG (*param1*, *param2*)	Removes the iWay Service Manager special register with the name specified in *param1*, and returns *param2*. This functionality is supported for run-time purposes only. **Parameters:** *param1*. String that represents the name of the special register. *param2*. String that represents the expected return value of this function. **Example:** @REMOVE_SREG ('custom_functions_location','custom functions location is removed') returns 'custom function location is removed'.

Security Functions

The security functions available in iWay Transformer are described in the following table.

Security Function	Description
@CHKDGT (*param1*, *param2*)	Determines if the alphanumeric character, at the specified position of the string in *param1*, is a number or a letter. Returns 'true' if it is a number, or 'false' if it is a letter. **Parameters:** *param1*. Alpha string that represents the number to be checked. *param2*. Position of the character in *param1* to check. The position numbering starts at 0 (zero) from the left. Supply this parameter as a constant in the Mapping Builder. **Examples:** @CHKDGT('4abc', '0') returns 'true'. @CHKDGT('6a89', '1') returns 'false'.

String Functions

The string functions available in iWay Transformer are described in the following table.

String Function	Description
@CONCAT (*param1*, *param2*, *param3*, *param4*)	Returns the string of concatenations of the specified parameters. If required, you can add or remove parameters for the @CONCAT function using the Mapping Builder. For more information, see *Customizing @CONCAT Functions* on page 280. The following signatures of the @CONCAT function are available: @CONCAT(*param1*, *param2*) @CONCAT(*param1*, *param2*, *param3*) @CONCAT(*param1*, *param2*, *param3*, *param4*) **Parameters:** *param1*. String to be concatenated. *param2*. String to be concatenated. *param3*. String to be concatenated. *param4*. String to be concatenated. **Example:** @CONCAT('The cow ', 'jumped ', ' over ', ' the moon') returns 'The cow jumped over the moon'.
@CRLF	Returns the combination of the Carriage Return and new Line Feed characters. **Parameters:** None. **Example:** @CONCAT('First line',@CRLF(),'Second line') returns 'First line Second line'.

String Function	**Description**
@DELSTR (*param1*, *param2*, *param3*)	Deletes the character substring from an alpha string for the length specified in *param3*. **Parameters:** *param1*. Alpha string or alpha string expression. *param2*. Position of the first character to be deleted. *param3*. Number of characters to be deleted, beginning with *param2* and continuing to the right. **Examples:** @DELSTR('ABCD', '2', '1') deletes the second letter of the string and returns 'ACD'. @IF(Y<0, @DELSTR(X, '1', '1'), @IF(Y>0, @DELSTR(X, '2', '1'), X)) If X contains a character string with a length greater than or equal to 2, the expression removes either the first or second character, or leaves the string intact, depending on the value that appears in column Y (negative, positive, or zero).
@EQUALS (*param1*, *param2*)	Compares two strings, and returns 'true' if they are equal or 'false' if they are not. **Parameters:** *param1*. String to compare. *param2*. String to compare. **Example:** @EQUALS('BA', 'AB') returns 'false'.
@FILL (*param1*, *param2*)	Repeats an alpha string or expression multiple times. This function accepts a maximum of 32 kilobytes (KB). **Parameters:** *param1*. An alpha string or expression. *param2*. The number of times that the string is repeated. **Example:** @FILL('*',5) creates a string of five asterisks '*****'.

String Function	Description
@FLIP (*param1*)	Reverses an alpha string, or the result of an alpha expression, to its mirrored image. **Parameter:** *param1*. Alpha string or alpha string expression. **Example:** @FLIP('Good') returns 'dooG'.
@HSTR (*param1*)	Returns the hexadecimal (base 16) string value of a decimal (base 10) number specified as *param1*. **Parameter:** *param1*. Decimal (base 10) number or numeric expression that represents a decimal number. **Example:** @HSTR('15') returns 'F'. @HSTR('16') returns '10'.
@HVAL (*param1*)	Returns the decimal (base 10) value of a hexadecimal (base 16) number specified as *param1*. **Parameter:** *param1*. Alpha string that represents a hexadecimal (base 16) number. **Example:** @HVAL('FF') returns 255. @HVAL('10') returns 16.
@INSERT (*param1*, *param2*, *param3*, *param4*)	Inserts one string into another at the specified position. **Parameters:** *param1*. Alpha string that represents the target string. *param2*. Alpha string that represents the source string. *param3*. Number that represents the character position in *param1*, after which the insertion takes place. *param4*. Number that represents how many characters from *param2* are inserted into *param1*. **Example:** @INSERT('abcde', 'xxx', '3', '2') returns 'abcxxde'.

String Function	Description
@INSTR (*param1*, *param2*)	Returns an integer that represents the first position of a substring within an alpha string or alpha expression. If the search argument is not found, the function returns 0 (zero). **Parameters:** *param1*. Input string or alpha expression string. *param2*. Alpha string that is searched for in the input string. **Examples:** @INSTR('abcd', 'b') returns 2. @INSTR('ABCDEF', 'DE') returns 4.
@LEFT (*param1*, *param2*)	Returns a substring of an alpha string, starting from the left, with its length specified in *param2*. **Parameters:** *param1*. Input string. *param2*. Number of characters to be returned, starting from the left. **Example:** @LEFT('abcdefg', '3') returns 'abc'.
@LEN (*param1*)	Returns an integer number that equals the length of an alpha string. The string is right-justified (for example, trailing blanks are removed) before starting. **Parameter:** *param1*. Input string. **Example:** @LEN ('abcdefg') returns 7.
@LOWER (*param1*)	Returns a string converted to all lowercase letters. **Parameter:** *param1*. Input string. **Example:** @LOWER('Who said THAT?') returns 'who said that?'

String Function	Description

@LPAD
(*param1*, *param2*)
	Returns the specified input string padded to the left, using the string specified in *param1*. The padding is inserted to the left of the string input until the length specified in *param2* is reached. If padding is not performed (for example, if there is an invalid parameter or the desired length is smaller than the length of the input string), *param1* is returned.
	
	Parameters:
	
	param1. String input to be padded.
	
	param2. Number that represents the desired length of the returned string.
	
	Examples:
	
	Single quotation marks are not part of the data. They are used to visually mark the length of the padded string.
	
	`@LPAD('constant','12') returns '            constant'.`
	
	`@LPAD('constant','3') returns '   constant'.`

String Function	Description

@LPAD (*param1*, *param2*, *param3*)	Returns the specified input string padded to the left, using the string specified in *param3*. The string is inserted to the left of the input string until the desired length specified in *param2* is reached. If padding is not performed (for example, if there is an invalid parameter or the desired length is smaller than the length of the input string), *param1* is returned.
	
	Parameters:
	
	param1. Input string to be padded.
	
	param2. Number that represents the length of the returned string.
	
	param3. String used for padding.
	
	Examples:
	
	Single quotation marks are not part of the data. They are used to visually mark the length of the padded string.
	
	`@LPAD('constant','12', 'L') returns 'LLLLLLLLLLLLconstant'.`
	
	`@LPAD('constant','3', 'L') returns 'LLLconstant'.`

@LTRIM (*param1*)	Removes leading white spaces (such as blanks, tabs, line feeds) from an alpha string or an alpha expression. **Parameter:** *param1*. Input string. **Example:** @LTRIM(' John') returns 'John'.
@LTRIM (*param1*, *param2*)	Removes a leading specified trim character from an alpha string or an alpha expression. **Parameters:** *param1*. Input alpha string or alpha expression. *param2*. Specified trim character. **Example:** @RTRIM('XJohn ',X) returns 'John'.

String Function	Description
@LTRIM (*param1*, *param2*, *param3*)	Removes leading white space (blanks, tabs, line feeds) or specified trim characters from an alpha string or an alpha expression. **Parameters:** *param1*. Input alpha string or alpha expression. *param2*. Specified trim character. *param3*. A true or false flag to trim all occurrences of the trim character specified in *param2*. **Example:** @RTRIM('XXJohn ',X, true) returns 'John'.
@MID (*param1*, *param2*, *param3*)	Returns the substring extract of the specified number of characters from an input string. This function will be deprecated in future releases. Use @SUBSTR instead. **Parameters:** *param1*. Input string. *param2*. Number that represents the starting position of the substring within *param1*. *param3*. Number of characters to be extracted (the length of the substring). **Example:** @MID('John', '3', '2') returns 'hn'.
@NOT_EQUALS (*param1*, *param2*)	Compares two strings in *param1* and *param2*, and returns 'false' if they are equal, or 'true' if they are not. **Parameters:** *param1*. String. *param2*. String. **Example:** @EQUALS('BA', 'AB') returns 'true'.

String Function	Description
@QUOTE (*param1*)	Returns the specified parameter in *param1*, delineated by the single quotation marks. **Parameter:** *param1*. String to be quoted. **Examples:** @QUOTE('quote me') returns 'quote me'. @QUOTE('23') returns '23'.
@QUOTEGEN (*param1*)	Generates a WHERE clause in an SQL statement. Generates single quotation marks around an alpha string, but not if the input is an integer or a number. **Parameter:** *param1*. String. It can be an input node value, a constant, or the result of another function. **Examples:** @QUOTEGEN('HELLO') returns 'HELLO'. @QUOTEGEN('1234') returns 1234. @CONCAT('SELECT column1 from table where column2 = ', @QUOTEGEN(parent/child/value)) returns: ❑ "SELECT column1 from table where column2 = 'value', IF parent/child/value type is an alpha string." ❑ "SELECT column1 from table where column2 = value, IF parent/child/value type is integer or number."

String Function	Description
@REP (*param1*, *param2*, *param3*, *param4*)	Returns the result of the replacement of an alpha substring in an input string with another substring. **Parameters:** *param1*. Input alpha string or expression in which the replacement takes place. *param2*. Alpha string or expression that provides the substring to copy to *param1*. *param3*. First position in *param1* that receives the substring from *param2*. *param4*. Number of characters that are moved from *param2* to *param1*, starting from the left-most character of *param2*. **Example:** @REP('12345', 'abcde', '3', '2') returns '12ab5'.
@RIGHT (*param1*, *param2*)	Returns a substring of an alpha string, starting from the right, with its length specified in *param2*. **Parameters:** *param1*. Input string from which the characters are taken. *param2*. Number of characters to be retrieved, starting from the character furthest right. **Example:** @RIGHT('abcdefg ', '3') returns 'efg'.

String Function	Description

@RPAD
(*param1*, *param2*)
	Returns the specified input string padded to the right, using the string specified in *param1*. The padding is inserted to the right of the input string until the desired length specified in *param2* is reached. If padding is not performed (for example, if there is an invalid parameter or the desired length is smaller than the length of the input string), *param1* is returned.
	
	Parameters:
	
	param1. String to be padded.
	
	param2. Number that represents the length of the returned string.
	
	Examples:
	
	Single quotation marks are not part of the data. They are used to visually mark the length of the padded string.
	
	`@RPAD ('constant','12') returns 'constant            '.`
	
	`@RPAD ('constant','3') returns 'constant   '.`

String Function	Description

@RPAD (*param1*, *param2*, *param3*)	Returns the specified input string padded to the right, using the string specified in *param3*. The padding string is inserted to the right of the input string until the desired length specified in *param2* is reached. If padding is not performed (for example, if there is an invalid parameter or the desired length is smaller than the length of the input string), the first parameter is returned.
	
	Parameters:
	
	param1. String to be padded.
	
	param2. Number that represents the length of the returned string.
	
	param3. String used for padding.
	
	Examples:
	
	Single quotation marks are not part of the data. They are used to visually mark the length of the padded string.
	
	`@RPAD ('constant','12', 'R') returns 'constantRRRRRRRRRRRR'.`
	
	`@RPAD ('constant','3', 'R')returns 'constantRRR'.`

@RTRIM (*param1*)	Removes trailing white spaces (such as blanks, tabs, line feeds) from an alpha string or an alpha expression. **Parameter:** *param1*. Input alpha string or alpha expression. **Example:** @RTRIM('John ') returns 'John'.
@RTRIM (*param1*, *param2*)	Removes a trailing specified trim character from an alpha string or an alpha expression. **Parameters:** *param1*. Input alpha string or alpha expression. *param2*. Specified trim character. **Example:** @RTRIM('JohnX ',X) returns 'John'.

String Function	**Description**
@RTRIM (*param1*, *param2*, *param3*)	Removes trailing white space (blanks, tabs, line feeds) or specified trim characters from an alpha string or an alpha expression. **Parameters:** *param1*. Input alpha string or alpha expression. *param2*. Specified trim character. *param3*. A true or false flag to trim all occurrences of the trim character specified in *param2*. **Example:** @RTRIM('JohnXX ',X, true) returns 'John'.
@STRTOKEN (*param1*, *param2*, *param3*)	Returns a specified string token from a delimited token string. **Parameters:** *param1*. Input string, delimited with tokens. *param2*. Requested token index (numeric). *param3*. Delimiter. You can use valid <XML> element tags (nodes) as delimiters. The XML format of a list can be a variable or URL. Content must consist of a <list><tag>Value</tag></list> set, in which element names can be arbitrary but must be consistent. **Parameter Notes:** ❏ The third parameter, *param3*, can have more than one character as the delimiter. ❏ An empty string is returned if a delimiter is not found or is empty, or if the input string is empty. ❏ Every delimiter is counted for the index calculation (no repetition). **Example:** variable BA = abcd,cdef,ghik,lmnp, then @STRTOKEN(BA, '2', ',') returns cdef.

String Function	Description
@SUBSTR (*param1*, *param2*, *param3*)	Returns the substring representing the extracts of a specified number of characters from an alpha string. **Parameters:** *param1*. Input alpha string. *param2*. Number that represents the starting position of the substring within *param1*. *param3*. Number of characters to be extracted (the length of the substring). **Example:** @SUBSTR('John', '3', '2') returns 'hn'.
@TRIM (*param1*)	Removes white space characters (such as blanks, tabs, and line feeds) from the left and right sides of an alpha string or an alpha expression. **Parameter:** *param1*. Input alpha string. **Example:** @TRIM(' John ') returns 'John'.
@TRIM (*param1*, *param2*)	Removes a specified trim character from the left and right sides of an alpha string or an alpha expression. **Parameters:** *param1*. Input alpha string or alpha expression. *param2*. Specified trim character. **Example:** @RTRIM('XJohnX ',X) returns 'John'.

String Function	Description
@TRIM (*param1*, *param2*, *param3*)	Removes white space characters (such as blanks, tabs, and line feeds) or specified trim characters from the left and right sides of an alpha string or an alpha expression. **Parameters:** *param1*. Input alpha string or alpha expression. *param2*. Specified trim character. *param3*. A true or false flag to trim all occurrences of the trim character specified in *param2*. **Example:** @RTRIM('XXJohnXX ',X, true) returns 'John'.
@UPPER (*param1*)	Returns a string converted to all uppercase letters. **Parameter:** *param1*. Alpha string. **Example:** @UPPER('Pablo Picasso') returns 'PABLO PICASSO'.

Time Functions

The time functions available in iWay Transformer are described in the following table.

Time Function	**Description**
@ADD_DATE (*param1*, *param2*, *param3*, *param4*)	Performs a calculation on a date variable. It constructs a resulting date out of an input date and three values added to that date: years, months, and days. The result is always a valid date format. **Parameters:** *param1*. Input date (format: MM/dd/yyyy). *param2*. Number of years to add to *param1*. *param3*. Number of months to add to *param1*. *param4*. Number of days to add to *param1*. A parameter with a value of 0 (zero) is ignored. **Example:** @ADD_DATE('01/01/1992', '1', '2', '2') returns 03/03/1993.
@ADD_TIME (*param1*, *param2*, *param3*, *param4*)	Performs a calculation on a time variable. It constructs a resulting time out of an input time and three values added to that time: hours, minutes, and seconds. The result is always a valid time format. **Parameters:** *param1*. Input time. *param2*. Number of hours to add to *param1*. *param3*. Number of minutes to add to *param1*. *param4*. Number of seconds to add to *param1*. A parameter with a value of 0 (zero) is ignored. **Example:** @ADD_TIME('12:00:00', '1', '2', '3') returns 13:02:03.
@CUSTOMDATE()	Will be deprecated in future releases. Use @DATE instead.
@DATE (*param1*)	Returns the system date in the specified format. For more information, see *Date Pictures* on page 142. **Parameter:** *param1*. Date format. **Example:** If the system date is 01/28/1992, @DATE ('dd/MM/yyyy') returns '28/01/1992'.

Time Function	Description
@DAY (*param1*)	Returns the day of the specified date as a number between 1 and 31. **Parameter:** *param1*. Input date. It is a date or date expression. **Example:** @DAY('01/28/1992') returns '28'.
@DOW (*param1*)	Returns the number of the day of the week for the specified date. For example, Sunday is 1 and Monday is 2. **Parameter:** *param1*. Input date (format MM/dd/yyyy). **Example:** @DOW('01/29/1992'), representing a Wednesday, returns '4'.
@DSTR (*param1*, *param2*)	Returns the specified date in the format specified by the second parameter. The original input must be in the format 'MM/dd/yyyy'. For more information, see *Date Pictures* on page 142. **Parameters:** *param1*. Input date (format MM/dd/yyyy). *param2*. Picture mask (format) of the returned date string. **Example:** @DSTR('2/12/1998','MMMM dd, yyyy') returns 'February 12, 1998'.
@DSTR (*param1*, *param2*, *param3*)	Returns an input date in the format specified by the third parameter. The input date must conform to the format specified by the second parameter. For more information, see *Date Pictures* on page 142. **Parameters:** *param1*. Input date. *param2*. Picture mask of the input date. *param3*. Picture mask (format) of the returned date string. **Example:** @DSTR('2/12/1998','dd/MM/yyyy','MMMM dd, yyyy') returns 'December 02, 1998'.

Time Function	Description
@DSTR (*param1*, *param2*, *param3*, *param4*)	Returns an input date format according to the fourth parameter, which accepts a Boolean value (true or false). **Parameters:** *param1*. Input date. *param2*. Picture mask of the input date. *param3*. Picture mask (format) of the returned date string. *param4*. Determines whether or not the input date complies with standard formatting requirements.
@DVAL (*param1*, *param2*)	Converts an input date to a numeric value. The numeric value represents the number of days elapsed since the day before the first day of the first century (01/01/01) until the input date. For more information, see *Date Pictures* on page 142. **Parameters:** *param1*. Input date string that can be interpreted as a date (for example, '01/01/92', 'Jan 1, 1992'). *param2*. Format of the input date. This parameter is required for the system to read and interpret *param1*. **Example:** @DVAL('01/01/92', 'MM/dd/yy') and @DVAL('Jan 1, 1992', 'MMM dd, yyyy') each return 727198.
@EOM (*param1*)	Returns the date of the end of the month specified in the parameter. **Parameter:** *param1*. Input date. **Example:** @EOM ('05/05/93') returns 05/31/93.
@EOY (*param1*)	Returns the date of the end of the year specified in the parameter. **Parameter:** *param1*. Input date. **Example:** @EOY ('10/05/93') returns '12/31/93'.

Time Function	**Description**
@GD2JD (*param1*, *param2*)	Converts a date in Gregorian format to Julian format. The Gregorian date format, which is based on the Gregorian solar calendar, is the most widely used date format in the world. A Gregorian date is specified by the Year, the Month (identified by name or number), and the Day of the Month (numbered sequentially starting at 1). For example, 2008-03-25 is the Gregorian date representation for March 25, 2008. In J.D. Edwards systems, Julian dates (JD) are identified by the year first, followed by the number of days into the year at which this date appears. For example, Jan 15, 1999 is represented as 99015. March 15 is 99074. March 15, 2001 is 101074, since dates in the current century start with 100 (2000), 101 (2001), and so on. **Parameters:** *param1*. Gregorian date. *param2*. Gregorian date picture mask.
@HOUR (*param1*)	Returns a number that represents the hour portion of the input time. **Parameter:** *param1*. Input time. **Example: @**HOUR('2:00:00') returns 2.

Time Function	Description
@JD2GD (*param1*, *param2*)	Converts a date in Julian format to Gregorian format. In J.D. Edwards systems, Julian dates (JD) are identified by the year first, followed by the number of days into the year at which this date appears. For example, Jan 15, 1999 is represented as 99015. March 15 is 99074. March 15, 2001 is 101074, since dates in the current century start with 100 (2000), 101 (2001), and so on. The Gregorian date format, which is based on the Gregorian solar calendar, is the most widely used date format in the world. A Gregorian date is specified by the Year, the Month (identified by name or number), and the Day of the Month (numbered sequentially starting at 1). For example, 2008-03-25 is the Gregorian date representation for March 25, 2008. **Parameters:** *param1*. Julian date. *param2*. Gregorian date picture mask.
@MINUTE (*param1*)	Returns a number that represents the minutes portion of the input time. **Parameter:** *param1*. Input time. **Example:** @MINUTE('2:35:00') returns 35.
@MONTH (*param1*)	Returns a number that represents the month portion of the input time. **Parameter:** *param1*. Input time. **Example:** @MONTH('01/28/1992') returns 1.
@SECOND (*param1*)	Returns a number that represents the seconds portion of the input time. **Parameter:** *param1*. Input time. **Example:** @SECOND('12:02:05') returns 5.

Time Function	Description
@SOM (*param1*)	Returns a number that represents the start of the month portion of the input time. **Parameter:** *param1*. Input time. **Example:** @SOM ('05/18/93') returns '05/01/93'.
@SOY (*param1*)	Returns a number that represents the start of the year portion of the input time. **Parameter:** *param1*. Input time. **Example:** @SOY ('10/05/93') returns '01/01/93'.
@TIME (*param1*)	Returns the system time. For more information, see *Time Pictures* on page 143. **Parameter:** *param1*. Desired time format. **Example:** @TIME ('HH:mm:ss') returns 17:08:42.
@TSTR (*param1*, *param2*)	Converts a time to an alpha string, according to the format provider. A blank picture interprets the string as 'HH:mm:ss'. For more information, see *Time Pictures* on page 143. **Parameters:** *param1*. Desired input time. *param2*. Format of the resulting character string. **Example:** @TSTR ('14:30', 'HH:mm PM') returns '2:30 PM'.
@YEAR (*param1*)	Returns a number that represents the year portion of the input time. **Parameter:** *param1*. Date or date expression. **Example:** @YEAR('01/28/1992') returns 1992.

Numeric Pictures

The numeric picture masks available in iWay Transformer are described in the following table.

Symbol	Location	Description
0	Number	Digit.
#	Number	Digit. Zero shows as absent.
.	Number	Decimal separator or monetary decimal separator.
-	Number	Minus sign.
,	Number	Grouping separator.
E	Number	Separates mantissa and exponent in scientific notation. It does not require quotation marks in the prefix or suffix.
%	Prefix or suffix	Multiply by 100 and show as a percentage.

Examples of numeric picture masks are shown in the following table. The ^ symbol represents one space character.

Numeric Value	Picture	Resulting Numeric Value
1234.56	#,###.##	1,234.56
123456789.56	#,###.##	123,456,789.56
-1234.56	N###,###.##C	^^-1,234.56
-1234.56	N######.##L	-1234.56^^
-1234.56	N######.##P*	-**1234.56
0	N######.##Z*	*********
-13.5	N##.##-DB;	DB13.50
45.3	N##.##+CR;	CR45.30
-13.5	N##.##-(,);	(13.50)
4055.3	$######.##	$^^4055.30

Date Pictures

The following table describes the date symbols and shows an example of each.

Symbol	Description	Example	Presentation
G	Era designator.	AD	Text
y	Year.	1996	Number
M	Month in year.	July & 07	Text & Number
d	Day in month.	10	Number
E	Day in week.	Tuesday	Text
D	Day in year.	189	Number
F	Day of week in month.	2 (second Wed. in July)	Number
w	Week in year.	27	Number
W	Week in month.	2	Number
'	Escape for text.	'	Delimiter
' '	Single quotation mark.	'	Literal
-	Separates month, day, and year.	12-24-86	Delimiter
/	Separates month, day, and year.	12/24/86	Delimiter

The typical date formats are 'dd/MM/yyyy' (European), 'MM/dd/yyyy' (American), and 'yyyy/MM/dd' (Scandinavian). When you define the attribute *Date* for the parameter in one of the functions, you must also select the format for the date item, as described in the following table. You can change the default format and place in it any positional directives and masking characters that you require.

The following table provides examples of the date format (picture), using the date of 21 March 1992. The ^ symbol represents one space character.

Date Picture	Result
MM/dd/yyyy	03/21/1992
##/##/##	21/03/92 when an XML parser default is set to European 03/21/92 when an XML parser is set to American
MMMM^dd^yyyy	March^21^1992
MMM^dd, ^yyyy	Mar.^21^1992
EEEE^^-^7	Saturday^^^-^7
E^7	Mon^7

Time Pictures

The following table describes the time symbols and shows an example of each.

Symbol	Description	Example	Presentation
h	Hour in am/pm (1-12)	12	Number
H	Hour in day (0-23)	0	Number
m	Minute in hour	30	Number
s	Second in minute	55	Number
-	Separates hours from seconds	1-22	1-22
:	Separates hours from seconds	1:22	1:22
S	Millisecond	978	Number
a	AM/PM marker	PM	Text
k	Hour in day (1-24)	24	Number
K	Hour in am/pm (0-11)	0	Number
z	Time zone	Pacific Standard Time	Text

The following table shows examples of time pictures and results.

Time Picture	Result	Description
HH:mm:SS	08:20:00	Time displayed on 24-hour clock.
HH:mm:SS	16:40:00	Time displayed on 24-hour clock.
HH:mm PM	8:20 pm	Time displayed on 12-hour clock.
HH:mm PM	4:40 pm	Time displayed on 12-hour clock.
HH-mm-SS	16-40-00	Example of the minus sign (-) as the time separator.

Custom Functions

A function is a procedure that is built within the iWay Transformer graphical user interface. It produces the required output based on calculation upon, or manipulation of, input data. You can apply a function to produce the output value for a specific node using the Mapping Builder.

In addition to the set of predefined functions discussed in *Predefined Functions* on page 102, iWay Transformer enables you to implement your own custom functions in Java, according to the format specified in this chapter.

You can integrate custom functions into iWay Transformer and make them available for use in your transformations at run time or design time.

Invisible Group

Invisible Group mapping techniques can help you perform complex looping tasks when designing mappings of the Transform component.

What Is It?

Invisible Group is a group node with the Visible property set to false. It is identified by the grayed out group icon ◆.

Where Is It Used?

You use Invisible Group in the Mapping Builder when designing the Output structure of the mappings. For more information, see *Mapping Builder*.

What Is It Used for?

Use Invisible Group to manipulate the Output structure of the mappings by wrapping the structure nodes with the invisible group nodes.

Where Is the Visible Property Located?

You can find the Visible property of the Group node in the Filter properties. For more information, see *Filter Tab* on page 245.

Chapter 4

iWay Transformer Tasks

The user interface of iWay Transformer provides you with the tools required to create and manage Transform components. It enables you to maintain high-level and detailed aspects of a project in a customizable environment.

This section describes the basic menus and options available in iWay Transformer.

In this chapter:

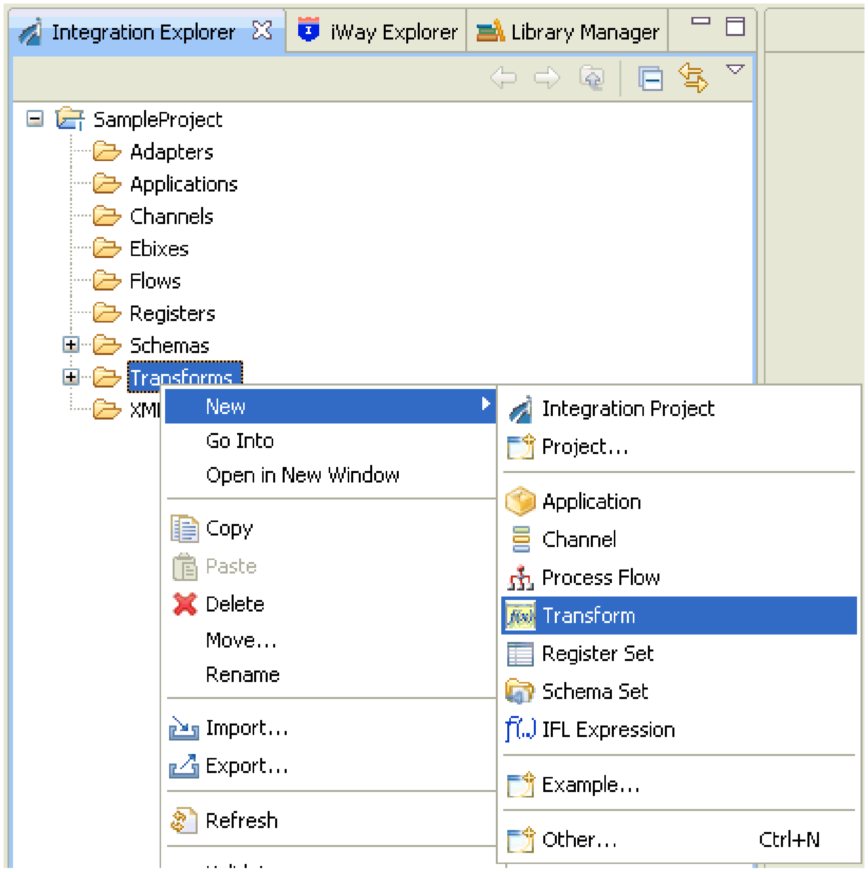
Creating a Transform Component

iWay Transformer provides a quick way to create a Transform component. It guides you through the major steps of building the component. After you create the component, you can manipulate it using the editing and configuration tools provided with iWay Transformer. The following procedure shows you how to create a Transform component using iWay Transformer.

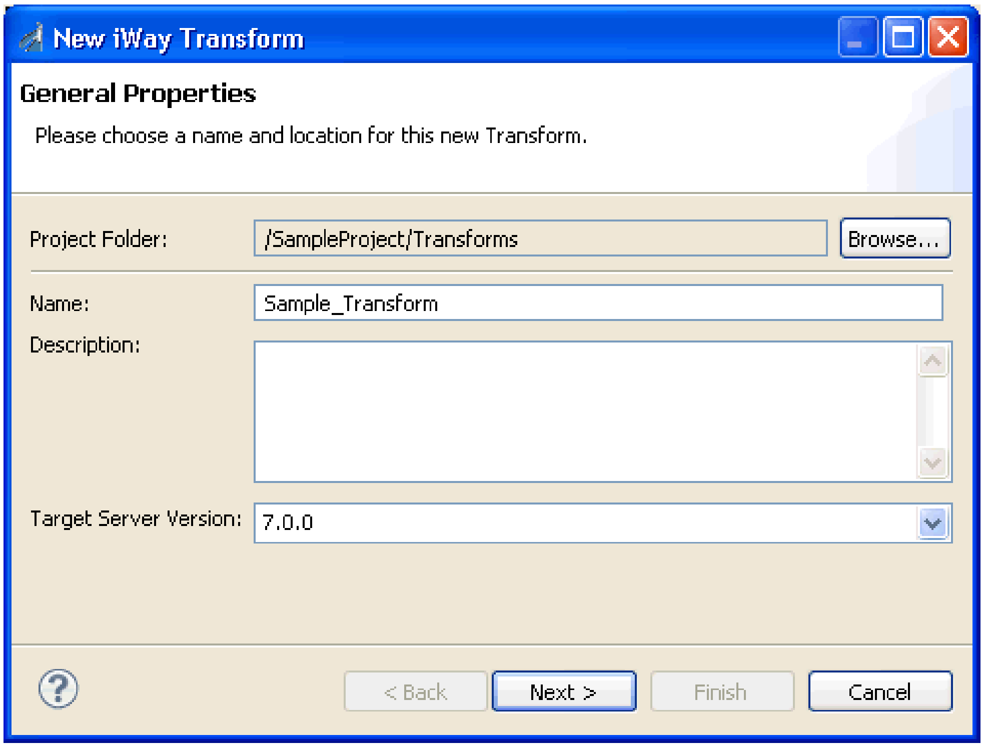
Procedure: How to Create a Transform Component

To create a Transform component:

1. Right-click the *Transforms* folder in your Integration Project, select *New*, and then click *Transform*, as shown in the following image.



The New iWay Transform dialog box opens, as shown in the next image.



General Properties

Please choose a name and location for this new Transform.

Project Folder: /SampleProject/Transforms

Name: Sample_Transform

Description:

Target Server Version: 7.0.0

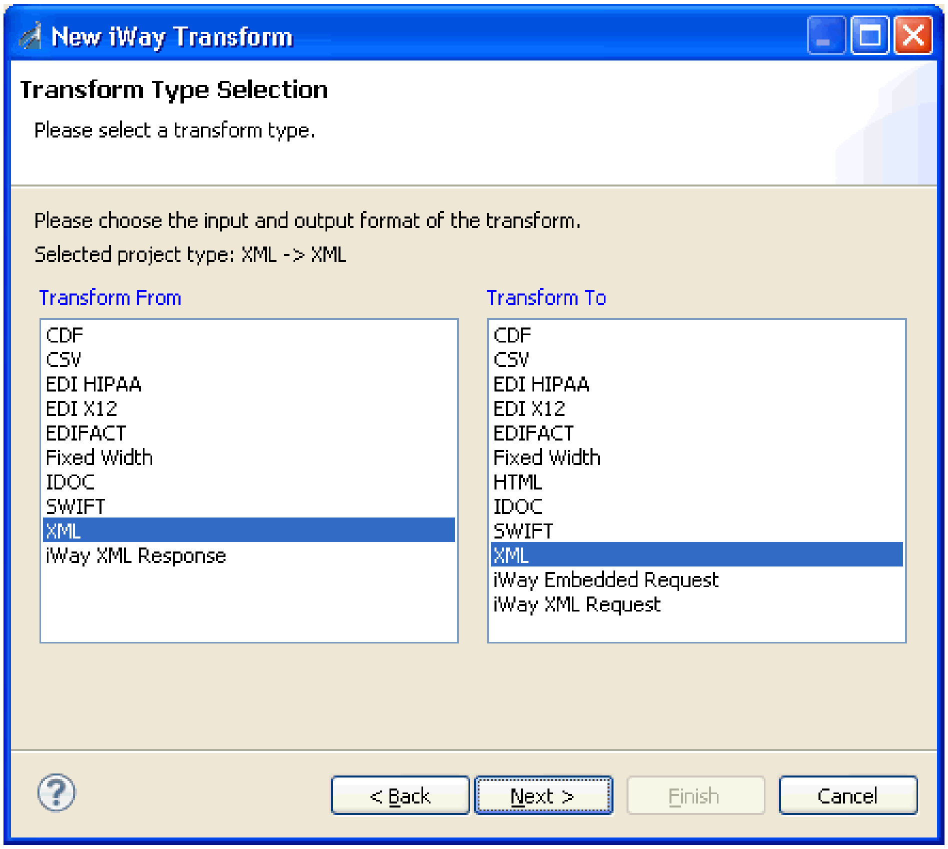
2. Type a name and a description (optional) for the new Transform component.

   Tip: If required, you can change the preselected project folder by clicking *Browse* next to the Project Folder field.

3. Select a target version of iWay Service Manager (iSM) from the Target Server Version drop-down list.

4. Click *Next*.

The Transform Type Selection pane opens, as shown in the following image.



Please choose the input and output format of the transform.

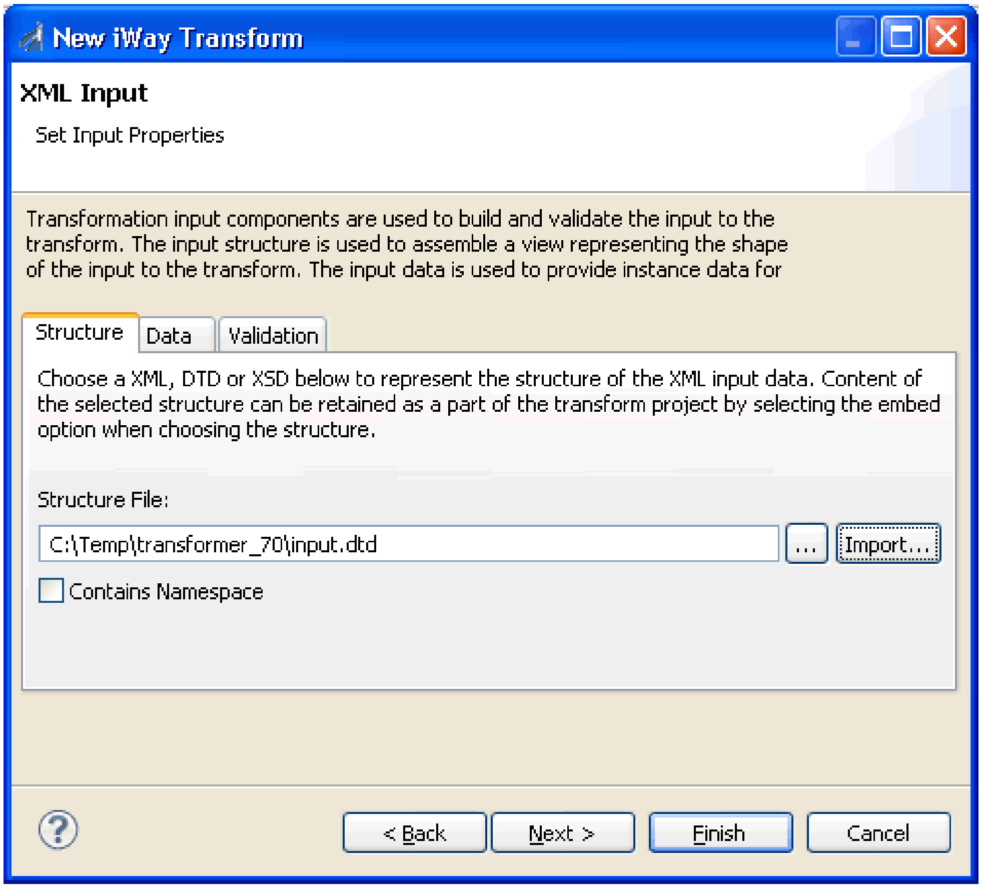
Selected project type: XML -> XML

Transform From: CDF, CSV, EDI HIPAA, EDI X12, EDIFACT, Fixed Width, IDOC, SWIFT, XML (selected), iWay XML Response

Transform To: CDF, CSV, EDI HIPAA, EDI X12, EDIFACT, Fixed Width, HTML, IDOC, SWIFT, XML (selected), iWay Embedded Request, iWay XML Request

5. From the list in the Transform From section, select the format of the input data, for example, *XML*.
6. From the list in the Transform To section, select the format of the output data, for example, *XML*.
7. Click *Next*.

The XML Input pane opens with the Structure tab selected, as shown in the following image.



XML Input

Set Input Properties

Transformation input components are used to build and validate the input to the transform. The input structure is used to assemble a view representing the shape of the input to the transform. The input data is used to provide instance data for

Choose a XML, DTD or XSD below to represent the structure of the XML input data. Content of the selected structure can be retained as a part of the transform project by selecting the embed option when choosing the structure.

Structure File: C:\Temp\transformer_70\input.dtd

☐ Contains Namespace

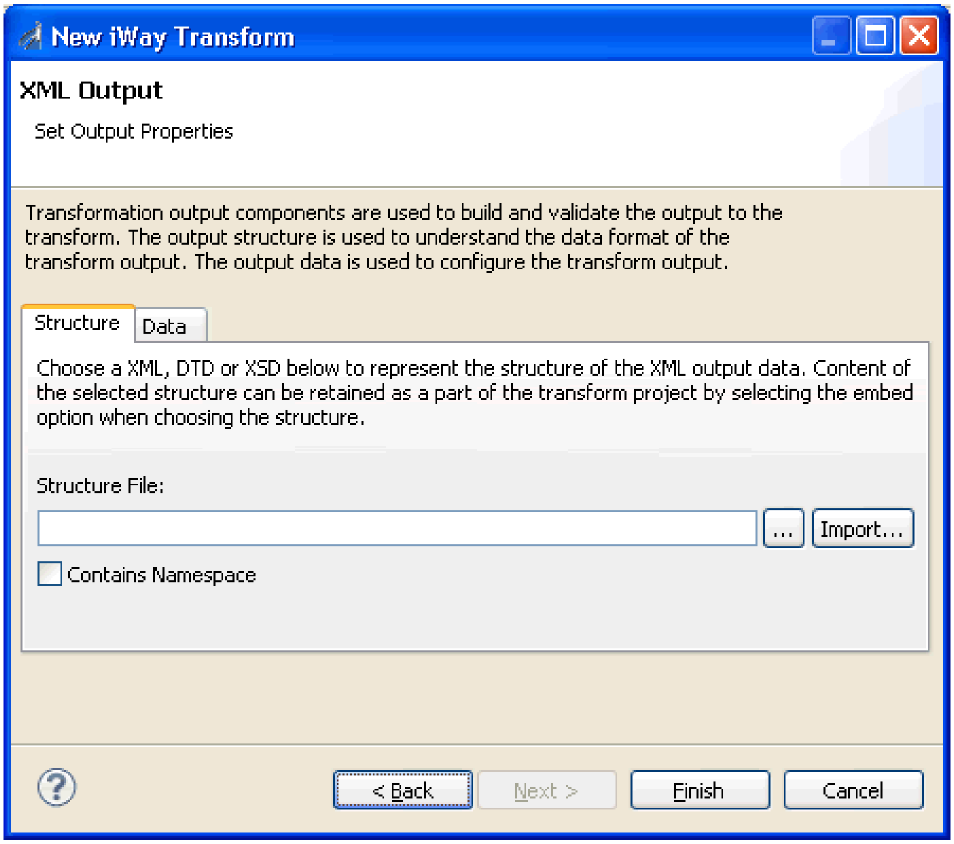
Tip: You can skip the remaining steps by clicking the *Finish* button if it is enabled. You can then manually alter the Transform component properties at your convenience from the Properties dialog box.

8. In the Structure File field, type the name of a DTD, XSD (schema), or XML file that represents the input dictionary.

   - ❏ Click the ellipsis button to the right of the field to browse the workspace for the structure file.
   - ❏ Click the *Import* button to the right of the field to browse your file system for the structure file.

9. Click *Next*.

   The XML Output pane opens with the Structure tab selected, as shown in the following image.



New iWay Transform

XML Output

Set Output Properties

Transformation output components are used to build and validate the output to the transform. The output structure is used to understand the data format of the transform output. The output data is used to configure the transform output.

Choose a XML, DTD or XSD below to represent the structure of the XML output data. Content of the selected structure can be retained as a part of the transform project by selecting the embed option when choosing the structure.

Structure File:

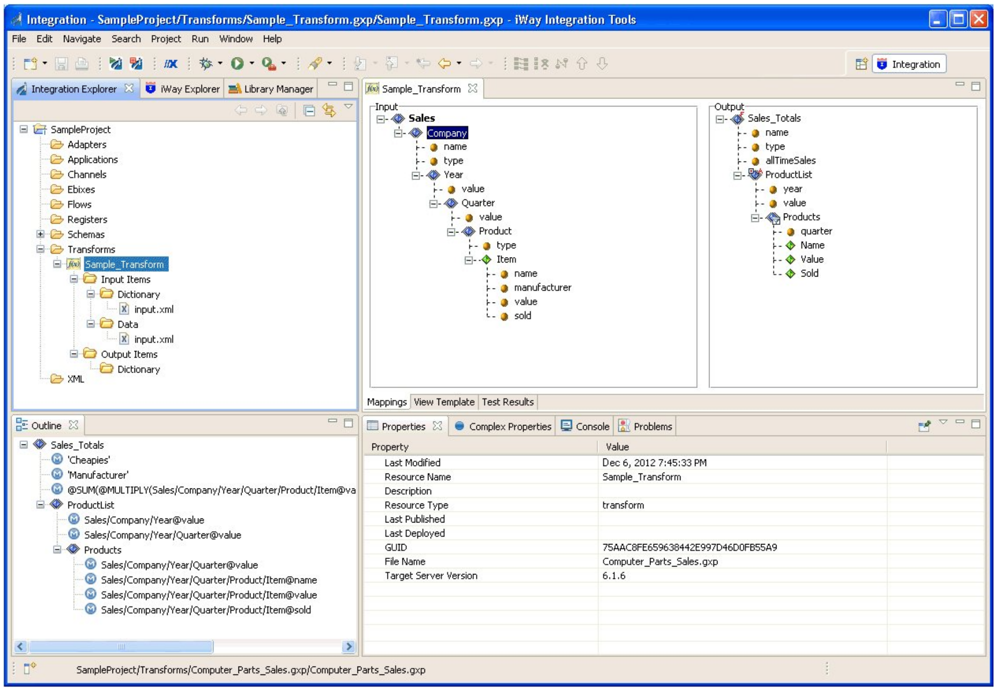
Contains Namespace

10. In the Structure File field, type the name of the data dictionary that represents the structure of the output data (optional).

    - Click the ellipsis button to the right of the field to browse the workspace for the structure file.
    - Click the *Import* button to the right of the field to browse your file system for the structure file.

11. Click *Finish*.

The new Transform component is displayed in the iWay Integration Tools workspace, as shown in the following image.



Property	Value
Last Modified	Dec 6, 2012 7:45:33 PM
Resource Name	Sample_Transform
Description	
Resource Type	transform
Last Published	
Last Deployed	
GUID	75AAC8FE659638442E997D46D0FB55A9
File Name	Computer_Parts_Sales.gxp
Target Server Version	6.1.6

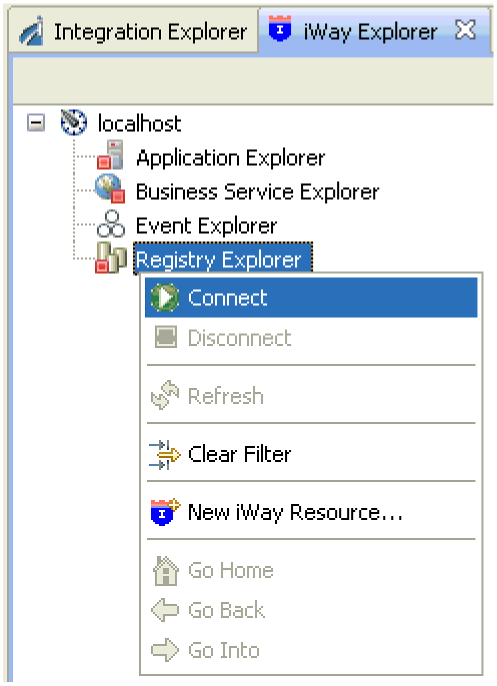
The Transform component and any related resources (for example, structure files or input data) exist in the workspace subdirectory that you named during the creation process. For example:

iIT_Home`\workspace\SampleProject\Transforms\Sample_Transform.gxp`

Procedure: How to Add a Transform Component From the iWay Registry

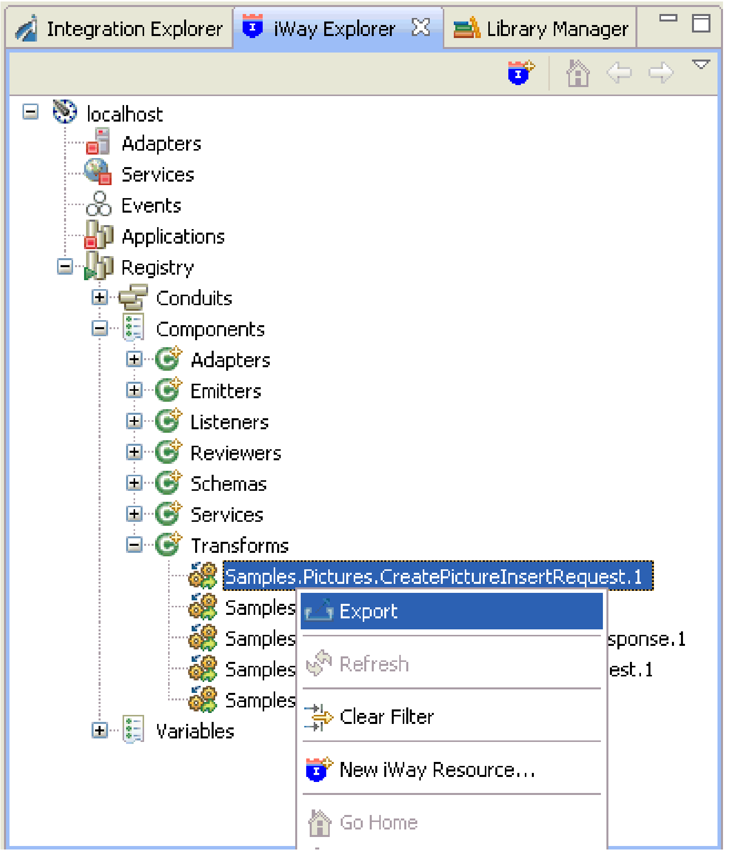
You can load a Transform component from the iWay registry into your Integration Project by exporting it using iWay Explorer.

1. Click the *iWay Explorer* tab.



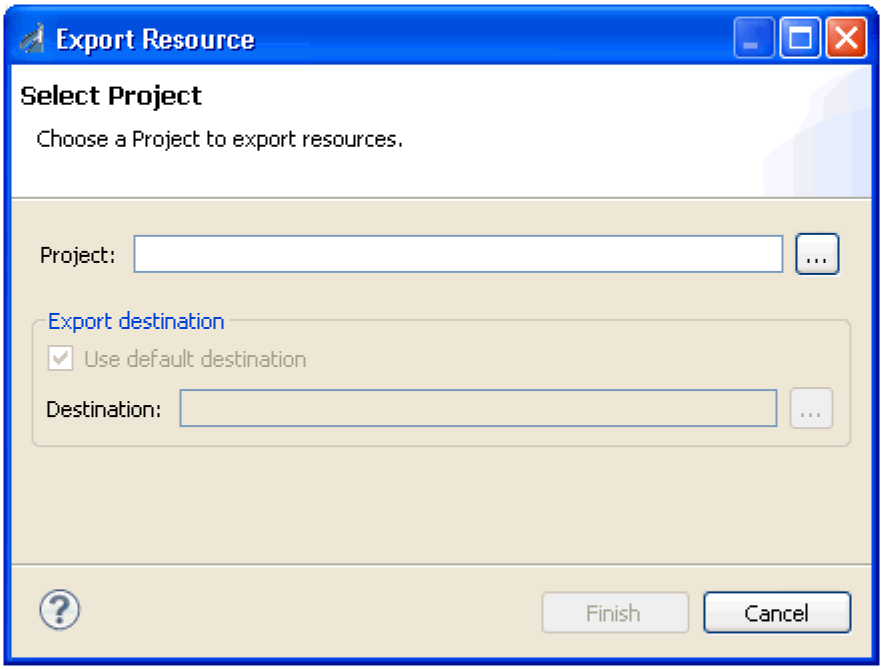
2. Right-click *Registry Explorer*, and select *Connect* from the context menu.

3. To obtain a list of Transform components that are available for export, expand *Registry Explorer*, *Components*, and then *Transforms*, as shown in the following image.



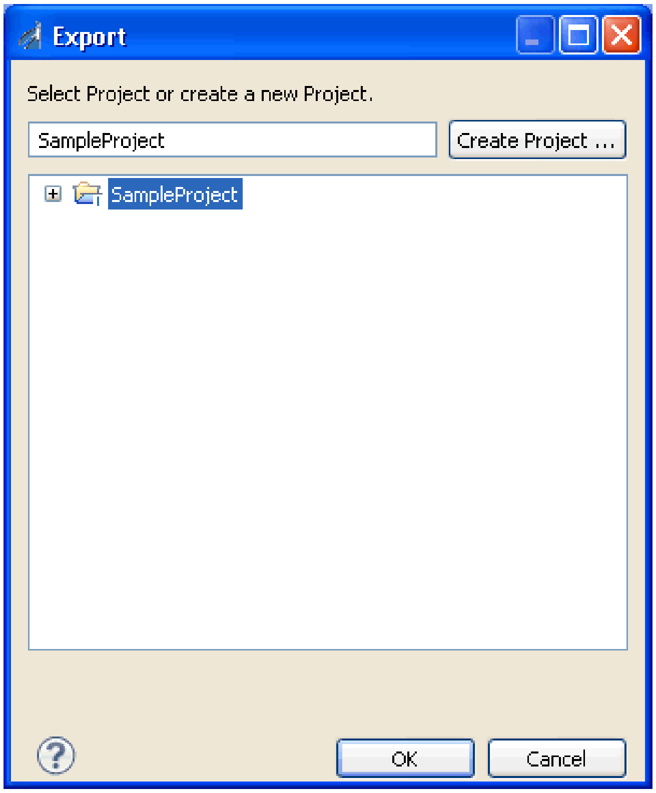
4. Right-click the Transform component, and select *Export* from the context menu.

The Export Resource dialog box opens, as shown in the following image.



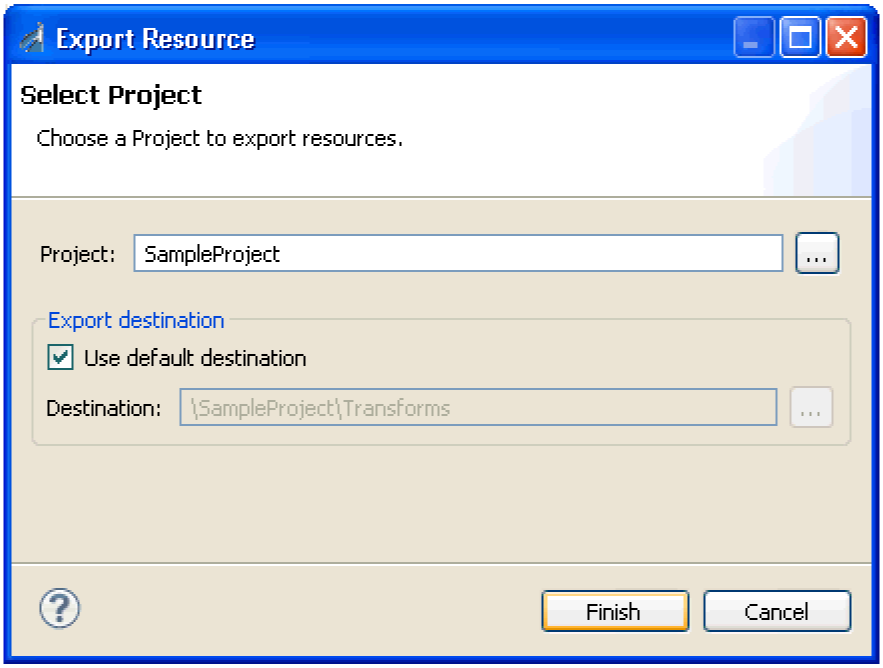
5. Click the ellipsis button to the right of the Project field.

The Export dialog box opens, as shown in the following image.



6. Select an available Integration Project to which the Transform component will be exported, and click *OK*.

You are returned to the Export Resource dialog box, where the selected Integration Project name is displayed in the Project field, as shown in the following image.



Project: SampleProject

☑ Use default destination

Destination: \SampleProject\Transforms

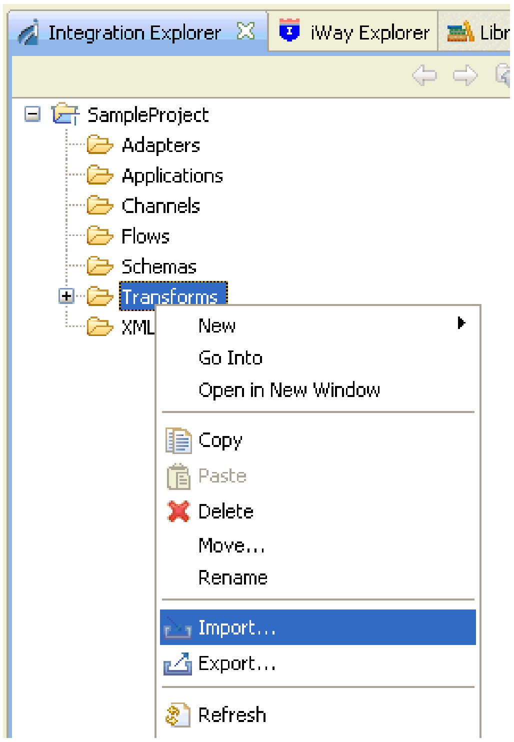
7. Click *Finish*.

Procedure: How to Add a Transform Component From the File System

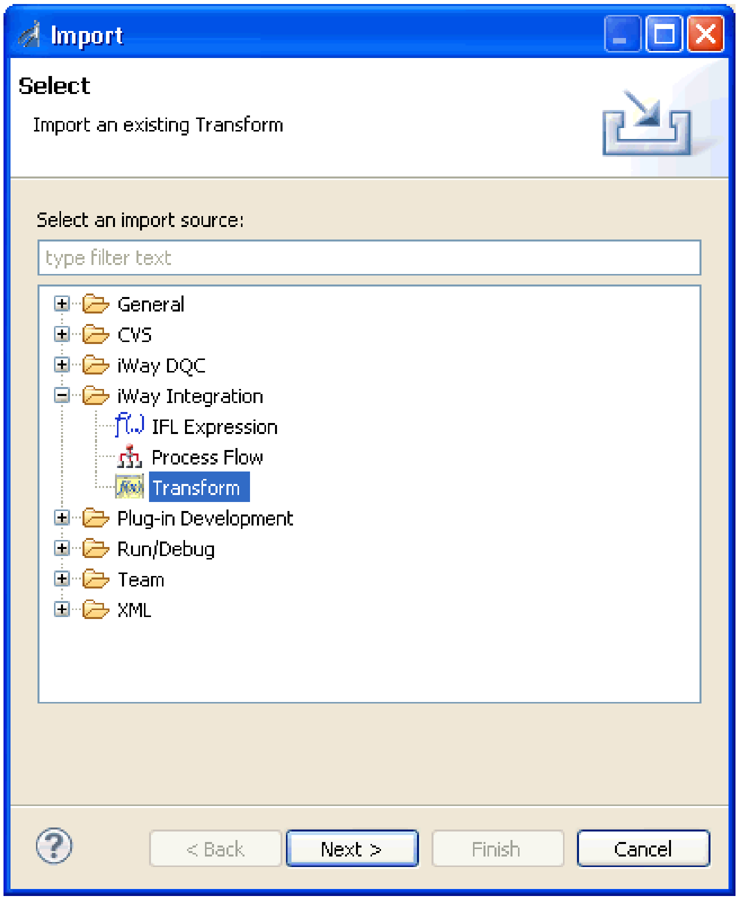
To add a Transform component from the file system:

1. Click the *Integration Explorer* tab.

2. Expand an available Integration Project, right-click the *Transforms* folder, and select *Import* from the context menu, as shown in the following image.



The Import dialog box opens, as shown in the next image.


Import

Select

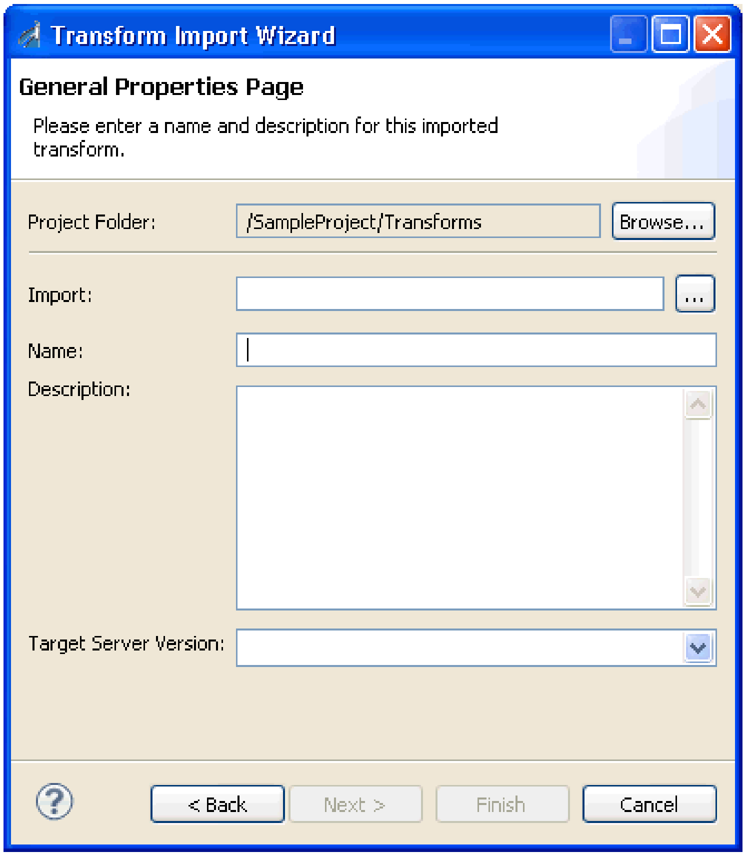
Import an existing Transform

Select an import source:

type filter text

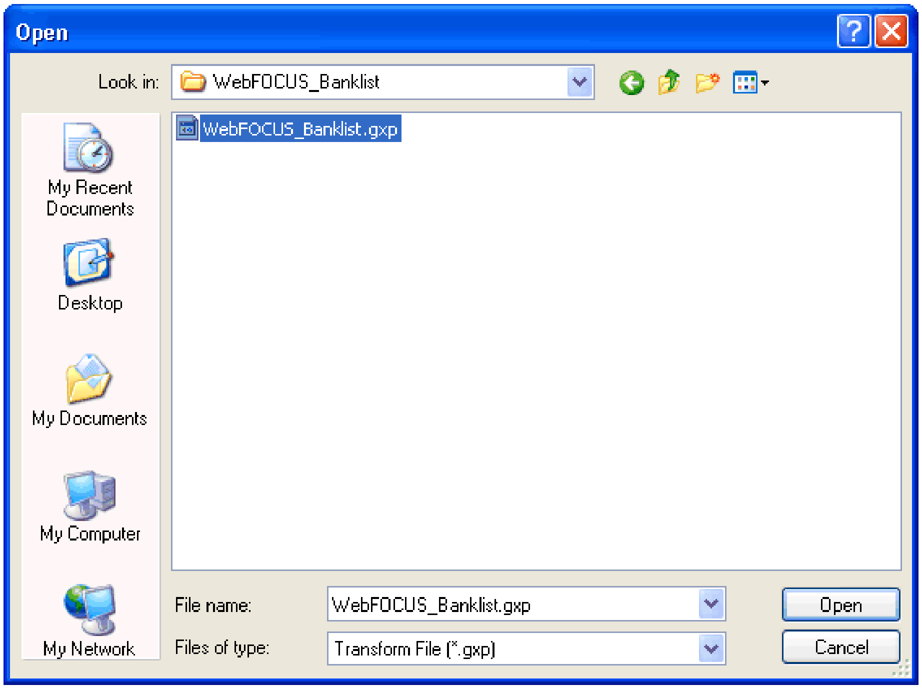
3. Expand the *iWay Integration* folder, select *Transform*, and click *Next*.

The Transform Import Wizard opens, as shown in the following image.



4. Click the ellipsis button to the right of the Import field.

The Open dialog box is displayed, as shown in the following image.



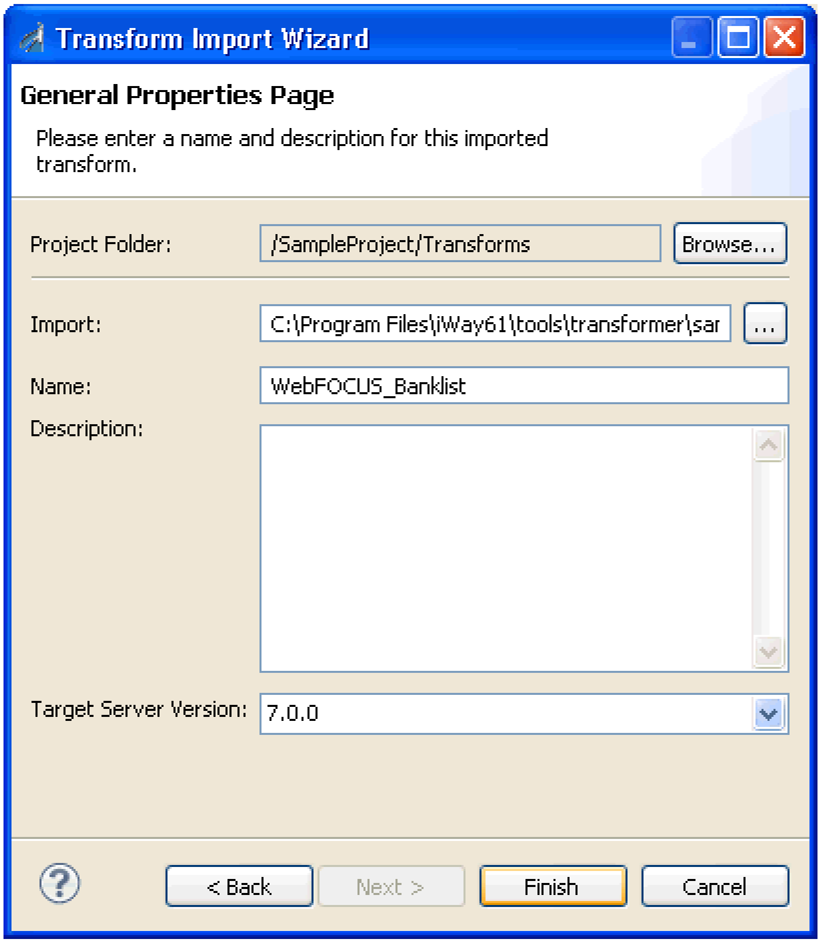
WebFOCUS_Banklist.gxp

File name: WebFOCUS_Banklist.gxp

Files of type: Transform File (*.gxp)

5. Browse to the location of the Transform component (.gxp file extension) on your file system, and click *Open*.

You are returned to the Transform Import Wizard, as shown in the following image.



General Properties Page

Please enter a name and description for this imported transform.

Project Folder: /SampleProject/Transforms

Import: C:\Program Files\iWay61\tools\transformer\sar

Name: WebFOCUS_Banklist

Description:

Target Server Version: 7.0.0

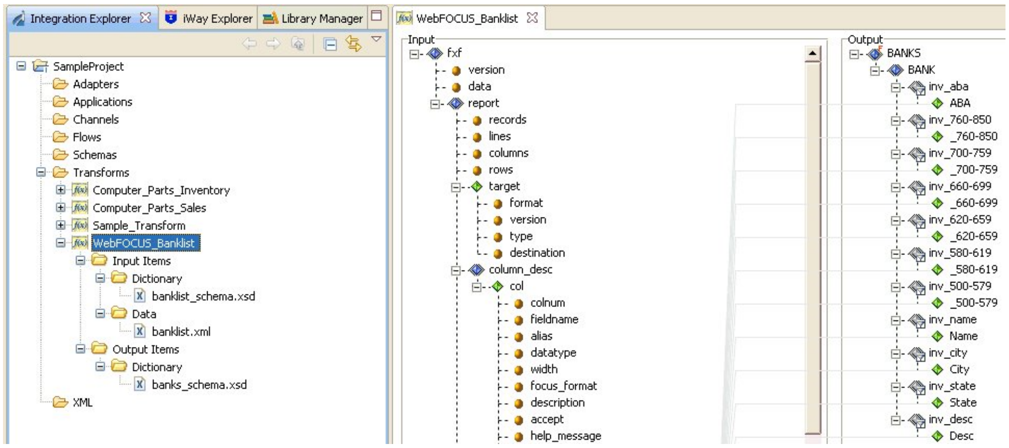
If required, you can change preselected values for the imported Transform component. Verify that the values specified in the following fields are accurate:

- **Project Folder.** The location to which the Transform component will be imported.
- **Import.** The location of the existing Transform component.
- **Name.** The name of the Transform component, as it will appear in the Integration Project.
- **Description.** The description of the Transform component, as it will appear in the Integration Project.

- ❑ **Target Server Version.** By default, this is the version of iWay Service Manager (iSM) that is installed in the iSM home directory.

6. Click *Finish*.

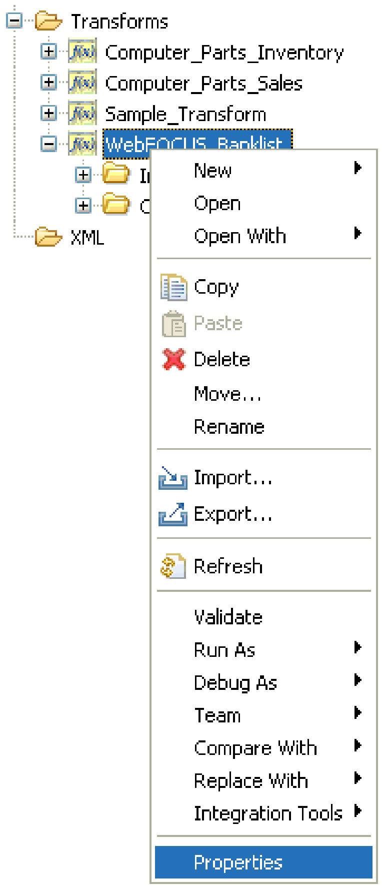
   The imported Transform component is loaded into the specified project folder location and displayed in iWay Transformer, as shown in the following image.



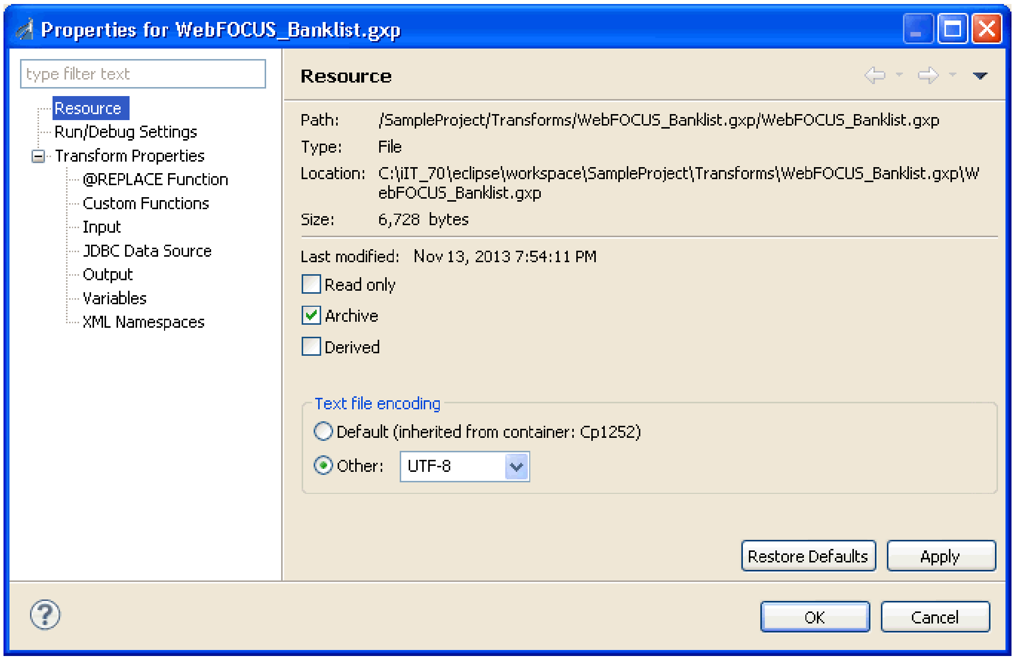
Configuring a Transform Component

The Properties dialog box enables you to view or modify the set of properties defined for a Transform component.

To access the Properties dialog box, right-click a Transform component, and select *Properties* from the context menu, as shown in the following image.



The Properties dialog box opens, as shown in the next image.



Resource

Path: /SampleProject/Transforms/WebFOCUS_Banklist.gxp/WebFOCUS_Banklist.gxp

Type: File

Location: C:\iIT_70\eclipse\workspace\SampleProject\Transforms\WebFOCUS_Banklist.gxp\WebFOCUS_Banklist.gxp

Size: 6,728 bytes

Last modified: Nov 13, 2013 7:54:11 PM

☐ Read only

☑ Archive

☐ Derived

Text file encoding

○ Default (inherited from container: Cp1252)

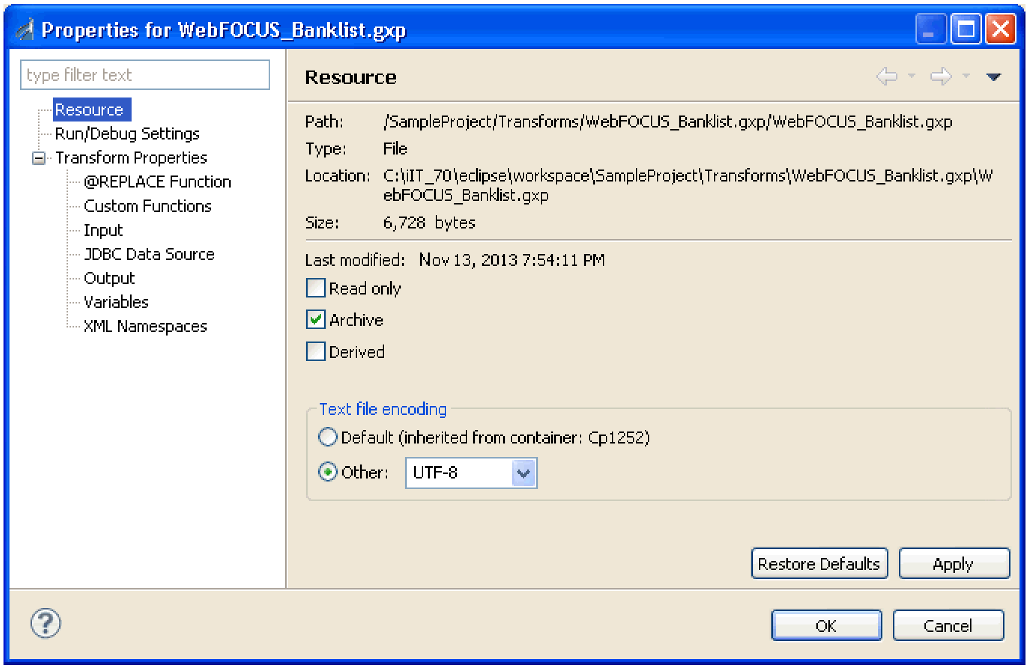
● Other: UTF-8

The Properties dialog box includes the following categories:

- **Resource.** For more information, see *Resource* on page 167.
- **Run/Debug Settings.** For more information, see *Run/Debug Settings* on page 168.
- **Transform Properties.** For more information, see *Transform Properties* on page 169.
  - **@REPLACE Function.** For more information, see *@REPLACE Function* on page 170.
  - **Custom Functions.** For more information, see *Custom Functions* on page 171.
  - **Input.** For more information, see *Input* on page 172.
  - **JDBC Data Source.** For more information, see *JDBC Data Source* on page 193.
  - **Output.** For more information, see *Output* on page 193.
  - **Variables.** For more information, see *Variables* on page 210.
  - **XML Namespaces.** For more information, see *XML Namespaces* on page 219.

Resource

The Resource category displays system-level information about the Transform component resource file, as shown in the following image.



Properties for WebFOCUS_Banklist.gxp

Resource

Path: /SampleProject/Transforms/WebFOCUS_Banklist.gxp/WebFOCUS_Banklist.gxp
Type: File
Location: C:\iIT_70\eclipse\workspace\SampleProject\Transforms\WebFOCUS_Banklist.gxp\WebFOCUS_Banklist.gxp
Size: 6,728 bytes

Last modified: Nov 13, 2013 7:54:11 PM

☐ Read only
☑ Archive
☐ Derived

Text file encoding
○ Default (inherited from container: Cp1252)
● Other: UTF-8

The following information is available:

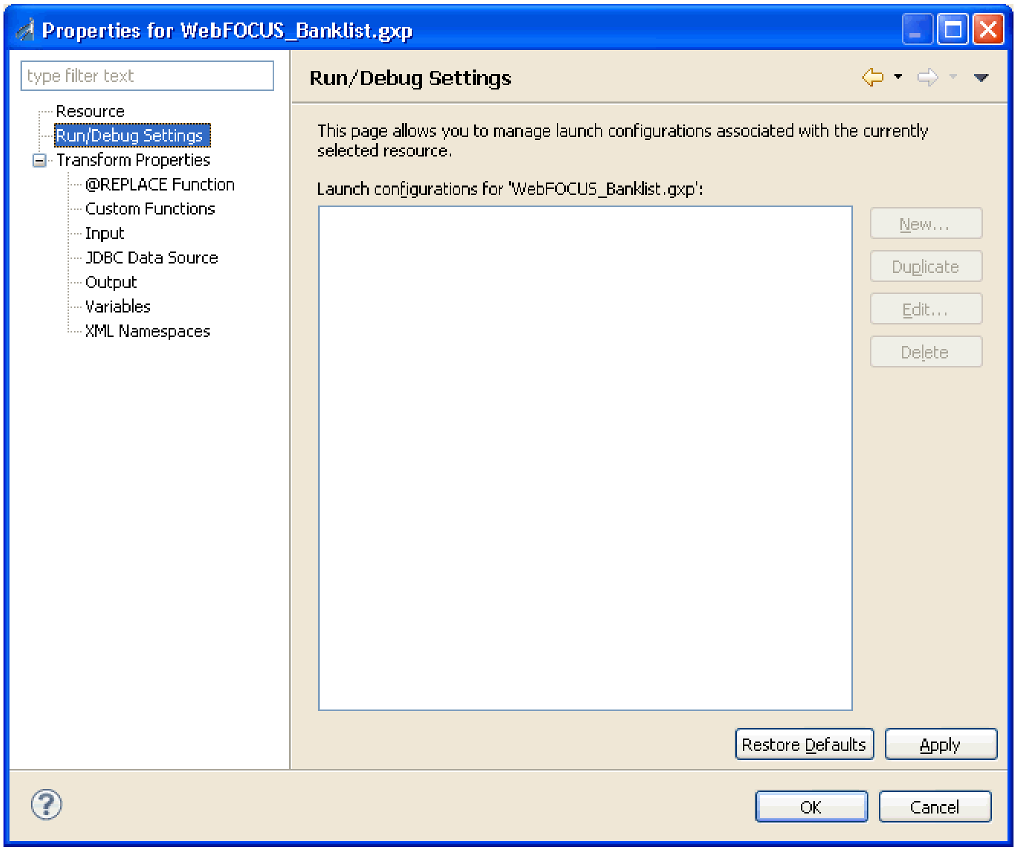
- Path
- Type
- Location
- Size
- Last Modified

You can set Read only, Archive, and Derived file access options for the resource if required. By default, the Archive option is set.

The Text file encoding section allows you to specify the type of encoding that is used for the characters in a file. By default, the character encoding is set to UTF-8.

Run/Debug Settings

The Run/Debug Settings category allows you to manage launch configurations that are associated with the resource that is currently selected. The Run/Debug Settings pane is shown in the following image.

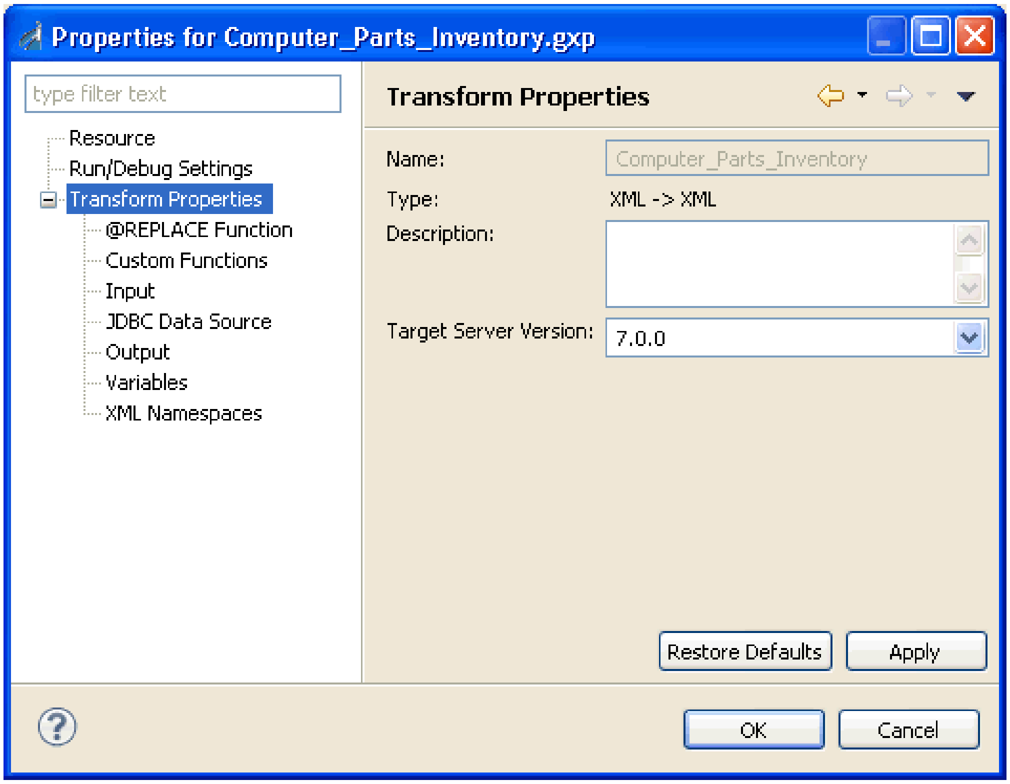


This page allows you to manage launch configurations associated with the currently selected resource.

Launch configurations for 'WebFOCUS_Banklist.gxp':

Transform Properties

The Transform Properties category includes basic information about the Transform component, as shown in the following image.



Name: Computer_Parts_Inventory
Type: XML -> XML
Description:
Target Server Version: 7.0.0

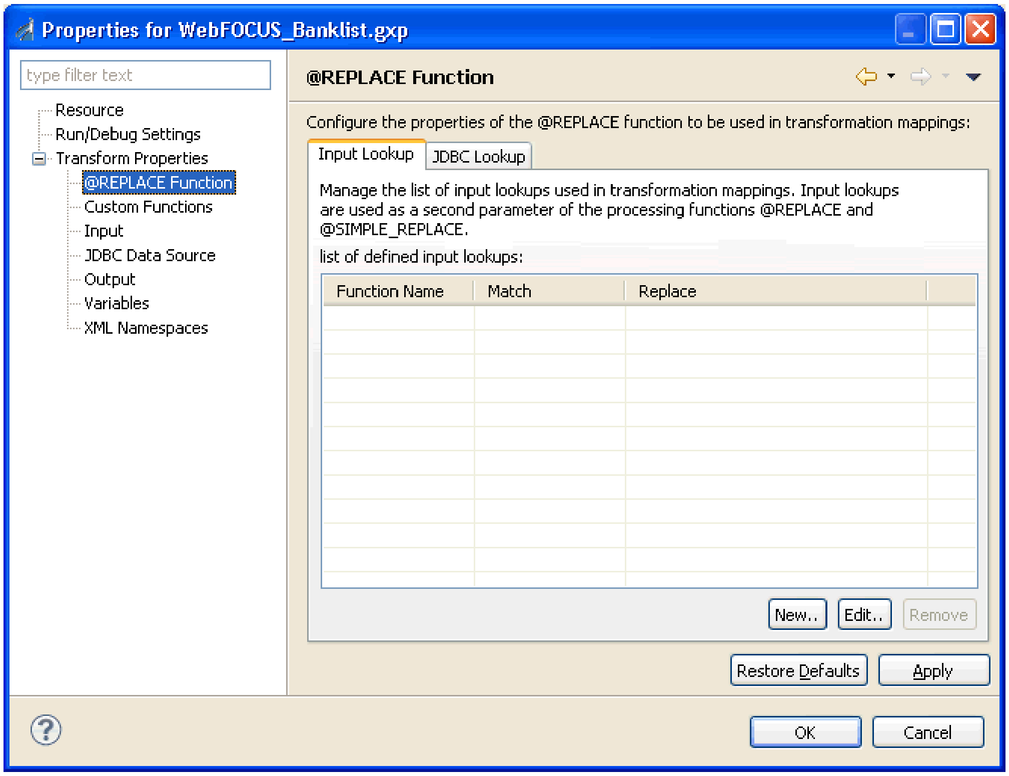
The following properties are listed:

- **Name.** Name of your Transform component.
- **Type.** Input and output format of your Transform component. If the format of the input or output is changed, the Type property will reflect that change.
- **Description.** Custom description for your Transform component.
- **Target Server Version.** Target version of iWay Service Manager (iSM) that is selected for this Transform component.

@REPLACE Function

The @REPLACE Function category provides a way to instantaneously match and replace certain input data values. Each individually configured replace function works similarly to a custom function in that you must first define the function, and then apply it in the output node mapping value definition that you want to affect.

The following image shows the @REPLACE Function pane.

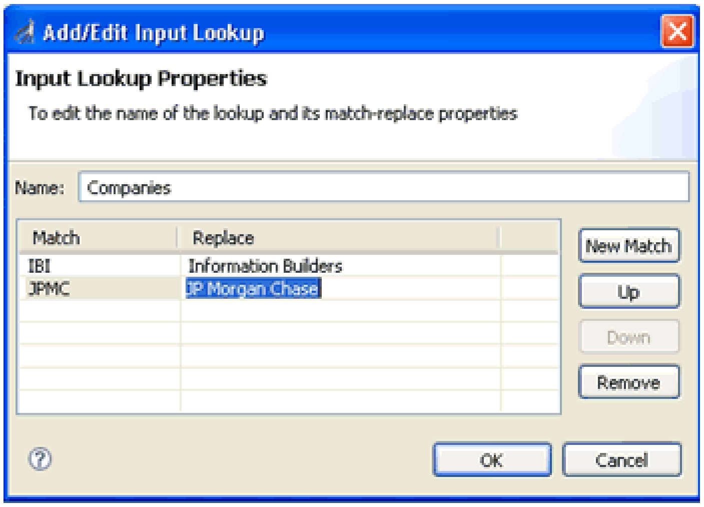


Configure the properties of the @REPLACE function to be used in transformation mappings:

Manage the list of input lookups used in transformation mappings. Input lookups are used as a second parameter of the processing functions @REPLACE and @SIMPLE_REPLACE.

list of defined input lookups:

Click *New Match* to define an input lookup for the replace function.



Add/Edit Input Lookup

Input Lookup Properties

To edit the name of the lookup and its match-replace properties

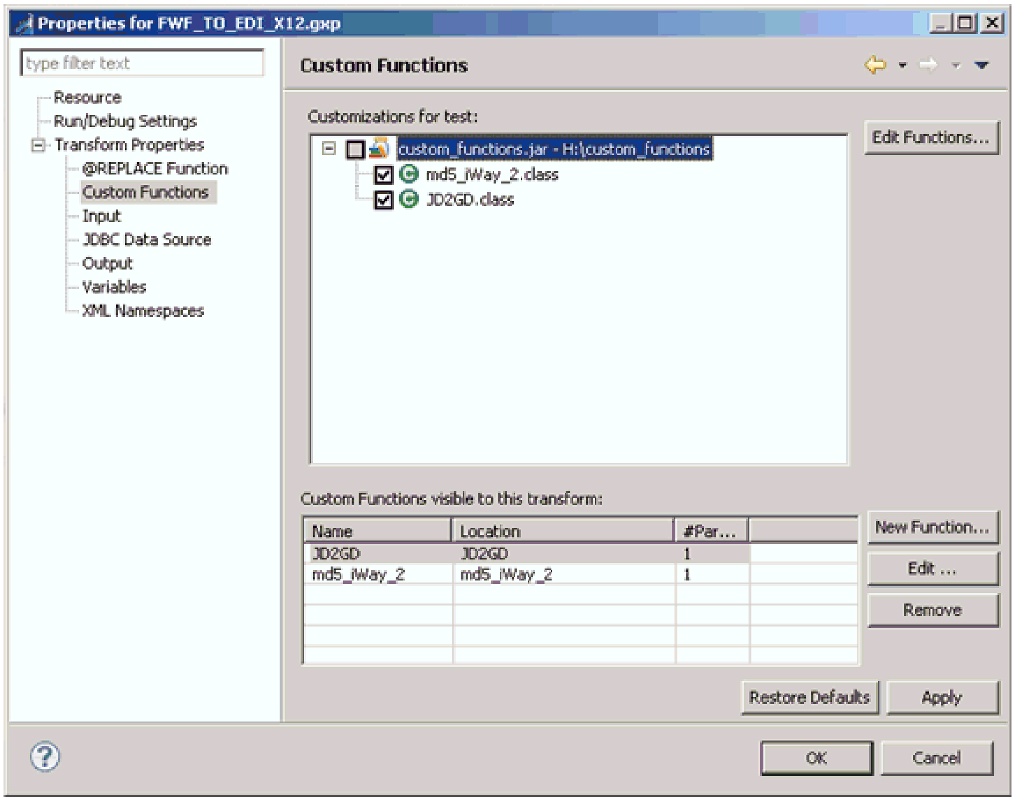
Name: Companies

Match	Replace
IBI	Information Builders
JPMC	JP Morgan Chase

Custom Functions

The Custom Functions category enables you to manage the list of custom functions that you can build on-demand, when a predefined iWay Transformer function does not exist to perform the task that you require.

The following image shows the Custom Functions pane.



Customizations for test:

- custom_functions.jar - H:\custom_functions
  - md5_iWay_2.class
  - JD2GD.class

Custom Functions visible to this transform:

Name	Location	#Par...
JD2GD	JD2GD	1
md5_iWay_2	md5_iWay_2	1

Custom functions must be written using Java and saved as Java class files (.class) in the following directory to make them available for use with iWay Transformer during design time.

`*iWaySMHome*\tools\transformer\custom_functions`

where:

iWaySMHome

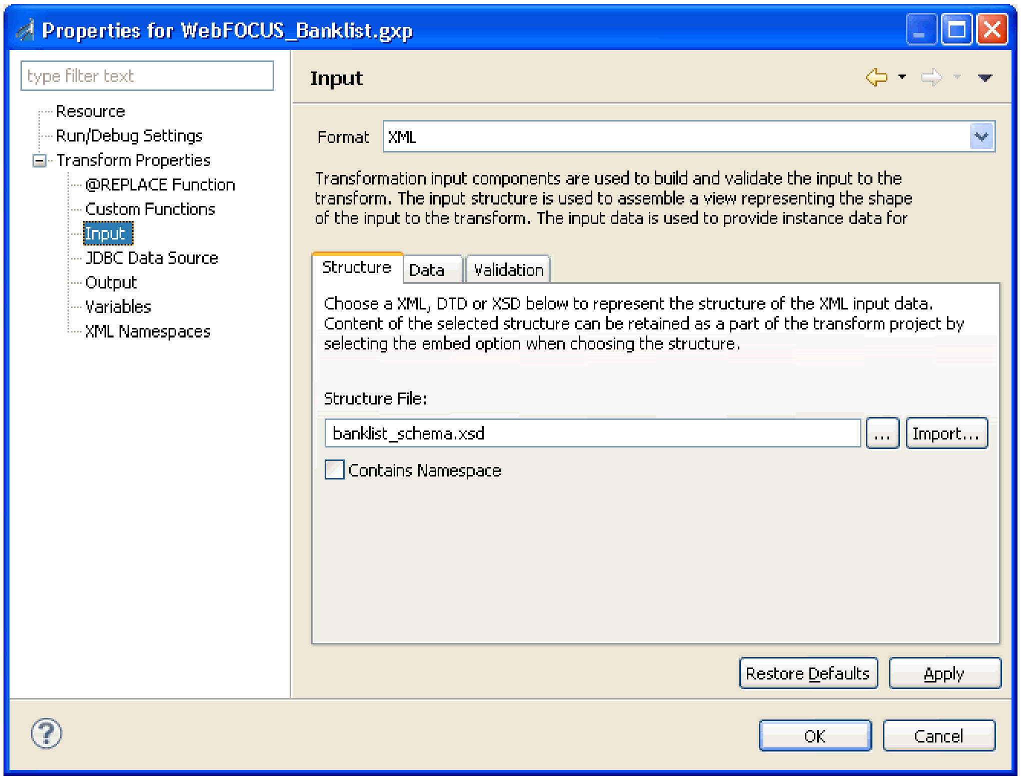
Is the directory in which iWay Service Manager is installed.

You must configure custom functions differently for use at run time.

Input

The Input category allows you to configure the input properties for the Transform component.

The following image shows the Input pane, with the Structure tab selected.



Format XML

Transformation input components are used to build and validate the input to the transform. The input structure is used to assemble a view representing the shape of the input to the transform. The input data is used to provide instance data for

Choose a XML, DTD or XSD below to represent the structure of the XML input data. Content of the selected structure can be retained as a part of the transform project by selecting the embed option when choosing the structure.

Structure File: banklist_schema.xsd

Contains Namespace

Input Structure

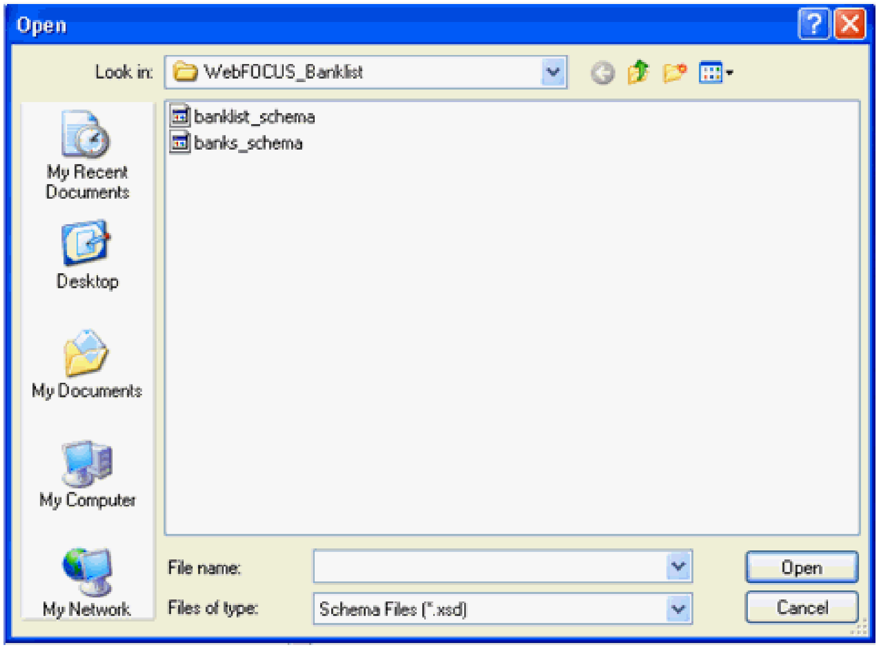
The Structure tab allows you to configure the dictionary, also called the structure, for your input.

For more information on using dictionaries, see *Dictionary* on page 95.

Configuring the Input Structure

On the Structure tab, you can specify metadata components by clicking the *Import* button to locate the file on your file system, or by typing the name of the file in the component field. You can also load the structure from the workspace using the Browse ... button.

The following image shows the Open dialog box that is displayed when you click the *Import* button next to a dictionary component field.

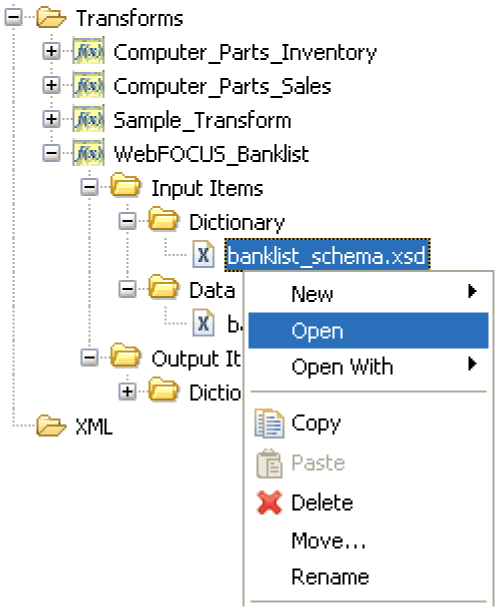


banklist_schema
banks_schema

File name:
Files of type: Schema Files (*.xsd)

Viewing the Input Structure

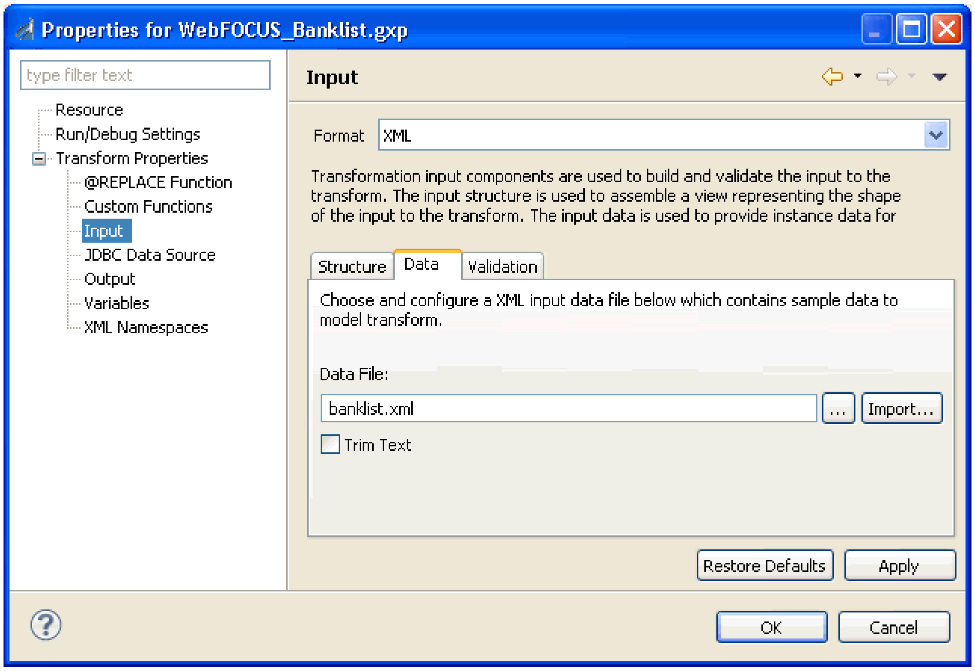
You can view the input structure using the context menu. Expand the structure folder, and double-click the name of the structure or select *Open* from the context menu, as shown in the following image.



Input Data

The Data tab allows you to select and configure an input data file, which contains the sample incoming document for a Transform component.

The following image shows the Input pane, with the Data tab selected.



Format XML

Transformation input components are used to build and validate the input to the transform. The input structure is used to assemble a view representing the shape of the input to the transform. The input data is used to provide instance data for

Choose and configure a XML input data file below which contains sample data to model transform.

Data File: banklist.xml

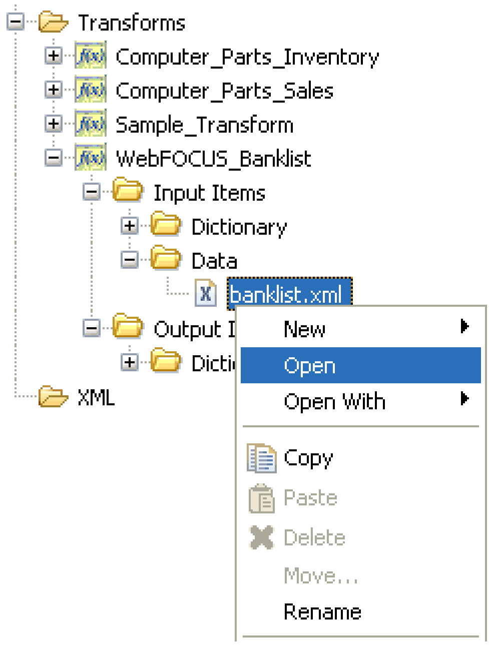
Trim Text

Configuring the Input Data

On the Data tab, you can specify the data file that you want to use to test the Transform component. For more information, see *Testing a Transform Component* on page 219. You can also overwrite the default data configuration for certain formats. For more information, see *Input Format Reference* on page 179.

Viewing the Input Data

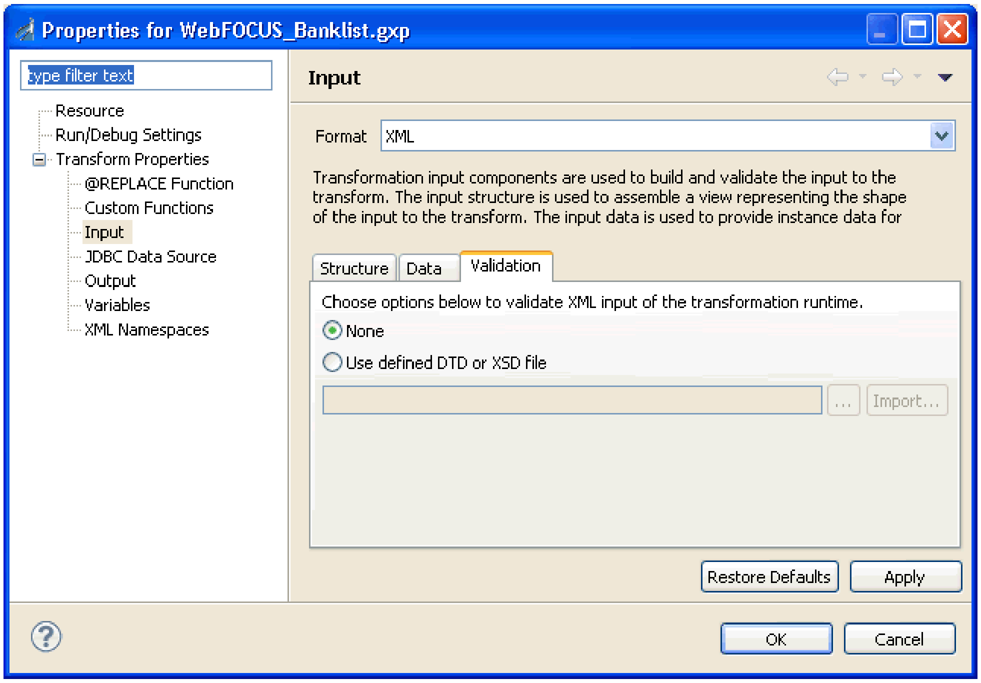
To view the input data, double-click the data file, or right-click the data file and select *Open* from the context menu, as shown in the following image.



Input Validation

The Validation tab allows you to specify options for validating an incoming document at run time against a specific XML schema or a DTD file. You can also validate the document against the rules contained in a Transform dictionary. We recommend that you use a schema for validation, rather than a DTD file.

The following image shows the Input pane, with the Validation tab selected.


Properties for WebFOCUS_Banklist.gxp

type filter text

- Resource
- Run/Debug Settings
- Transform Properties
  - @REPLACE Function
  - Custom Functions
  - Input
  - JDBC Data Source
  - Output
  - Variables
  - XML Namespaces

Input

Format: XML

Transformation input components are used to build and validate the input to the transform. The input structure is used to assemble a view representing the shape of the input to the transform. The input data is used to provide instance data for

Structure | Data | Validation

Choose options below to validate XML input of the transformation runtime.

[✓ None]

Use defined DTD or XSD file

For more information, see *Input Format Reference* on page 179.

Input Format Reference

The following table describes the major concepts and node types used to represent input items.

Input Document

This is the incoming document or message to which a transformation applies. The following formats are supported:

- CSV. For more information, see *CSV Input Properties* on page 183.
- EDI HIPAA. For more information, see *EDI Input Properties* on page 184.
- EDI X12. For more information, see *EDI Input Properties* on page 184.
- EDIFACT. For more information, see *EDI Input Properties* on page 184.
- Fixed Width. For more information, see *Fixed Width Input Properties* on page 187.
- IDOC. For more information, see *IDOC Input Properties* on page 189.
- SWIFT. For more information, see *SWIFT Input Properties* on page 190.
- XML. For more information, see *XML and iWay XML Response Input Properties* on page 191.
- iWay XML Response. For more information, see *XML and iWay XML Response Input Properties* on page 191.
- CDF. For more information, see *CDF Input Properties* on page 182. **Note:** CDF format is deprecated. iWay Software recommends that you use Fixed Width format instead.

The input document is displayed on the Input pane as a logical tree of component items, such as groups or elements. For more information, see the Input Document Tree description that follows. Each input item contains a name that identifies the type of item, optionally a number of attributes, or content.

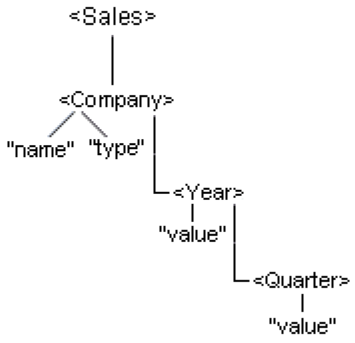
Input Document Tree

This is a sample tree of groups, elements, and attributes encoded in the input document. Each item in this tree has exactly one parent, and can have more than one child.

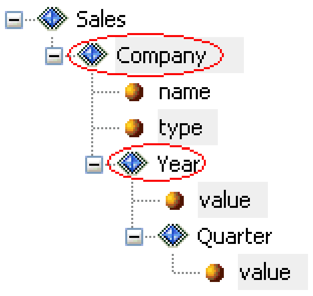
The following is a section of an XML input document:

```
<?xml version="1.0" encoding="UTF-8"?>
<Sales>
        <Company name="Video and Sound Card Express" type="Computer Parts">
                <Year value="2001">
                        <Quarter value="1st"/>
                </Year>
        </Company>
</Sales>
```

The following is a tree structure representation of the preceding code:



In iWay Transformer, the resulting input document tree is as follows:



Group ◆

This is a block of data that has a group element as its root. It can contain other groups or elements nested within, as children. Multiple groups can also exist on the same level. A group was formerly called a parent.

Element ◆

An element usually belongs to the group as a leaf, which cannot have any other nested groups or elements.

Attribute ●

This is a value associated with a group or element. It consists of a name and an associated textual value. Attributes are used only for XML input. Each group or element can have zero to many attributes.

Parent

A parent is a group in relation to the elements or groups contained within it. The input can have more than one parent (root group). In the previous example, the group Company is a parent of the group Year.

Child

Node A is called a child of node B, if and only if B is the parent of A. In the previous example, the element Year is a child of the group Company.

Descendant

Node A is called a descendant of node B, if either (1) A is a child of B, or (2) A is the child of some node C that is a descendant of B.

Ancestor

Node A is called an ancestor of node B, if and only if B is a descendant of A.

Sibling

Node A is called a sibling of node B, if and only if B and A share the same parent. Node A is a preceding sibling if it comes before B in the input document tree. Node B is a following sibling if it comes after A in the input document tree.

CDF Input Properties

The tables in this topic describe the tabs and fields for Common Data Format (CDF) input. It is a conceptual data abstraction for storing, accessing, and manipulating multidimensional data sets.

Note: CDF format is deprecated. iWay Software recommends that you use Fixed Width format instead.

Structure tab: Allows you to configure the dictionary for the CDF input data using the Dictionary Builder interface.

Field	Description

Structure	Identifies the structure component used for the input dictionary of the Transform component.
	
	To locate the file, type the file name, or click the *Browse* ... button.
	
	The following is a sample CDF structure file, which is created when you click the *New* button.
	
	`<TransformCDFlayout>`
	`  <RecordHeader RecordCount="1" Mode="Ignore" LineFeed="nl"`
	`       Format="ASCII">`
	`    <Column Name="Name" StartOffset="1" Length="8" />`
	`    <Column Name="Title" StartOffset="10" Length="17" />`
	`  </RecordHeader>`
	`  <RecordDetails>`
	`    <RecordLayout Type="20" StartOffset="1" Length="2">`
	`      <Column Name="Column1" StartOffset="1" Length="2"/>`
	`      <Column Name="Column2" StartOffset="4" Length="7"/>`
	`      <Column Name="Column3" StartOffset="12" Length="6"/>`
	`    </RecordLayout>`
	`  </RecordDetails>`
	`</TransformCDFlayout>`
	
	You can use the CDF Dictionary Editor tool to modify the base structure.

Data tab: Allows you to select and configure a CDF input data file, which contains the sample incoming document for your Transform component.

Field	Description
Data File	To locate the file, type the file name, or click the *Browse* button.

Field	Description
Decode data content	To decode data content from EBCDIC format, select this check box. This check box is not selected by default.
content	This check box is not selected by default.

CSV Input Properties

The tables in this topic describe the tabs and fields for the Comma-Separated Values (CSV) input format. In a CSV file, each block of data is separated by a comma. However, iWay Transformer allows for other delimiters. For more information, see the following description of the Delimiter field on the Data tab.

Structure tab: Allows you to configure the dictionary for the CSV input data.

Field	Description
Structure	Identifies the structure component used for the input dictionary of the Transform component. Typically, this is either the input data file itself, or a CSV file that has an identical structure to the intended input data file. To locate the file, type the file name, or click the *Browse* ... button.
Restore Defaults	Clears the Structure field.

Data tab: Allows you to select and configure a CSV input data file, which contains the sample incoming document for your Transform component.

Field	Description
Data File	To locate the file, type the file name, or click the *Browse* button.
Delimiter	Specifies the delimiter character, which separates the elements of the CSV data. The default value is a comma (,). iWay Transformer can also process files that use other delimiter characters. The files must correspond to the CSV format.

Field	Description

Header is included in the data	You can select this check box if the header is included in the document. This check box is selected by default. The header typically contains column names.
	
	The following excerpt from a sample input.csv file includes a header with the data. The header is in the first line.
	
	`Country,"Province-Territory Name","Population (in 1000's)","Area Size (in km2)","Type"`
	
	`Canada,"Alberta","3097.5","661185","Province"`
	`Canada,"British Columbia","4092.8","948596","Province"`
	`.`
	`.`
	`.`

Restore Defaults	Applies the default values to the Data tab by clearing the *Data File* field, selecting the *Header is included in the data* check box, and setting the *Delimiter* to a comma (,).

EDI Input Properties

The tables in this topic describe the tabs and fields for the Electronic Data Interchange (EDI) input formats, the following of which are created in a similar fashion:

- EDI HIPAA (Health Insurance Portability and Accountability Act).
- EDI X12. This data format is based on ASC X12 standards. It is used to exchange specific data between two or more trading partners.
- EDIFACT (Electronic Data Interchange For Administration, Commerce, and Transport).

You can edit the data dictionary for EDI input using the Dictionary Builder, and you can store the components in the library for future reuse.

Structure tab: Allows you to configure the metadata for the EDI message.

Field	Description
Header	Identifies the header component used for the input dictionary of the Transform component. The header contains envelope details that include trading partner information and message layout. To locate the file, type the file name, or click the *Browse* … button.
Structure	Identifies the structure component used for the input dictionary of the Transform component. The structure contains the layout and grammar of the document (or transaction of the EDI message). To locate the file, type the file name, or click the *Browse* button.

Data tab: Allows you to select and configure an EDI input data file, which contains a sample incoming document for your Transform component.

Field	Description
Segment Delimiter	Specifies the character that indicates the end of a segment. The default values are: ❑ *7E ~ Tilde* for EDI HIPAA and EDI X12 formats. ❑ *None* for EDIFACT format.
Segment Suffix	Used in combination with a segment delimiter, the segment suffix indicates the end of a segment. You can select a predefined segment suffix from the drop-down list. The default values are: ❑ *None* for EDI HIPAA and EDI X12 formats. ❑ *0A LF Line Feed* for EDIFACT format.

Element Delimiter	Specifies the character that indicates the end of an element. You can select a predefined element delimiter character from the drop-down list. The default values are: ❑ *2A * Asterisk* for EDI HIPAA and EDI X12 formats. ❑ *2B + Plus* for EDIFACT format.
Component Element Delimiter	Specifies the character that indicates the end of a component element. You can select a predefined component element delimiter character from the drop-down list. By default, *3A : Colon* is used for EDI HIPAA, EDI X12, and EDIFACT formats.
Escape Character	Specifies the escape character that is used when reading the input data. You can select a predefined escape character from the drop-down list. The default values are: ❑ *5C \ Backslash* for EDI HIPAA and EDI X12 formats. ❑ *3F ? Question Mark* for EDIFACT format.
Restore Defaults	Applies default values to the Data tab.

Validation tab: Allows you to specify options that validate EDI input for the transformation run time.

Field	**Description**
None	To skip validation, select this option. This option is selected by default.
Use validation rule defined by the project's input structure	To validate the incoming document using the rule defined by the Transform component input dictionary, select this option. This option is not selected by default. For EDI HIPAA and EDI X12, selecting this option activates the Ignore NTE Segment check box.

Field	Description
Ignore NTE Segment	For EDI HIPAA and EDI X12, select this check box if you do not want to validate mapping rules specific to NTE. This check box is not selected by default. This option allows NTE segment translations to appear in the XML document in the same sequence as that of the input document. However, if this option is not checked, specific rules can be applied in the dictionary to position NTE segment translations accordingly. NTE is a floating segment that can occur in any place within a document. It is applicable to versions prior to EDI X12 version 4010. This option is not available for EDIFACT.
Restore Defaults	Applies the default values to the Validation tab by selecting the None option.

Fixed Width Input Properties

The tables in this topic describe the tabs and fields for the Fixed Width input format, also called FWF. Fixed Width files are flat files that contain fixed width data fields that constitute records. Records are commonly separated by new line characters. Fixed Width format is very similar to the deprecated CDF format, except for features that are applicable to Fixed Width only, such as looping.

Structure tab: Allows you to select an XML file that represents the data dictionary for the Fixed Width input data.

Field	Description

Structure	Identifies the metadata component used for the input data dictionary of the Transform component.
	
	To locate the file, type the file name, or click the *Browse* button.
	
	The following is a sample Fixed Width structure file, which is created when you click the *New* button.
	
	`<FIXED_WIDTH>`
	`  <RecordHeader>`
	`    <Column Name="DESCRIPTION" StartOffset="1"`
	`     Length="10" />`
	`  </RecordHeader>`
	`  <RecordDetails>`
	`     <Loop ID="Loop" Req="M" Min="1" Max="1">`
	`       <RecordLayout Type="10" StartOffset="1"`
	`        Length="2">`
	`           <Column Name="Column1" StartOffset="3"`
	`            Length="5" />`
	`           <Column Name="Column2" StartOffset="8"`
	`            Length="5" />`
	`       </RecordLayout>`
	`     </Loop>`
	`  </RecordDetails>`
	`</FIXED_WIDTH>`

Data tab: Allows you to select and configure a Fixed Width input data file, which contains a sample incoming document for your Transform component.

Field	Description
Data File	To locate the file, type the file name, or click the *Browse* button.
Record Delimiter	Specifies delimiter options for the input data file. The record delimiter is a character that defines the boundary between records. The default value is a carriage return.
Trim Columns	Selected by default, this option is used to trim columns from the input data file.
Restore Defaults	Clears all fields on the Data tab, and selects the Trim Columns check box.

IDOC Input Properties

The tables in this topic describe the tabs and fields for the SAP Intermediate Documents (IDOC) input format. IDOCs represent a standard data structure for electronic data interchange between application programs written for SAP systems or between an SAP application and an external program.

You can access IDOCs using the iWay Application Adapter for mySAP ERP in iWay Explorer. For more information, see the *iWay Application Adapter for mySAP ERP User's Guide*.

iWay Transformer does not support the collected IDOC files, due to the structural mapping. You must split the IDOC batch documents into the individual IDOC files in order to process them correctly.

Structure tab: Allows you to select a file that represents the data dictionary for the IDOC input data.

Field	Description
Structure	Identifies the structure component used for the input dictionary of the Transform component. To locate the file, type the file name, or click the *Browse* ... button. The following is an excerpt from a sample structure file named structure.txt. It is included in the samples provided with the product.

```
BEGIN_RECORD_SECTION
BEGIN_CONTROL_RECORD
BEGIN_FIELDS
NAME                     TABNAM
TEXT                     Name of table structure
TYPE                     CHARACTER
LENGTH                   000010
FIELD_POS                0001
BYTE_FIRST               000001
BYTE_LAST                000010
NAME                     MANDT
TEXT                     Client
TYPE                     CHARACTER
LENGTH                   000003
FIELD_POS                0002
BYTE_FIRST               000011
BYTE_LAST                000013
.
.
.
```

Field	Description
Restore Defaults	Clears the Structure field.

Data tab: Allows you to select and configure an IDOC input data file, which contains a sample incoming document for your Transform component.

Field	Description
Data File	To locate the file, type the file name, or click the *Browse* button.
Restore Defaults	Clears the Data File field.

SWIFT Input Properties

The tables in this topic describe the tabs and fields for the Society for Worldwide Interbank Financial Telecommunication (SWIFT) input format. SWIFT is a messaging and transaction processing format used by worldwide financial organizations.

Structure tab: Allows you to select a data dictionary that represents the SWIFT input data.

Field	Description
Structure	Identifies the structure component used for the input dictionary of the Transform component. To locate the file, type the file name, or click the *Browse* ... button.

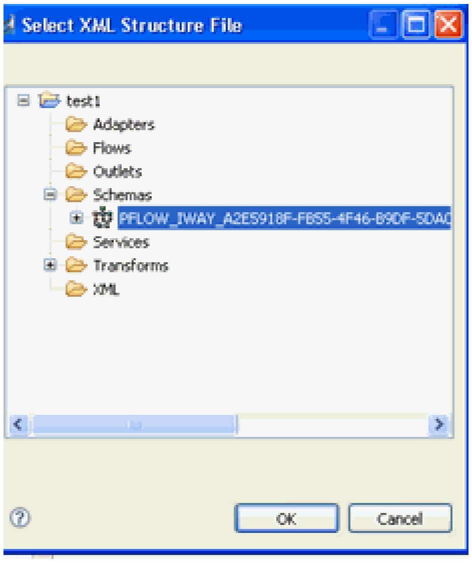
Data tab: Allows you to select and configure a SWIFT input data file, which contains a sample incoming document for your Transform component.

Field	Description
Data File	To locate the file, type the file name, or click the *Import* button. You can also load the file from your workspace using the *Browse* button.
Restore Defaults	Clears the Data File field.

XML and iWay XML Response Input Properties

The tables in this topic describe the tabs and fields for the XML input formats, which are called XML and iWay XML Response. iWay XML Response is the commonly defined document format used in iWay Service Manager data integration services.

Structure tab: Allows you to select an XML, DTD, or XSD (schema) file that represents the data dictionary for the XML input data.

Field	Description
Structure	Identifies the structure component used for the input dictionary of the Transform component. It is typically represented as an XSD (schema), DTD, or XML file that describes the incoming message. To locate the file, type the file name, or click the *Browse* ... button to open the following dialog box.  Select XML Structure File test1 Adapters Flows Outlets Schemas PFLOW_IWAY_A2E5918F-FB55-4F46-B9DF-5DAC Services Transforms XML OK Cancel Navigate to the schema reference within the project structure, and click *OK*. You can load a schema from the file system by clicking the *Import* button.

Field	Description
Contains Namespace	Select this check box if the input contains a namespace. This check box is not selected by default. For more information, see *Working With Namespaces* on page 265.

Data tab: Allows you to select and configure an XML input data file, which contains a sample incoming document for your Transform component.

Field	Description
Data File	To locate the file, type the file name, or click the *Browse* button.
Trim Text	Removes all leading and trailing white space characters from the XML input data file.

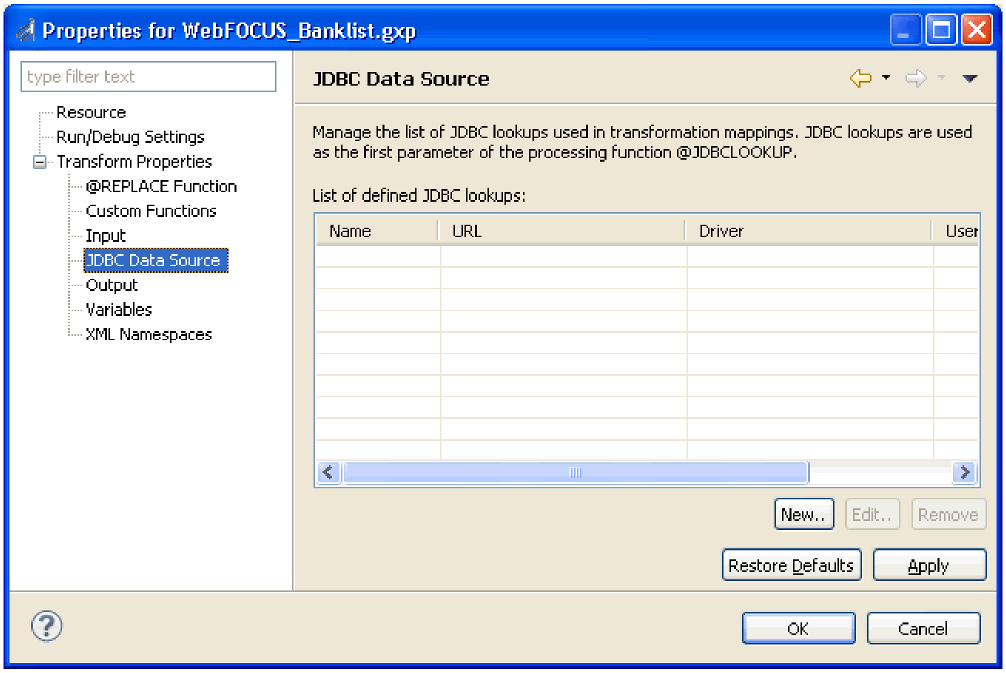
Validation tab: Allows you to specify options that validate the XML input against schemas or DTDs during transformation run time.

Field	Description
None	To skip validation, select this option. This option is selected by default.
Use validation rule defined in the input data	To perform validation using the dictionary defined in the input data, select this option. This option is not selected by default.
Use user-defined DTD or XSD file	To validate the incoming document using an external dictionary, select this option. To locate the dictionary component, XSD (schema), or DTD, type the file name, or click the *Browse* button. For a dictionary component, we recommend that you use schemas instead of DTDs, because an updated parser is available.
Restore Defaults	Applies the default values to the Validation tab by selecting the None option.

JDBC Data Source

The JDBC Data Source category enables you to manage the list of JDBC lookups used by the @JDBCLOOKUP function during transformation mappings. You can define multiple JDBC connections or add a connection from an existing project. In addition, you can specify any predefined function (for example, @SREG) as the URL. For more information, see *Working With Functions* on page 269.

The following image shows the JDBC Data Source pane.



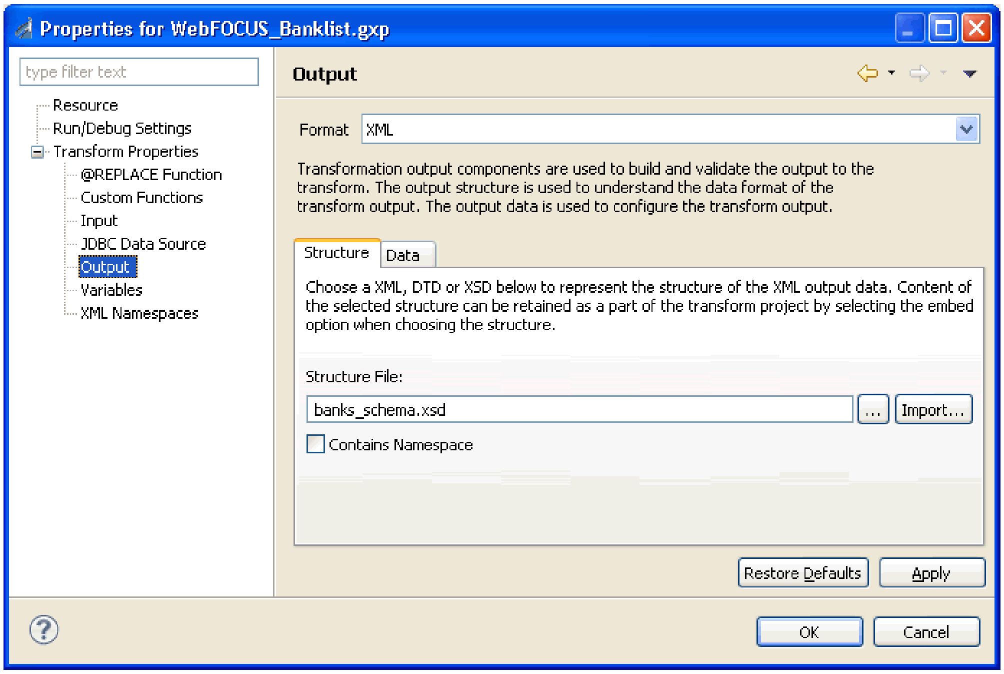
Manage the list of JDBC lookups used in transformation mappings. JDBC lookups are used as the first parameter of the processing function @JDBCLOOKUP.

List of defined JDBC lookups:

Output

The Output category allows you to configure the output properties for the Transform component, according to the expected output requirements, which typically describe the structure.

The following image shows the Output pane, with the Structure tab selected.



Format: XML

Transformation output components are used to build and validate the output to the transform. The output structure is used to understand the data format of the transform output. The output data is used to configure the transform output.

Choose a XML, DTD or XSD below to represent the structure of the XML output data. Content of the selected structure can be retained as a part of the transform project by selecting the embed option when choosing the structure.

Structure File: banks_schema.xsd

☐ Contains Namespace

Output Structure

The Structure tab allows you to configure the structure of your output.

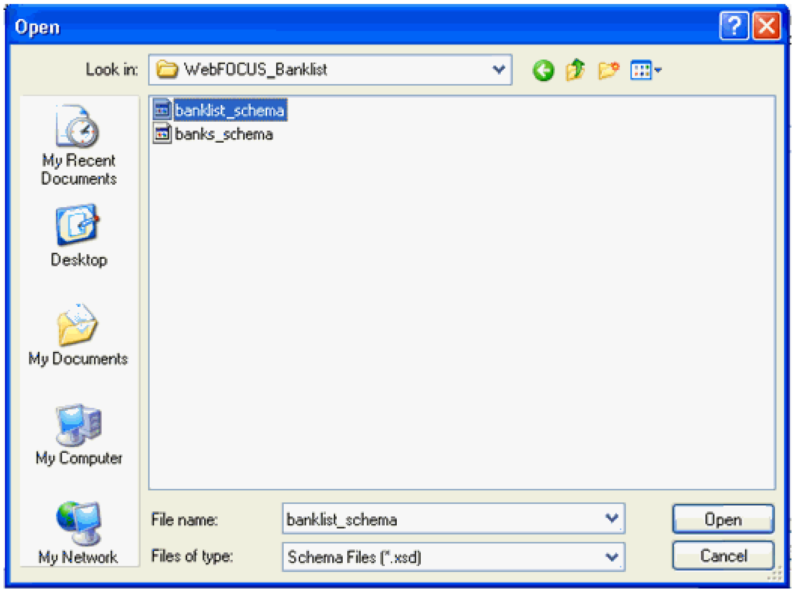
For more information on using dictionaries, see *Dictionary* on page 95.

Configuring the Output Structure

On the Structure tab, you can specify dictionary components by clicking the *Browse* button to locate the file on your file system, or by typing the name of the file in the component field.

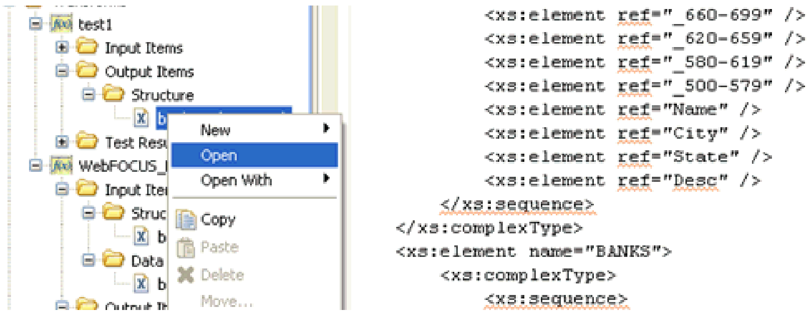
For supported library formats like EDI and SWIFT, the option to select a dictionary from the system library is provided. For XML and iWay XML Response input formats, the option to import a schema from the iWay Registry, an iWay Service Manager configuration, an iWay Web service, or an Ebix archive is provided. For more information, see *XML, iWay XML Embedded Request, and iWay XML Request Output Properties* on page 207.

The following image shows the Open dialog box that is displayed when you click the *Import* button next to a dictionary component field, for example, Structure.



Viewing the Output Structure

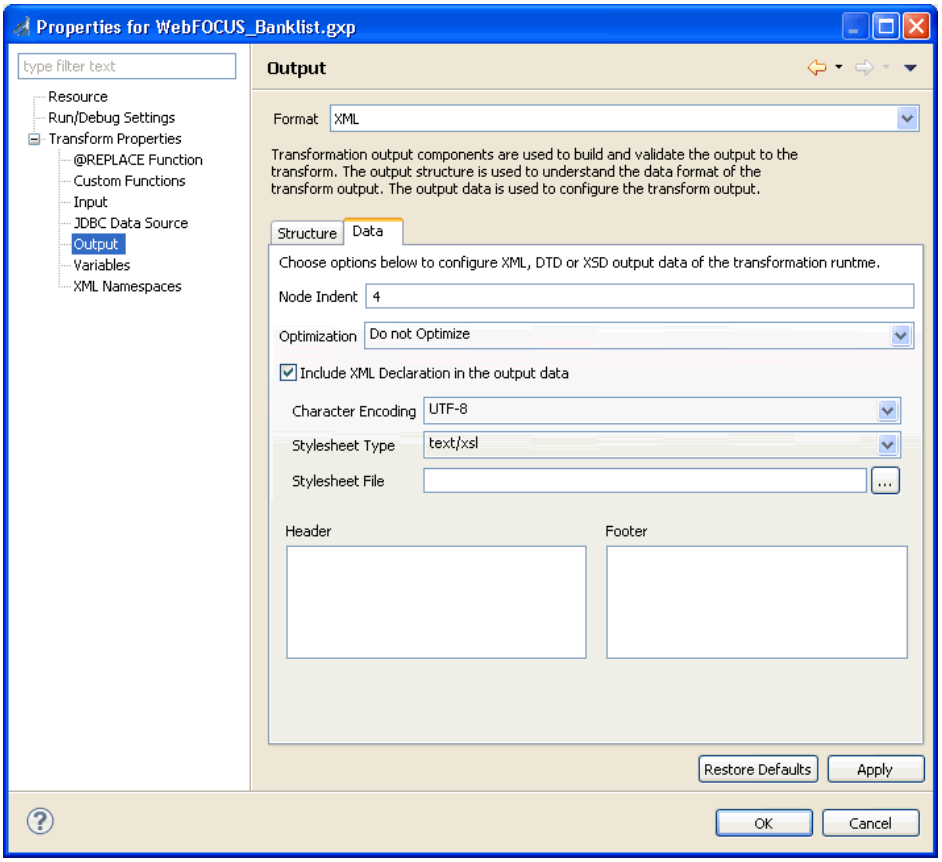
To view the output dictionary, double-click the data file, or right-click the data file and select *Open* from the context menu, as shown in the following image.



```
        <xs:element ref="_660-699" />
        <xs:element ref="_620-659" />
        <xs:element ref="_580-619" />
        <xs:element ref="_500-579" />
        <xs:element ref="Name" />
        <xs:element ref="City" />
        <xs:element ref="State" />
        <xs:element ref="Desc" />
    </xs:sequence>
</xs:complexType>
<xs:element name="BANKS">
    <xs:complexType>
        <xs:sequence>
```

Output Data

The Data tab allows you to configure output data. The following image shows the Output pane, with the Data tab selected.



Format: XML

Transformation output components are used to build and validate the output to the transform. The output structure is used to understand the data format of the transform output. The output data is used to configure the transform output.

Choose options below to configure XML, DTD or XSD output data of the transformation runtme.

Node Indent: 4

Optimization: Do not Optimize

[✓] Include XML Declaration in the output data

Character Encoding: UTF-8

Stylesheet Type: text/xsl

Stylesheet File:

Configuring the Output Data

On the Data tab, you can configure the output data for the Transform component that will be used during run time. For more information, see *Output Format Reference* on page 197.

Output Validation

The Validation tab allows you to specify options that validate the output document at run time. For more information, see *Output Format Reference* on page 197.

Output Format Reference

This topic provides a reference for the output formats supported by iWay Transformer.

CDF Output Properties

CDF format is deprecated. iWay Software recommends that you use Fixed Width format instead.

The tables in this topic describe the tabs and fields for Common Data Format (CDF) output. It is a conceptual data abstraction for storing, accessing, and manipulating multidimensional data sets.

Structure tab: Allows you to configure the structure for the CDF output data.

Field	Description

Structure	Identifies the structure component used for the output dictionary of the Transform component.
	
	To locate the file, type the file name, or click the *Browse* ... button or *Import* button.
	
	The following is a sample CDF structure file, which is created when you click the *New* button.
	
	`<TransformCDFlayout>`
	`  <RecordHeader RecordCount="1" Mode="Ignore" LineFeed="nl" Format="ASCII">`
	`   <Column Name="Name" StartOffset="1" Length="8" />`
	`   <Column Name="Title" StartOffset="10" Length="17" />`
	`  </RecordHeader>`
	`  <RecordDetails>`
	`   <RecordLayout Type="20" StartOffset="1" Length="2">`
	`   <Column Name="Column1" StartOffset="1" Length="2"/>`
	`   <Column Name="Column2" StartOffset="4" Length="7"/>`
	`   <Column Name="Column3" StartOffset="12" Length="6"/>`
	`   </RecordLayout>`
	`   </RecordDetails>`
	`</TransformCDFlayout>`

Data tab: Allows you to configure CDF output data.

Field	Description
Padding	Specifies the character used for padding empty spaces. If you leave this field blank, iWay Transformer uses a blank space for padding.

Align	Specifies the alignment of the output. Left-justified text (the default value) is aligned to the left side of each field. Right-justified text is aligned to the right side.
	
	For example, consider the following dictionary file:
	
	`<TransformCDFlayout>`
	`<RecordHeader RecordCount="1" Mode="Ignore"`
	`LineFeed="nl" Format="ASCII">`
	`    <Column Name="ReportTitle" StartOffset="1"`
	`     Length="2"/>`
	`</RecordHeader>`
	`  <RecordDetails>`
	`    <RecordLayout Type="NA" StartOffset="1"`
	`     Length="2">`
	`      <Column Name="MSG_ID            "`
	`       StartOffset="1" Length="8"/>`
	`      <Column Name="ITEM_UNIQUE_ID    "`
	`       StartOffset="9" Length="20"/>`
	`      <Column Name="ITEM_CASE_NUM     "`
	`       StartOffset="30" Length="18"/>`
	`      <Column Name="ORIGIN            "`
	`       StartOffset="49" Length="8"/>`
	`.`
	`.`
	`.`
	
	Left-aligned data is displayed as:
	
	`ABC 55    777    XXX`
	
	Right-aligned data is displayed as:
	
	`ABC     55     777 XXX`

Encode	To encode the data in EBCDIC format, select this check box. By default, this check box is not selected.
Restore Defaults	Applies the default values to the Data tab, by clearing the Padding field, selecting *Left* from the Align field drop-down list, and deselecting the *Encode* check box.

CSV Output Properties

The tables in this topic describe the tabs and fields for the Comma-Separated Values (CSV) output format. In a CSV file, each piece of data is separated by a comma. However, iWay Transformer allows for other delimiters. For more information, see the following description of the Delimiter field on the Data tab.

Structure tab: Allows you to configure the dictionary for the CSV output data.

Field	Description
Structure	Identifies the structure component used for the output dictionary of the Transform component (optional). To locate the file, type the file name, or click the *Browse* ... button.

Data tab: Allows you to configure CSV output data.

Field	Description
Delimiter	Sequence of one or more characters specifying the border between column fields. The default value is a comma (,). Although CSV stands for comma-separated values, iWay Transformer can process files that use other delimiter characters. The files must correspond to the CSV format.
Header is included in the data	You can select this check box if the header is included in the document. This check box is selected by default. The header typically contains column names. The following excerpt from a sample input.csv file includes a header with the data. The header is in the first line. `Country,"Province-Territory Name","Population (in 1000's)","Area Size (in km2)", "Type" Canada,"Alberta","3097.5","661185", "Province" Canada,"British Columbia","4092.8","948596","Province"` . . .

Field	Description
Always Add Quotes	To add quotation marks around the output values, select this check box. This check box is not selected by default.
Restore Defaults	Applies the default values to the Data tab, by setting the *Delimiter* to a comma (,), selecting the *Header is included in the data* check box, and deselecting the *Always Add Quotes* check box.

EDI Output Properties

The tables in this topic describe the tabs and fields for the Electronic Data Interchange (EDI) output formats, the following of which are created in a similar fashion:

- ❑ EDI HIPAA (Health Insurance Portability and Accountability Act).
- ❑ EDI X12. This is a data format based on ASC X12 standards. It is used to exchange specific data between two or more trading partners.
- ❑ EDIFACT (Electronic Data Interchange For Administration, Commerce, and Transport).

Structure tab: Allows you to configure the metadata for the EDI output message.

Field	Description
Header	Identifies the header component used for the output structure of the Transform component. The header contains envelope details that include trading partner information and message layout. To locate the file, type the file name, or click the *Browse* ... button.
Structure	Identifies the structure component used for the output dictionary of the Transform component. The structure contains the layout and grammar of the document (or transaction of the EDI message). To locate the file, type the file name, or click the *Browse* button.

Data tab: Allows you to configure an EDI output data file.

Field	Description
Segment Delimiter	Specifies the character that indicates the end of a segment. The default values are: ❏ *7E ~ Tilde* for EDI HIPAA and EDI X12 formats. ❏ *None* for EDIFACT format.
Segment Suffix	Used in combination with a segment delimiter, the segment suffix indicates the end of a segment. You can select a predefined segment suffix from the drop-down list. The default values are: ❏ *None* for EDI HIPAA and EDI X12 formats. ❏ *0A LF Line Feed* for EDIFACT format.
Element Delimiter	Specifies the character that indicates the end of an element. You can select a predefined element delimiter character from the drop-down list. The default values are: ❏ *2A * Asterisk* for EDI HIPAA and EDI X12 formats. ❏ *2B + Plus* for EDIFACT format.
Component Element Delimiter	Specifies the character that indicates the end of a component element. You can select a predefined component element delimiter character from the drop-down list. By default, *3A : Colon* is used for EDI HIPAA, EDI X12, and EDIFACT formats.

Field	Description
Escape Character	Specifies the escape character that is used when reading the output data. You can select a predefined escape character from the drop-down list. The default values are: ❑ *5C \ Backslash* for EDI HIPAA and EDI X12 formats. ❑ *3F ? Question Mark* for EDIFACT format.

Validation tab: Allows you to specify options that validate EDI output of the transformation run time.

Field	Description
None	To skip validation, select this option. This option is selected by default.
Use validation rule defined by the project's output structure	To validate the transform using the rule defined by the Transform component output structure, select this option. This option is not selected by default. For EDI HIPAA and EDI X12, selecting this option activates the Ignore NTE Segment check box.
Ignore NTE Segment	For EDI HIPAA and EDI X12, select this check box if you do not want to validate mapping rules specific to NTE. This check box is not selected by default. This option allows NTE segment translations to appear in the XML document in the same sequence as that of the output document. However, if this option is not checked, specific rules can be applied in the dictionary to position NTE segment translations accordingly. NTE is a floating segment that can occur in any place within a document. It is applicable to versions prior to EDI X12 version 4010. This check box is not available for EDIFACT.

Field	Description
Restore Defaults	Applies the default values to the Validation tab by selecting the None option.

Fixed Width Output Properties

The tables in this topic describe the tabs and fields for the Fixed Width output format, also called FWF. Fixed Width files are flat files that contain fixed width data fields that constitute records. Records are commonly separated by carriage return (new line) characters.

Structure tab: Allows you to select an XML file that represents the data dictionary for the Fixed Width output data. The data dictionary describes the structure of the output message.

Field	Description

Structure	Identifies the structure component used for the output dictionary of the Transform component.
	
	To locate the file, type the file name, or click the *Browse* ... button.
	
	The following is a sample Fixed Width structure file, which is created when you click the *New* button.
	
	`<FIXED_WIDTH>`
	`  <RecordHeader>`
	`    <Column Name="DESCRIPTION" StartOffset="1"`
	`     Length="10" />`
	`  </RecordHeader>`
	`  <RecordDetails>`
	`     <Loop ID="Loop" Req="M" Min="1" Max="1">`
	`       <RecordLayout Type="10" StartOffset="1"`
	`        Length="2">`
	`           <Column Name="Column1" StartOffset="3"`
	`            Length="5" />`
	`           <Column Name="Column2" StartOffset="8"`
	`            Length="5" />`
	`       </RecordLayout>`
	`     </Loop>`
	`  </RecordDetails>`
	`</FIXED_WIDTH>`

Restore Defaults	Clears the Structure field.

Data tab: Allows you to configure a Fixed Width output data file.

Field	Description
Record Delimiter	Specifies delimiter options for an output data file. The record delimiter is a character that defines the boundary between records. The default value is a carriage return.
Padding	Specifies the character used for padding empty spaces. If you leave this field blank, iWay Transformer uses a blank space for padding.
Align	Specifies the alignment of the output. Left-justified text (the default value) is aligned to the left side of each field. Right-justified text is aligned to the right side. For example, consider the following dictionary file:

```
<RecordDetails>
  <RecordLayout Type="NA" StartOffset="1"
   Length="2">
    <Column Name="MSG_ID                "
     StartOffset="1" Length="8"/>
    <Column Name="ITEM_UNIQUE_ID        "
     StartOffset="9" Length="20"/>
    <Column Name="ITEM_CASE_NUM         "
     StartOffset="30" Length="18"/>
    <Column Name="ORIGIN                "
     StartOffset="49" Length="8"/>
     .
     .
     .
```

Left-aligned data is displayed as:

```
ABC 55    777    XXX
```

Right-aligned data is displayed as:

```
ABC     55     777 XXX
```

Field	Description
Encode	When selected, encodes your data in EBCDIC format. By default, this check box is not selected.
Trim Columns	Removes trailing spaces from columns in the output data file. By default, this check box is selected.
Support Record ID	Returns the support record ID flag. By default, this check box is selected.

Field	Description
Restore Defaults	Applies the default values to the Data tab, by clearing the Record Delimiter field and Padding field, selecting *Left* from the Align field drop-down list, and selecting the *Trim Columns* check box.

HTML Output Properties

The following table describes the tabs and fields for the HTML output format.

Data tab: Allows you to configure HTML output data.

Field	Description
Mode	Specifies the format of the HTML output. Choose either *Form* or *Table* (the default value) format.
Stylesheet Type (Optional)	Specifies the format of the style sheet that you are using. The default value is text/css.
Stylesheet File (Optional)	Identifies the style sheet that is applied to your output. To locate the file, type the file name, or click the *Browse* or *Import* button. This file must match the selected file type.

Header (Optional)	Specifies header comments that are included in the output structure as necessary. You can specify one or more lines, separated by a pipe symbol (|). For example,
	
	`This information|is inserted|into the output`
	
	adds the following header to the output:
	
	`<!--This information-->`
	`<!--is inserted-->`
	`<!--into the output-->`

Field	Description

Footer (Optional)	Specifies footer comments that are included in the output structure as necessary. You can specify one or more lines, separated by a pipe symbol (|). For example,
	`This information|is inserted|into the output`
	adds the following footer to the output:
	`<!--This information-->`
	`<!--is inserted-->`
	`<!--into the output-->`

Restore Defaults	Applies the default values to the Data tab, by selecting *Form* from the Mode drop-down list, selecting *text/css* from the Stylesheet Type drop-down list, and clearing the Stylesheet File, Header, and Footer fields.

IDOC Output Properties

The following table describes the tabs and fields for the SAP Intermediate Documents (IDOC) output format. IDOCs represent a standard data structure for electronic data interchange between application programs written for SAP systems or between an SAP application and an external program.

You can access IDOCs using the iWay Application Adapter for mySAP ERP in iWay Explorer. For more information, see the *iWay Application Adapter for mySAP ERP User's Guide*.

iWay Transformer does not support the collected IDOC files, due to the structural mapping. You must split the IDOC batch documents into the individual IDOC files in order to correctly process them.

Structure tab: Allows you to select a file that represents the data dictionary of the IDOC output data.

Field	Description
Restore Defaults	Clears the Structure field.

SWIFT Output Properties

The following table describes the tabs and fields for the Society for Worldwide Interbank Financial Telecommunication (SWIFT) output format. SWIFT is a messaging and transaction processing format used by worldwide financial organizations.

Structure tab: Allows you to select a data dictionary that represents the SWIFT output data.

Field	Description
Structure	Identifies the structure component used for the output dictionary of the Transform component. To locate the file, type the file name, or click the *Browse* ... button.
Import Dictionary from System Library	Allows you to add a data dictionary from a system library.

XML, iWay XML Embedded Request, and iWay XML Request Output Properties

The tables in this topic describe the tabs and fields for the XML, iWay XML Embedded Request, and iWay XML Request output formats.

Structure tab: Allows you to select an XML, DTD, or XSD (schema) file that represents the data dictionary for the XML output data.

Field	Description
Structure	Identifies the structure component used for the output dictionary of the Transform component. It is typically represented as an XSD (schema), DTD, or XML file that describes the outgoing message. To locate the file, type the file name, or click the *Browse* ... button. The following is an example of an iWay XML Embedded Request:

```
<?xml version="1.0" encoding="UTF-8" ?>
 <Insert_DW_Customer>
  <iway>
   <request agent="XDJdbcAgent">
    <connection>
     <dsn>DW_SQL</dsn>
     <user>iwaytest</user>
     <password>ENCR(4485729374774629470
               110485)</password>
     <sql>
     <query>INSERT INTO StageDimCustomer (Customer,
        CustomerName, Address, ARGroup, GroupName,
Owner,
        SalesRep, Jurisdiction, ShippingWarehouse,
        DeliveryDay, CustomerCity, CustomerState,
        CustomerZip, Beantown, Bullseye,
WarehouseTransfer)
        VALUES (555, 'DOLLAR STORE NO. 150       ',
'33 BAY
        VIEW ROAD        ', 'H6A', SAMPLE,
Inc.        ', '
        ', 31, 0, 9, '4', '', 'NY', '13454', '', '',
'YES')
     </query>
    </sql>
   </connection>
  </request>
 </iway>
</Insert_DW_Customer>
```

You can load a schema from the server registry or from a server configuration by selecting *Import Schema*, *From Server*. Another option is to load the schema from one of the available iWay Web services by selecting *Import Schema*, *From Webservice*. You can also load the schema from an Ebix archive by selecting *Import Schema*, *From Ebix*.

Field	Description
Contains Namespace	Select this check box if the output contains namespaces. This check box is not selected by default. You can also specify a namespace for individual items. For more information, see *Working With Namespaces* on page 265.

Data tab: Allows you to configure an XML output data file.

Field	Description

Node Indent	Indicates the number of spaces used to indent the node. The default value is 4.
	
	For example, if the node indent is 0, all elements are aligned at the margin, as follows:
	
	`<a>`
	`<b>`
	`</b>`
	`<c>`
	`</c>`
	`</a>`
	
	If the node indent is 4, all elements except the root are indented 4 spaces, as follows:
	
	`<a>`
	`    <b>`
	`    </b>`
	`    <c>`
	`    </c>`
	`</a>`

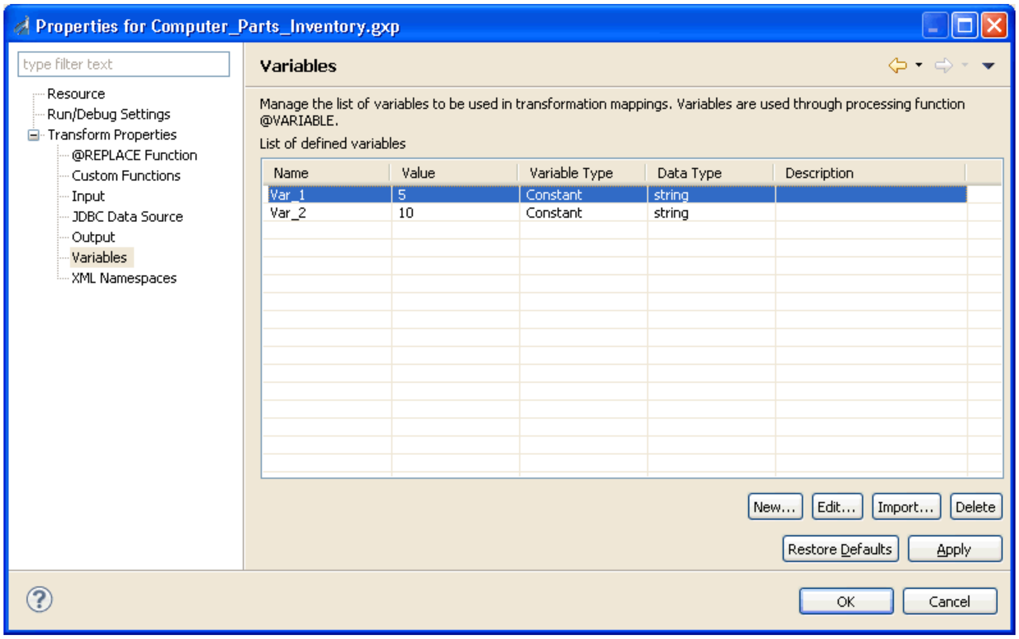
Optimization	Enables you to retain or omit empty elements and attributes. To retain empty elements and attributes, select *Do Not Optimize*. This is the default value. To omit empty elements and attributes, select *Remove All Empty Nodes*. To omit empty elements, including empty nested groups, select *Remove Empty Group Nodes*.

Field	Description
Include XML Declaration in the output data	Automatically inserts an XML declaration into the output data structure. This option is selected by default.
Character Encoding	Select the character encoding from the drop-down list. By default, UTF-8 is selected.
Stylesheet Type (Optional)	Select either *text/xsl* or *text/css*. By default, text/xsl is selected. This option is an HTML property that is rarely used in plain XML format.
Stylesheet File (Optional)	To locate the style sheet file that you want to use for the output, type the full path, click *Browse*, or click *Import* to navigate to the file. This option is an HTML property that is rarely used in plain XML format.
Header (Optional)	Specifies the header information that you want in the output. This option is an HTML property that is rarely used in plain XML format.
Footer (Optional)	Specifies the footer information that you want in the output. This option is an HTML property that is rarely used in plain XML format.

Variables

The Variables category enables you to manage the list of variables that you can use for the output node values. Using variables improves the readability and usability of the output. You can use a defined variable in the output node mapping value through the @GETCONSTANT or @VARIABLE function.

The following image shows the Variables pane.

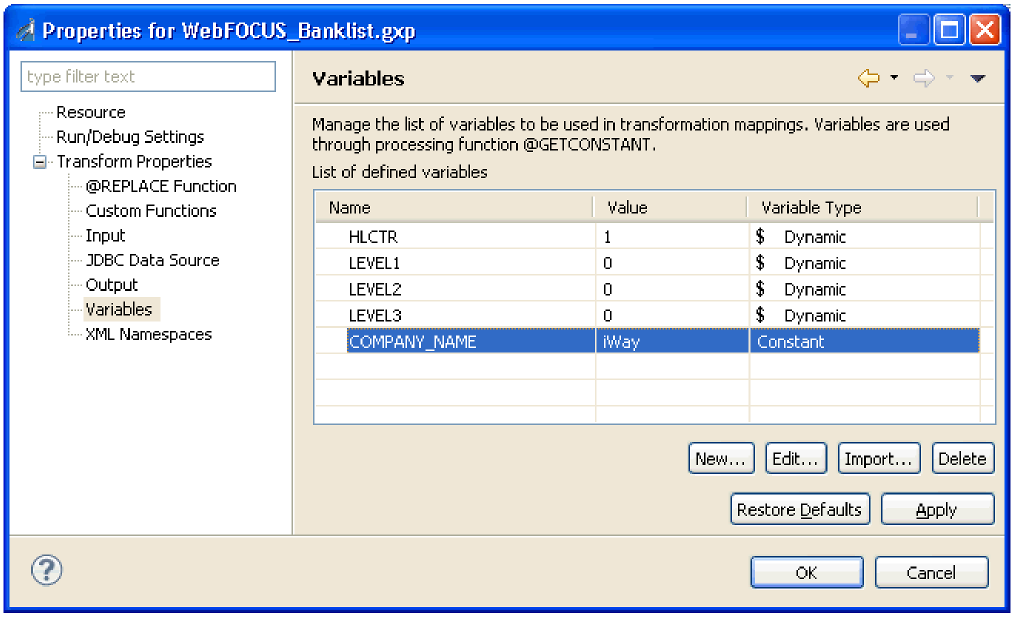


Manage the list of variables to be used in transformation mappings. Variables are used through processing function @VARIABLE.

List of defined variables

Name	Value	Variable Type	Data Type	Description
Var_1	5	Constant	string	
Var_2	10	Constant	string	

The Variables category is not upward compatible. Transform components with defined variables do not function properly in earlier versions of iWay Transformer. Projects must use global constants in earlier versions of iWay Transformer, as shown in the following image.



Manage the list of variables to be used in transformation mappings. Variables are used through processing function @GETCONSTANT.

List of defined variables

Name	Value	Variable Type
HLCTR	1	$ Dynamic
LEVEL1	0	$ Dynamic
LEVEL2	0	$ Dynamic
LEVEL3	0	$ Dynamic
COMPANY_NAME	iWay	Constant

Variable is a new mapping value type. It represents an intermediate value used globally during the life of a transformation. You can define two types of variables, *dynamic* and *constant*:

- The value of a dynamic variable can change during the course of a transformation (run time). As a result, a dynamic variable is an ideal candidate for supporting elements of the transformation, such as loop counters or global data storage. The use of variables is much less resource-expensive, and more efficient, than that of Special Registers (SREGs). Consequently, we recommend that you use variables for any values that you do not need to make available after the execution of the transformation.

- Constant variables (formerly known as global constants) represent a subset of the scope of a variable. As a result, you can now define a constant value as a variable of type constant. This feature is backward compatible with the implementation of global constants from previous releases of iWay Transformer.

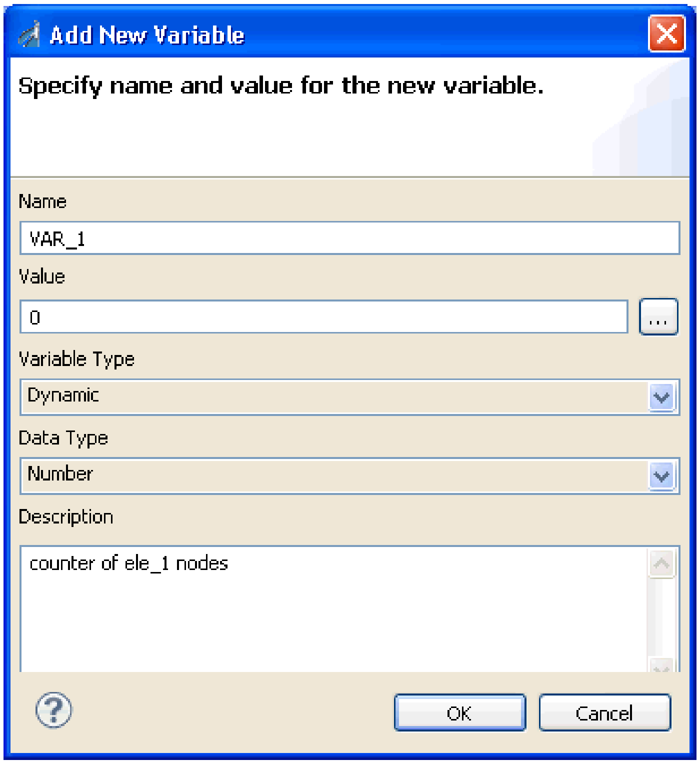
You must define the following parameters for a new variable:

- **Name.** The name of the variable.

- **Value.** The value of the variable.

- **Variable Type.** The type of the variable, Dynamic or Constant.
- **Data Type.** The data type for the value of the variable. For example, the data type for a counter is Number.
- **Description (optional).** A brief description of the variable and its functionality according to your design.

The following image shows the Add New Variable dialog box.


Add New Variable

Specify name and value for the new variable.

Name: VAR_1

Value: 0

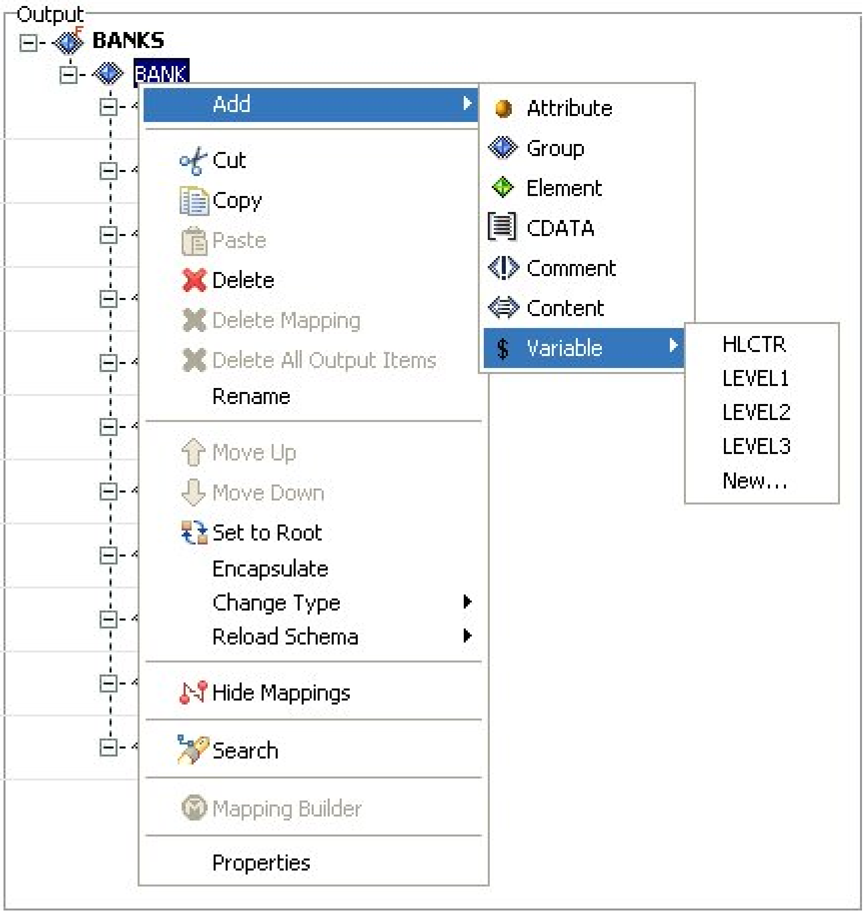
Variable Type: Dynamic

Data Type: Number

Description: counter of ele_1 nodes

After you define and initialize the variable in the Variables category, you can use it within the workspace of a project. The full list of available variables is added to the type of nodes to add, under the Variable option on the context menu of the mapping workspace. That way, it can be added within your output mappings and assigned values.

The following image shows the Variable option on the context menu.



You can also import variables from other Transform components. Variables in the iWay Transformer workspace are identified by the dollar sign ($) icon.

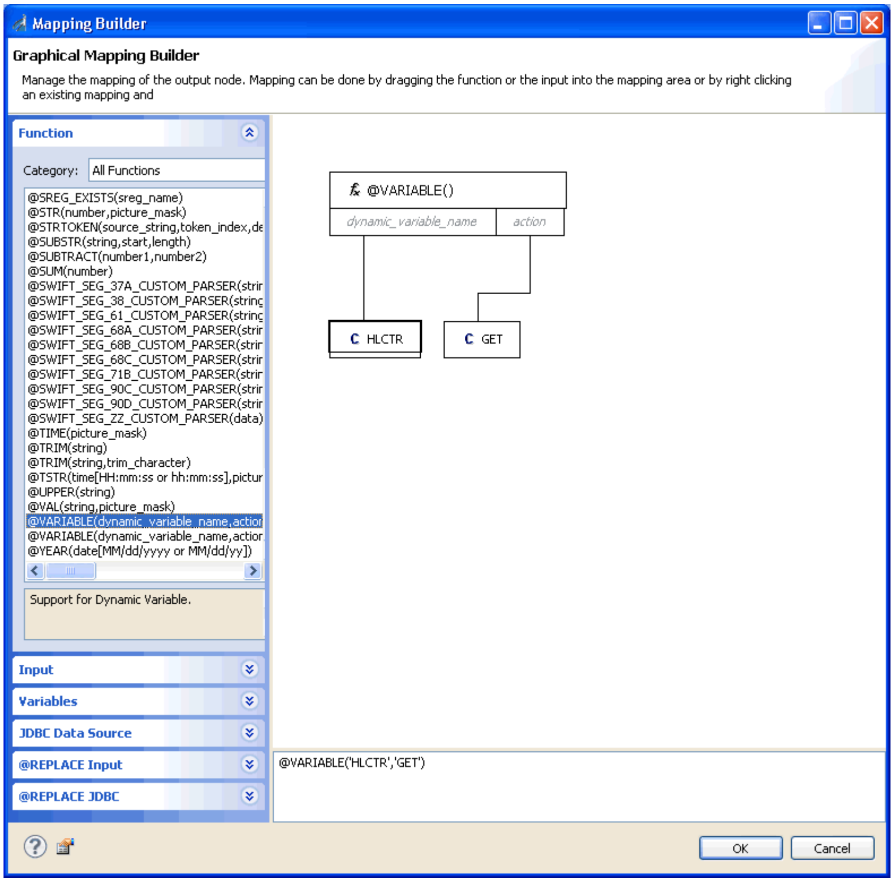
You can use two predefined processing functions in iWay Transformer to manipulate variables. They support two different signatures, depending on the action (ADD, GET, or SET) that needs to be performed over the referenced variable:

- @VARIABLE(*name, action, value*). Returns the value of the specified variable.
- @VARIABLE(*name, action*). Returns the action of the specified variable.

The following table describes the arguments for each function.

Name	Description	Value	Default	Required
name	Variable name	N/A	VAR_1	Yes
action	Action that is performed by this function	❏ 0: SET-void assigns the variable value to a mapped value specified in the input property. ❏ 1: GET-return String returns the variable value. ❏ 2: ADD-void appends to a value depending on the variable data type: ❏ dt string: append ❏ dt number: add	2 (ADD)	Yes
value	Value that is used in the action	N/A	1 for number. Null for string.	No

Standard drag-and-drop functionality applies to the variable functions, as shown in the following image of the Mapping Builder.



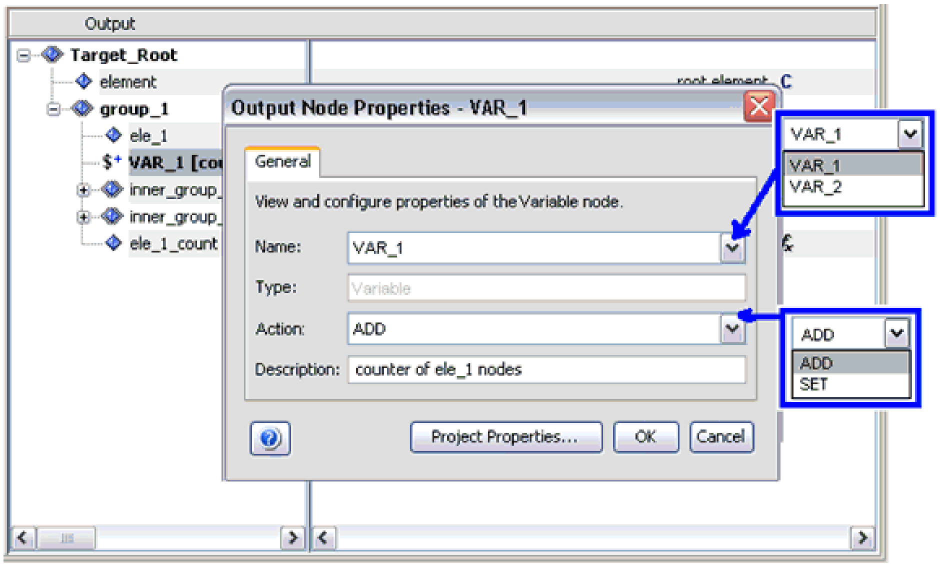
Manage the mapping of the output node. Mapping can be done by dragging the function or the input into the mapping area or by right clicking an existing mapping and

```
@SREG_EXISTS(sreg_name)
@STR(number,picture_mask)
@STRTOKEN(source_string,token_index,de
@SUBSTR(string,start,length)
@SUBTRACT(number1,number2)
@SUM(number)
@SWIFT_SEG_37A_CUSTOM_PARSER(strir
@SWIFT_SEG_38_CUSTOM_PARSER(string
@SWIFT_SEG_61_CUSTOM_PARSER(string
@SWIFT_SEG_68A_CUSTOM_PARSER(strir
@SWIFT_SEG_68B_CUSTOM_PARSER(strir
@SWIFT_SEG_68C_CUSTOM_PARSER(strir
@SWIFT_SEG_71B_CUSTOM_PARSER(strir
@SWIFT_SEG_90C_CUSTOM_PARSER(strir
@SWIFT_SEG_90D_CUSTOM_PARSER(strir
@SWIFT_SEG_ZZ_CUSTOM_PARSER(data)
@TIME(picture_mask)
@TRIM(string)
@TRIM(string,trim_character)
@TSTR(time[HH:mm:ss or hh:mm:ss],pictur
@UPPER(string)
@VAL(string,picture_mask)
@VARIABLE(dynamic_variable_name,actior
@VARIABLE(dynamic_variable_name,actior
@YEAR(date[MM/dd/yyyy or MM/dd/yy])
```

Support for Dynamic Variable.

```
@VARIABLE('HLCTR','GET')
```

You can change the variable name and action using the Output Node Properties dialog box, as shown in the next image.



Output Node Properties - VAR_1

General

View and configure properties of the Variable node.

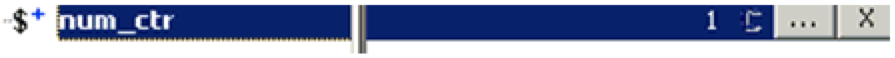
Name: VAR_1

Type: Variable

Action: ADD

Description: counter of ele_1 nodes

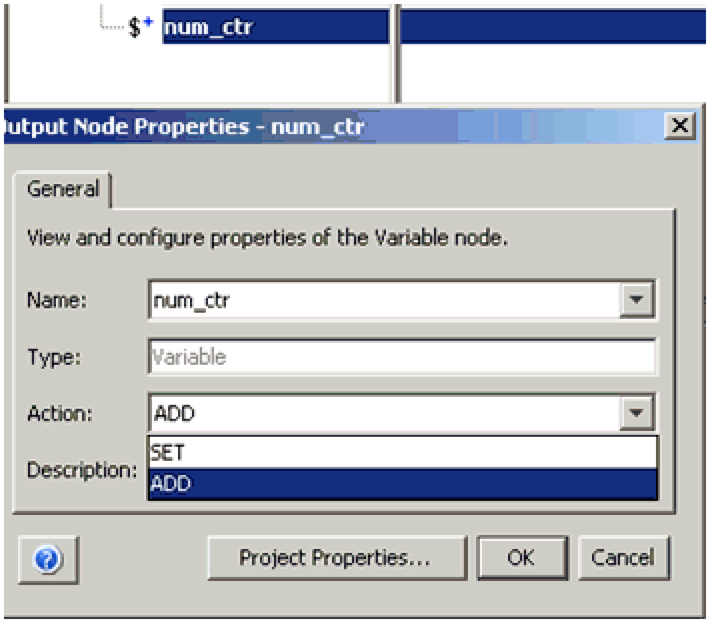
By default, if the variable is numeric, an increment of 1 is added to the output structure, as shown in this image.


num_ctr 1

Incrementing the variable by -1 performs a subtraction.

You can change the output node properties for a variable from ADD to SET to assign a value to the variable node, instead of an increment.

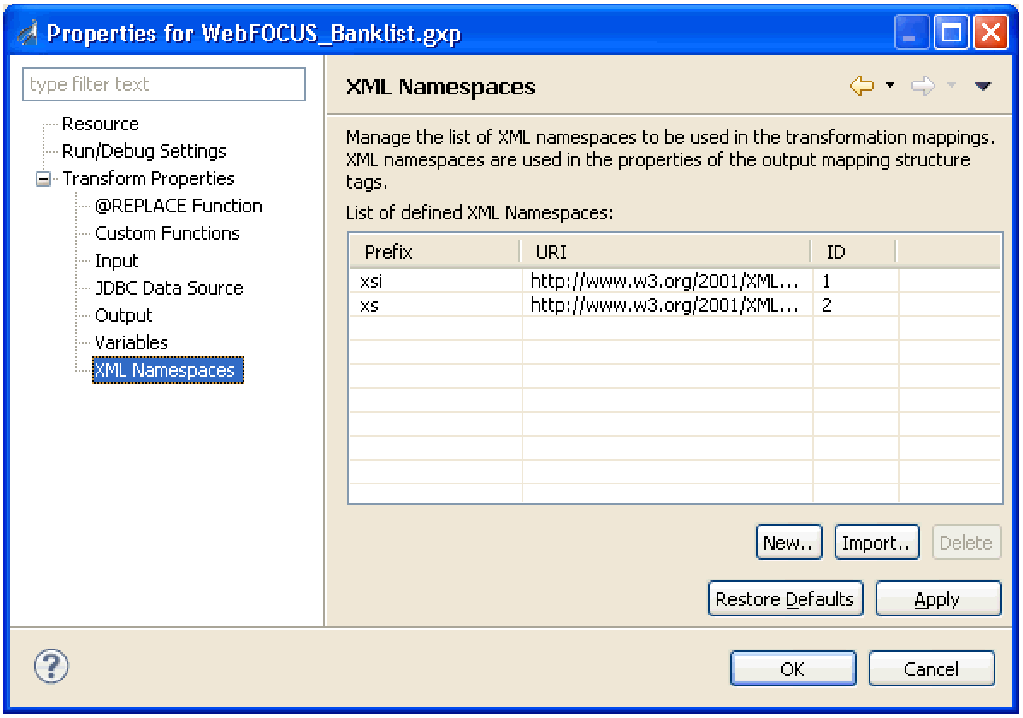
The following image shows the Action drop-down list, from which you can change ADD to SET.



You can also insert the @VARIABLE function inside a formula that is developed using the Mapping Builder (for example, as a result of the @IF condition), just like any regular predefined iWay Transformer function.

XML Namespaces

The XML Namespaces category enables you to load XML namespaces from other projects, or create your own. The following image shows the XML Namespaces pane.



XML Namespaces

Manage the list of XML namespaces to be used in the transformation mappings. XML namespaces are used in the properties of the output mapping structure tags.

List of defined XML Namespaces:

Prefix	URI	ID
xsi	http://www.w3.org/2001/XML...	1
xs	http://www.w3.org/2001/XML...	2

For more information on defining XML namespaces, see *Working With Namespaces* on page 265.

Testing a Transform Component

The following procedure describes how to test run a Transform component.

Procedure: How to Test a Transform Component

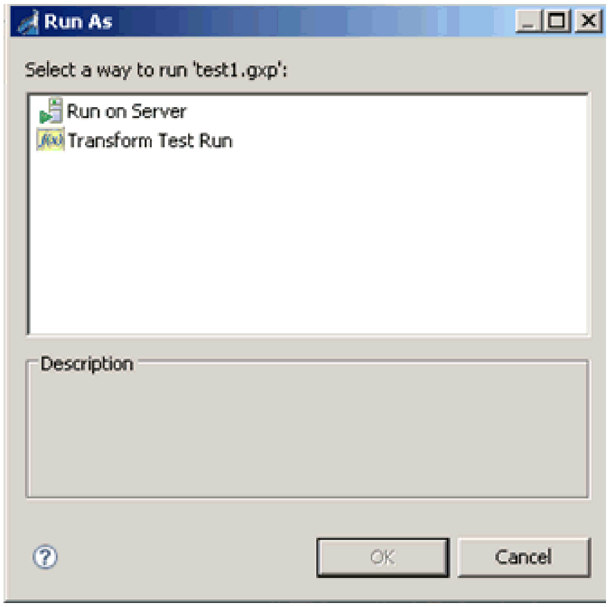
1. Select the Transform component that you wish to test run, and right-click it to open the context menu.
2. Select *Run As* and then *Transform*.
   The test run configuration wizard opens.
3. Select a server to test run against using the Server URL combo box. Alternatively, type the URL of a server.
4. Click *Run* to start the test run process.

Tip: You can re-run the last test by clicking the icon.

Working With a Transform Component

iWay Transformer offers design-time and run-time service modes for managing Transform components.

The following image shows the Run As dialog box, from which you select the way to run a Transform component.


Run As

Select a way to run 'test1.gxp':

Run on Server
Transform Test Run

Description

Design-Time Mode (Transform Test Run)

Design-time mode supports the development of and publication of Transform components on the run-time server. You can create your own components in design-time mode, or use existing components. You can transform messages between XML message formats and non-XML message formats.

Electronic messages often use standard formats such as EDI X12 or SWIFT. iWay makes such formats available as part of its standard adapter suite. Frequently, however, modifications to these standards are required for specific business situations. iWay Transformer provides the tools to customize and extend the offered standard formats to cover such specific situations.

Once you have defined a Transform component, the design-time tool enables you to test it and validate it against test profiles. Once you have validated the Transform component, you can publish it to one or more run-time servers.

Ensure that the iWay home folder is set in Preferences.

Run-Time Mode (Run on Server)

Run-time mode uses a published Transform component as an integral part of iWay Service Manager. You can also use this component at various points within a message processing life cycle, as required.

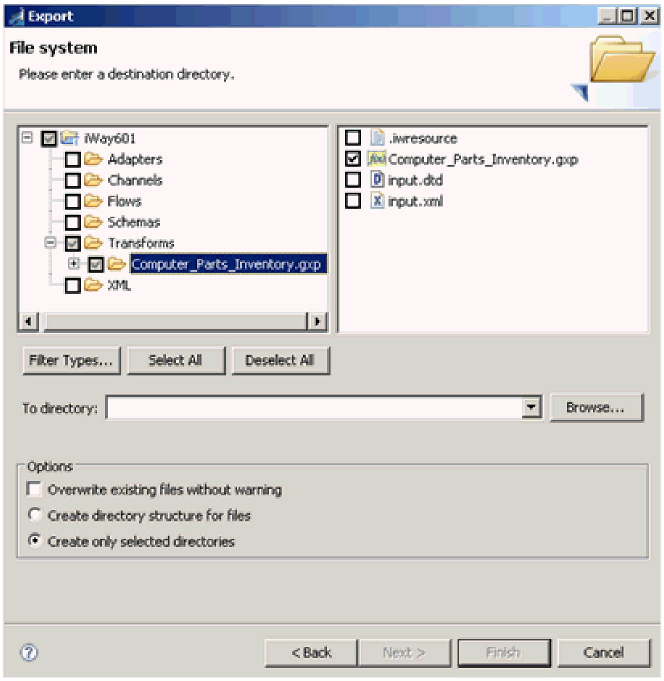
Opening a Transform Component

You can import a Transform component from your local drive, or an iWay server. Once a Transform component is part of your Integration Project, open it by double-clicking its name, or selecting *Open* from the context menu.

Saving a Transform Component

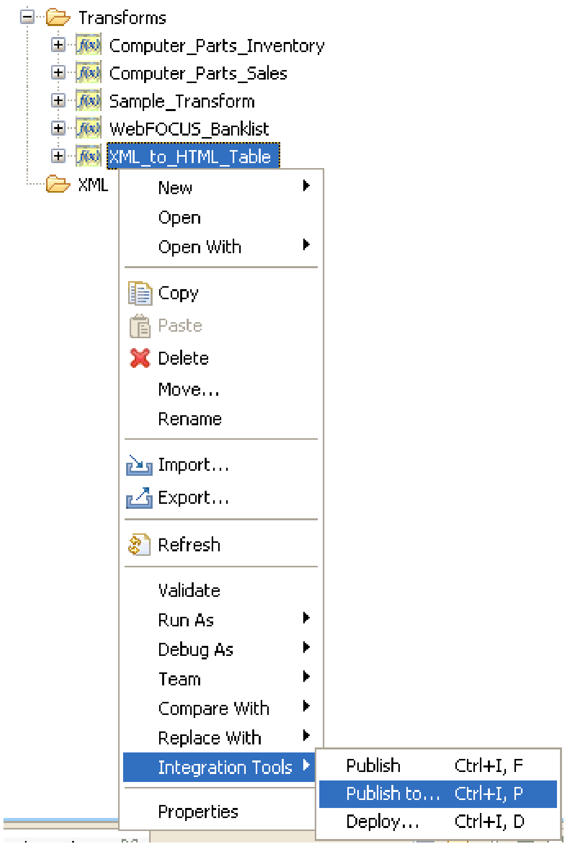
You can export a Transform component from your Integration Project to your file system or other destination.

The following image shows the Export dialog box.

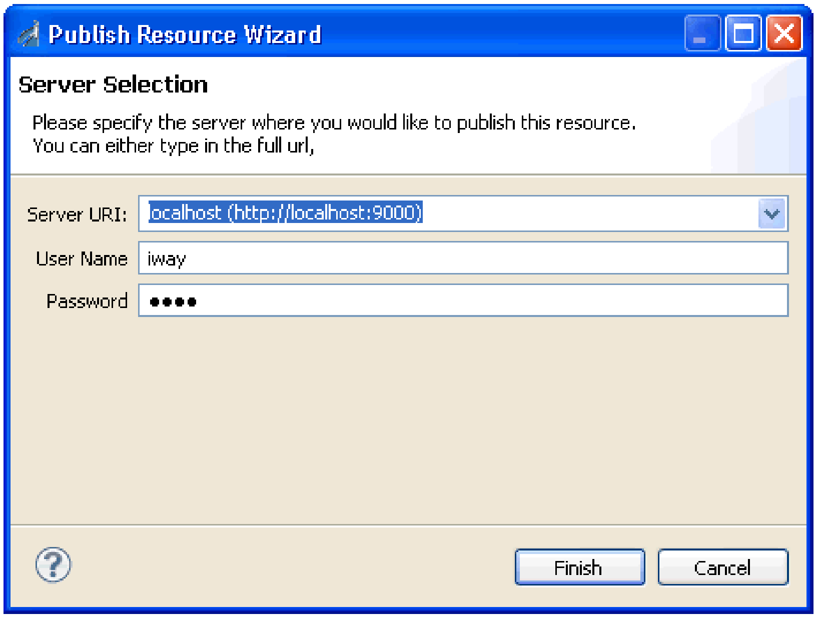

Export
File system
Please enter a destination directory.
iWay601
Adapters
Channels
Flows
Schemas
Transforms
Computer_Parts_Inventory.gxp
XML
.iwresource
Computer_Parts_Inventory.gxp
input.dtd
input.xml
Filter Types...
Select All
Deselect All
To directory:
Browse...
Options
Overwrite existing files without warning
Create directory structure for files
Create only selected directories
< Back
Next >
Finish
Cancel

Publishing a Transform Component

To publish a Transform component, select the *Publish to* option from the context menu, as shown in the following image.



Specify the server on which to publish the Transform component, as shown in the next image.



Publish Resource Wizard

Server Selection

Please specify the server where you would like to publish this resource.
You can either type in the full url,

Server URI: localhost (http://localhost:9000)

User Name: iway

Password: ●●●●

Working With the Mapping Builder

When designing a Transform component, you must define the mapping rules and relationships between input and output documents, according to your requirements.

In iWay Transformer, the Mapping Builder is the user interface that enables you to graphically map your transformations. The following topics describe the mapping types and relationships that are supported for input and output documents and the mapping values that you can assign.

Mappings Tab

The Mappings tab is the workspace and default display for designing a Transform component.

The following information is available on the Mappings tab:

- ❑ Input and output document structure tree. You can entirely or partially expand or collapse the structure nodes.

  Note: As part of the document structure tree, the individual fields and records of your input and output are referred to as nodes throughout this manual.

- ❑ The data associations between input and output nodes, represented by lines between nodes.

If you do not want to display the mapping lines, click the *Hide Mappings* button to disable this option.

- Details of a particular structure node.

  You can right-click the node and select *Properties*.

The Mappings tab includes the following panes, which systematically display mapping-related information:

- **Input.** This pane is displayed on the left. It contains the visual representation of the document tree of the structure of the input data (incoming document). The input structure is read only on the Mappings tab.
- **Output.** This pane is displayed to the right of the Input pane. It is your workspace for designing the structural layout of the output document, building its visual representation and specifying its properties.

When working with structures, you can perform the following tasks:

- Load and configure input or output structure nodes. Input configuration is limited.

  For more information on loading the input and output documents, see *Configuring a Transform Component* on page 164.
- Drag and drop nodes from the Input pane to the Output pane in order to copy parts of the input document tree to the output.
- Add or delete a structure node.

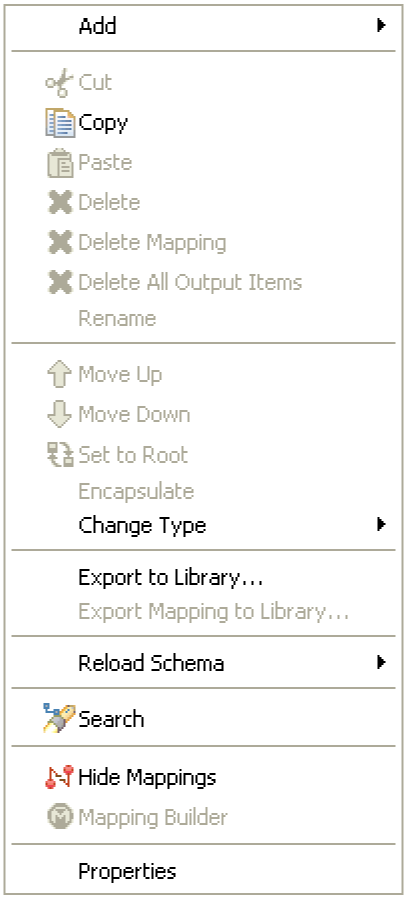
The following menus let you easily perform various mapping-related tasks:

- The Input Node menu provides the available options for manipulating the nodes in the input structure, which is displayed in the Input pane. For more information, see *Input Node Workspace Menu* on page 226.
- The Output Node menu provides the available options for manipulating the nodes in the output structure, which is displayed in the Output pane. For more information, see *Output Node Workspace Menu* on page 228.
- The Mapping Values menu provides the available options for assigning values to the nodes in the output structure, which is displayed in the Mapping pane. For more information, see *Mapping Values* on page 256.

Input Node Workspace Menu

Right-click an input node in the workspace to display the menu options that are available for the nodes in the Input pane. You can expand or collapse various nodes within the structure tree, copy a node, and show or hide mappings between input and output nodes.

The following image shows the options available on the Input Node menu.



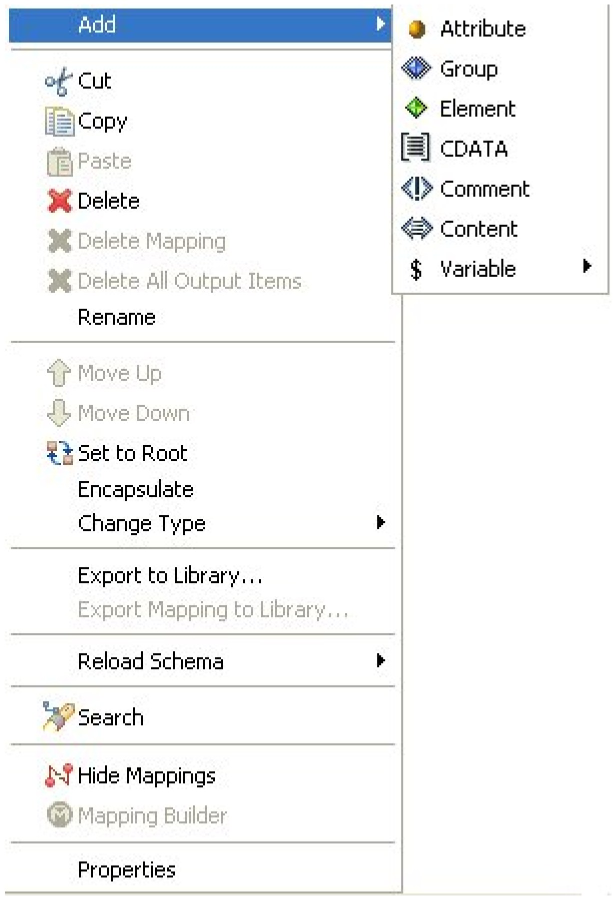
The following table describes the Input Node menu options.

Option	Description
Copy	Copies the selected input node and enables you to paste it in the output.

Option	Description
Reload Schema	Reloads the schema for the Input or Output structure pane.
Search	Opens the Search dialog box, enabling you to search for text or to search for and map input and output nodes.
Hide Mappings	Toggles between showing and hiding the mapping lines between the input and output structures.
Properties	Displays the properties of the selected input node.

Output Node Workspace Menu

Right-click an output node in the workspace to display the menu options that are available for the nodes in the Output pane. The following image shows the options available on the Output Node menu.



The options that are available on the Output Node menu depend on which node is currently selected in the Output pane.

You can disable certain options, depending on the type and position of the output node in the structure (for example, group, element, or attribute) or the data format (for example, XML or EDI X12).

The Add and Change Type option lists are dynamic. They also depend on the type of the output node and the output data format.

The following table describes the options available on the Output Node menu.

Option	Description
Add	Enables you to alter the existing output structure by inserting a new structure node of a specific type. The following types of structure nodes are supported: ❏ Attribute (for XML format only) ❏ Group ❏ Element ❏ CDATA (for XML format only) ❏ Comment (for XML format only) ❏ Content (for XML format only) ❏ Variable ❏ iWay XML Request (Available only when the output type is iWay XML Request.) ❏ iWay Embedded XML Request (Available only when the output type is iWay Embedded XML Request.) Refer to the Add submenu of a particular node for the list of enabled node types that you are permitted to add. The values of the available node types are enabled or disabled according to the mapping rules for the output data format. For example, if the XML data format is used, the only available types of nodes that you can add to an element type of node is an attribute node or a CDATA node.
Cut	Cuts the selected output node, making it available for the paste output operation.
Copy	Copies the selected node to a location that you determine.
Paste	Pastes the cut or copied node.
Delete	Removes a node from the Output structure. A confirmation dialog box is displayed for this option.

Option	Description
Delete Mapping	Deletes the selected mapping.
Delete All Output Items	Deletes all output items.
Rename	Renames the selected output structure node. This option is also available if you double-click the name of the node.
Move Up	Moves the selected node up the output structure tree, under the same parent node.
Move Down	Moves the selected node down the output structure tree, under the same parent node.
Set to Root	Enables you to represent the root of the document structure tree. This option is not available for record-based data formats, such as CSV. This option is available only for the nodes of type Group, because this is the only type of node that is permitted to perform the root functionality.
Encapsulate	Creates an invisible parent group wrapped around the output node that you choose to encapsulate. The encapsulating feature is useful when you are resolving complex looping or mapping issues. For an example, explore the WebFOCUS_Banklist sample transform project that is packaged with iWay Transformer.
Change Type	Changes the type of the node to one of the following: ❑ Attribute ❑ Group ❑ Element ❑ CDATA ❑ Comment ❑ Content For more information, see *Output Structure* on page 235.

Option	**Description**
Export to Library	Exports the selected component to the library.
Export Mapping to Library	Exports the selected mapping to the library.
Reload Schema	Reloads the schema for the Input or Output structure pane.
Search	Opens the Search dialog box, enabling you to search for text or to search for and map input and output nodes.
Hide Mappings	Toggles between showing and hiding the mapping lines between the input and output nodes. The mappings signify the relationships between the input and output nodes, where the particular input value is used to construct the value of the output node.
Mapping Builder	Opens the Mapping Builder window, which enables you to modify the mapping value by changing the input node that it maps to. Alternatively, you can add a function, constant, or expression to the existing mapping. For more information on how to use the Mapping Builder, see *Working With Functions* on page 269.
Properties	Displays the Properties tab, which shows the layout and structural information pertaining to the selected output node. For more information, see *Group Properties* on page 241.

Input Structure

The input structure contains the input, which represents the groups, elements, and attributes encoded in the input document. Each node in this tree has exactly one parent, and can have more than one child.

The following table describes the various components provided in iWay Transformer to represent input nodes. Definitions of basic terms commonly used for nodes within an input document tree are also provided.

Input Document

The incoming document or message to which transformations apply. The following formats are supported:

- CSV. For more information, see *CSV Input Properties* on page 183.
- EDI HIPAA. For more information, see *EDI Input Properties* on page 184.
- EDI X12. For more information, see *EDI Input Properties* on page 184.
- EDIFACT. For more information, see *EDI Input Properties* on page 184.
- Fixed Width. For more information, see *Fixed Width Input Properties* on page 187.
- IDOC. For more information, see *IDOC Input Properties* on page 189.
- SWIFT. For more information, see *SWIFT Input Properties* on page 190.
- XML. For more information, see *XML and iWay XML Response Input Properties* on page 191.
- iWay XML Response. For more information, see *XML and iWay XML Response Input Properties* on page 191.
- CDF. For more information, see *CDF Input Properties* on page 182. **Note:** CDF format is deprecated. iWay Software recommends that you use Fixed Width format instead.

The input document is displayed on the input pane as a visual representation of the input document tree. It maintains a logical order of component nodes, such as groups or elements, in relation to the underlying document, wrapping the nodes that are contained within other nodes. For more information, see the Input Document Tree Example that follows. Each input node contains a name that identifies the type of the node, optionally a number of attributes, or content.

Group

Block of data that has a group element as its root. It can contain other groups or elements nested within, as children. Multiple groups can also exist on the same level. A group was formerly called a parent.

iWay Transformer does not support mixed content group nodes containing data values. However, the group node for XML and HTML data formats can have any number of attribute nodes inside, if applicable.

Element

Usually belongs to the group as a leaf, which cannot have any other nested groups or elements. It typically stores a data value. In XML and HTML data formats, it can contain attribute nodes or CDATA.

Attribute

Value associated with a group or element, consisting of a name, and an associated textual value. Attributes are used for XML and HTML data formats only. Each group or element can have zero to many attributes.

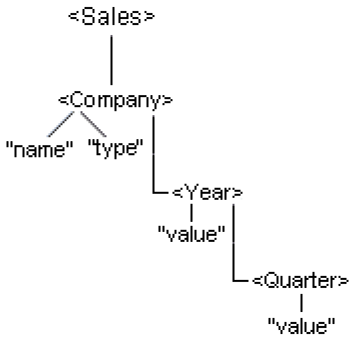
Example of Input Document Tree

Sample document tree of groups, elements, and attributes encoded in the input message. Each node in this tree has exactly one parent, and may have more than one child.

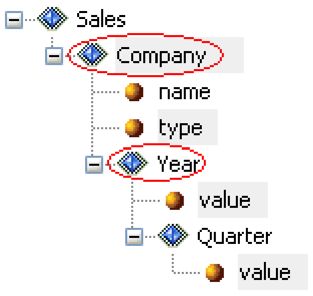
The following is a section of an XML input document:

```
<?xml version="1.0" encoding="UTF-8"?>
<Sales>
                <Company name="Video and Sound Card Express" type="Computer Parts">
                                <Year value="2001">
                                                <Quarter value="1st"/>
                                </Year>
                </Company>
</Sales>
```

The following is a tree structure representation of the preceding XML:



In iWay Transformer, the resulting input document tree is as follows:



You can collapse or expand the document tree from the given node to its deepest descendant. iWay Transformer supports a set of different relationships that link nodes together.

The common types of document tree relationships are described in the following table.

Parent The input can have more than one parent (root group). Parent is a group in relation to the elements or groups contained within it. In the preceding example, the group, Company, is a parent of two elements, Year and Quarter.
Child Node A is called the child of node B, if and only if B is the parent of A. In the preceding example, the element Year is a child of the Company group.
Descendant Node A is called a descendant of node B, if either (1) A is a child of B, or (2) A is the child of node C, which is a descendant of B.
Ancestor Node A is called an ancestor of node B, if and only if B is a descendant of A.
Sibling Node A is called a sibling of node B, if and only if B and A share the same parent. Node A is a preceding sibling if it comes before B in the input document tree. Node B is a following sibling if it comes after A in the input document tree.

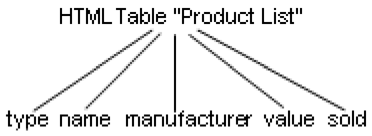
Output Structure

The output structure contains the output document tree, which represents the completed structure of groups, elements, and other supported node types, such as attributes or CDATA, encoded in the output document.

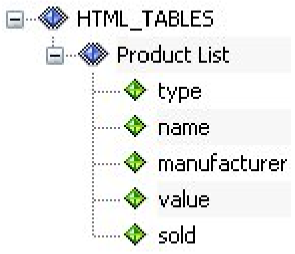
Each node in this tree has exactly one parent group. It can have one or more sibling nodes, unless it is the root node in the tree, which has to be unique according to the mapping rules in iWay Transformer. You can adjust the properties of the given node, such as visibility or namespace, to affect its appearance in the actual output document.

This topic examines the output document tree in the XML to HTML table example, which is included as a sample Transform component.

The output structure is represented as follows:



In iWay Transformer, the output document tree is visualized as follows in the Output pane of the Mappings tab:



Mapping Types

The output document tree or hierarchy provides a graphical representation of how the output data will be organized. The output hierarchy defines output nodes line-by-line and uses a document tree structure to produce the desired output according to your requirements.

Nodes of type group and element typically serve as the primary building blocks for the output document tree, with the group type node at the root of the document tree, from which the rest of the structure is derived. Groups can have children and can be nested, which means that group nodes can be children and parents in relation to other group nodes.

Element nodes cannot have children, except in XML format, in which elements can contain XML-specific type nodes, such as attributes or CDATA items. Element nodes are always derived from the group node parent, which means that they are acting as leaves of the group branches.

Note: You can change the visibility of groups and elements or use predefined looping mechanisms to present your output data in the format specific to your requirements.

The following node types are supported in the Mapping Builder:

- Group
- Element
- Attribute (for XML format only)

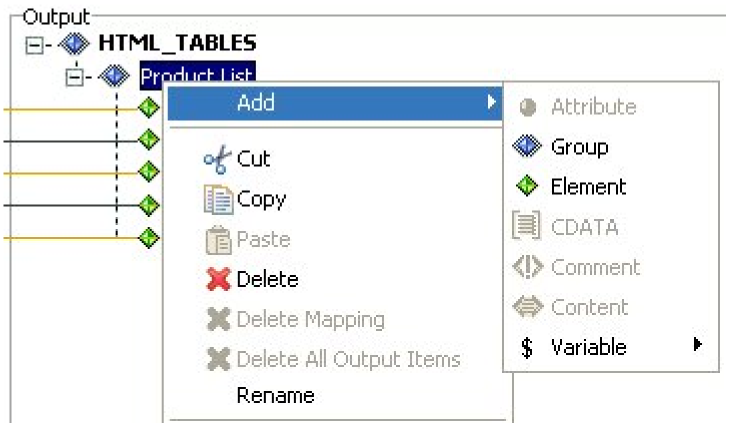
- Comment (for XML format only)
- Content (for XML format only)
- CDATA (for XML format only)
- Variables (for XML format only)

Adding New Output Nodes

One of the efficient methods for adding new output nodes is structure mapping. For more information about structure mapping, see *Group* on page 238. Another method of adding new output nodes is by building your document output tree from scratch, which is achieved by right-clicking an existing output node and selecting *Add* followed by the type of node you wish to add.

The supported types of the output nodes you may insert will depend on the type of the parent node to which you want to add your data nodes and can be driven by the context menu. For example, if you right-click an XML element node in your output structure, you may discover that the only possible type of nodes which can be added to that element node is an attribute node. By following these restrictions, the integrity of the output structure is maintained.

You can review the available options for the output node type by right-clicking the parent node and selecting the *Add* submenu.

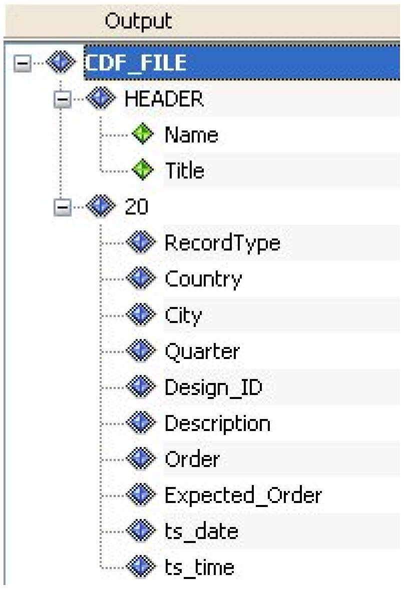


Group

A group is a block of data that has a group node as its root. It can contain other group or element nodes nested within, and other node types if supported (for example, attribute or CDATA for XML data format). Multiple groups can also exist on the same level. The nodes contained within a certain group are also called children of this particular group, sometimes referred to as a parent group in relation to the nodes nested within it.

Every output structure, regardless of the output data format, has at least one group node at its root. A group marks the start of an output block by specifying the start of an output loop. The first node in your output structure, referred to as a Root, is created automatically by iIT Transformer when you create your initial output structure. You can then build your structure from the root group node by adding the nodes of supported type one by one, or by copying the blocks of data from existing input or output structures. The output is produced by looping through the entire output structure (starting from the root group), one or more times (iterations), depending on your settings, while reading the mapped values from the input file.

Consider the CDF output structure displayed in the following image.



The first output node, CDF_FILE, is the first group node and is also called the Root. The presence of the Root node ensures that iIT Transformer loops over the entire output structure if needed. The next node in the output tree is a HEADER node, which is also a group type of node. Appearance of the HEADER node marks an output block of data, which includes the child element nodes Name and Title. The node, 20, is also a group node, which contains an output block of data to include the next ten child element nodes (RecordType to ts_time). Group nodes are distinguished by the Group icon, which has a blue diamond shape with a double border. Element nodes are distinguished by the Element icon, which has a blue diamond shape.

The following list contains the visual variations of the group node when a specific property is configured for it.

- **Invisible**

  A group node that has its visible property set to false. This means that even though the group node is present in the output structure during design time, the invisible node's value or the whole block of its children nodes will not be displayed in the run-time output produced by the project. For more information on hiding a group, see *Filter Tab* on page 251.

- **Context**

  A group that has a context applied. The context property is the explicitly defined reference of the output group node to the particular input group node. It helps to control looping, and to resolve challenges and opportunities in the area of data structure searching. It is especially useful for nested multi-level repeated structures. It operates on the logical structure of the message (similar to XPath). It is available for use by any data format supported by iIT Transformer.

  To set the context property, right-click the group node **b** in the Output pane and select *Properties*. The Properties tab opens and displays the General tab by default.

  In the Context field, specify the path value for the looping node on the input side. Click the ellipses ... button, which opens the Input Fields pane. Select an input node to be mapped from the hierarchy tree and click *OK*.

  For more information on setting a context for a group, see *General Tab* on page 250.

- **Filter**

A group that has a filter applied. The filter property is useful when your incoming document data may contain loop iterations that should not appear in your output, according to your requirements. The filter property is designed to help you remove unnecessary blocks of data. For more information on specifying a filter for a group, see *Filter Tab* on page 251.

Depending on the group properties (loop, context, and filter) that are specified in the Properties tab for the selected output node, the group icon changes in the Output pane to reflect these settings. The following table lists all of these variations for your reference.

Group Setting	**Visible Icon**	**Invisible Icon**
Loop is set to true	◆T	◇T
Loop is set to false	◆F	◇F
Loop is set to aggregate	◆A	◇A
Context is applied and loop is true	◆T	◇T
Context is applied and loop is false	◆F	◇F
Context is applied and loop is aggregate	◆A	◇A
Filter is applied and loop is true	◆T	◇T
Filter is applied and loop is false	◆F	◇F
Filter is applied and loop is aggregate	◆A	◇A
Context and filter are applied and loop is true	◆T	◇T
Context and filter are applied and loop is false	◆F	◇F
Context and filter are applied and loop is aggregate	◆A	◇A

Additional XML Output Structures

In iIT Transformer, it is easy to create complex output structures for your output file. The way you arrange the nested components in the Output pane of the Mapping window is exactly how your output will be structured.

You can use parent nodes to create complex output structures with nested parent nodes and output loops for the XML output formats (XML and e-business). EDI (X12, HIPAA, EDIFACT, and SWIFT) output is treated as XML output due to the use of the e-business metadata by the Transformation Engine.

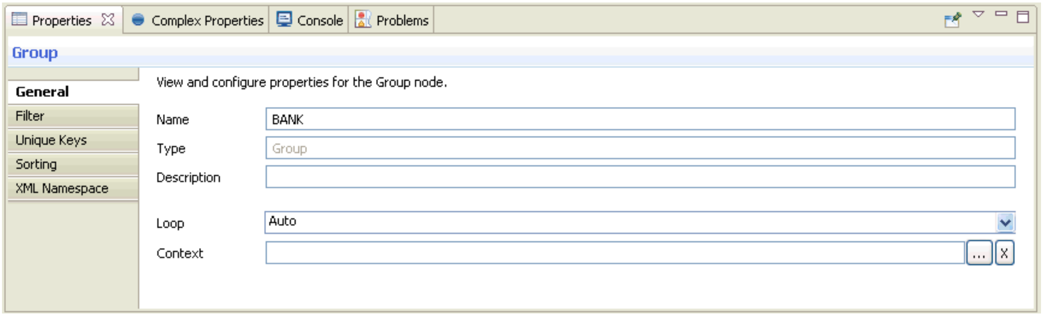
In addition to indicating the possible start of an output loop, a group node can also play an important role in building the following structures depending on the output data format:

- **XML output document -** The first parent node value is defined as the root node of your XML output document. Other group nodes are defined as XML parent node names; they can be used to create nested loops for complex XML structures.

- **EDI (X12, HIPAA, EDIFACT, and SWIFT) output document -** The group node for complex EDI messages can be used to create nested loops.

Group Properties

A set of useful properties is available for group output nodes. To view the properties for a group node, right-click the group node and select *Properties*. The Properties tab opens, as shown in the following image.



View and configure properties for the Group node.

Name: BANK
Type: Group
Description:
Loop: Auto
Context:

The *General* tab is displayed by default. Notice that the Type field contains a default value of *Group*, which indicates the type of mapping node being used. The properties of a group node are displayed in the following tabs:

- General

- Filter

- Unique Keys

- ❑ Sorting
- ❑ XML Namespace

The following topics describe the tabs and options available from the Properties tab for the group node.

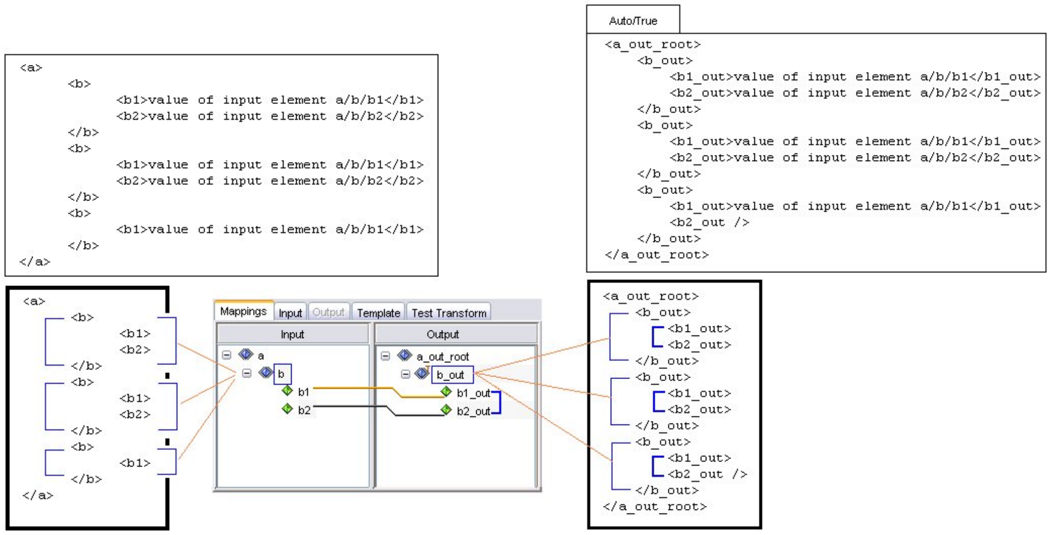
General Tab

The General tab displays information such as the name and type of the node, as well as looping and context settings of the current node.

Looping

In order to determine how your group node will loop, you need to examine the list of available values from the *Looping* drop-down list, and select the applicable option.

- ❑ **Auto** - Enforces the default looping mechanism. This is the default value of the Looping drop-down list.
- ❑ **True** - Indicates that looping is necessary for the data in this group.



```
<a>
      <b>
            <b1>value of input element a/b/b1</b1>
            <b2>value of input element a/b/b2</b2>
      </b>
      <b>
            <b1>value of input element a/b/b1</b1>
            <b2>value of input element a/b/b2</b2>
      </b>
      <b>
            <b1>value of input element a/b/b1</b1>
      </b>
</a>
```

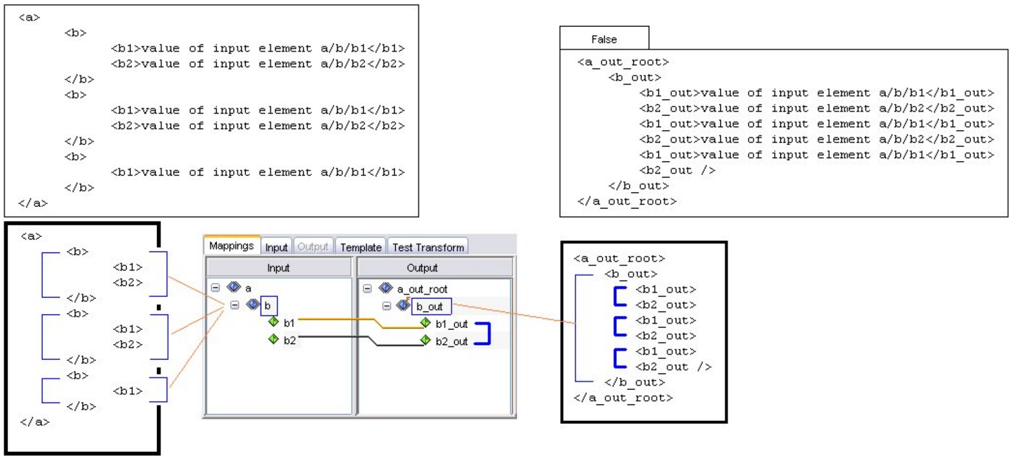
Auto/True

```
<a_out_root>
    <b_out>
        <b1_out>value of input element a/b/b1</b1_out>
        <b2_out>value of input element a/b/b2</b2_out>
    </b_out>
    <b_out>
        <b1_out>value of input element a/b/b1</b1_out>
        <b2_out>value of input element a/b/b2</b2_out>
    </b_out>
    <b_out>
        <b1_out>value of input element a/b/b1</b1_out>
        <b2_out />
    </b_out>
</a_out_root>
```

```
<a>
      <b>
            <b1>
            <b2>
      </b>
      <b>
            <b1>
            <b2>
      </b>
      <b>
            <b1>
      </b>
</a>
```

```
<a_out_root>
    <b_out>
        <b1_out>
        <b2_out>
    </b_out>
    <b_out>
        <b1_out>
        <b2_out>
    </b_out>
    <b_out>
        <b1_out>
        <b2_out />
    </b_out>
</a_out_root>
```

- **False** - Indicates that the group node should not loop.



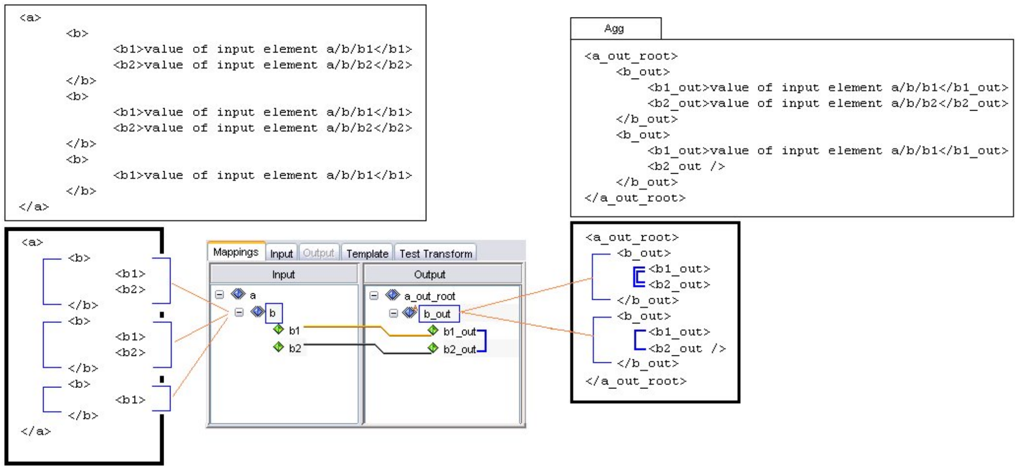
```
<a>
      <b>
            <b1>value of input element a/b/b1</b1>
            <b2>value of input element a/b/b2</b2>
      </b>
      <b>
            <b1>value of input element a/b/b1</b1>
            <b2>value of input element a/b/b2</b2>
      </b>
      <b>
            <b1>value of input element a/b/b1</b1>
      </b>
</a>
```

False

```
<a_out_root>
    <b_out>
        <b1_out>value of input element a/b/b1</b1_out>
        <b2_out>value of input element a/b/b2</b2_out>
        <b1_out>value of input element a/b/b1</b1_out>
        <b2_out>value of input element a/b/b2</b2_out>
        <b1_out>value of input element a/b/b1</b1_out>
        <b2_out />
    </b_out>
</a_out_root>
```

- **Agg** - Indicates that looping should be aggregate.

  If aggregate looping is selected for the particular group, it means that all instances of this group node which have the same attribute values are combined into one unique group node. Any nodes that are children of these elements are also gathered and made subordinate to the new parent node.



```
<a>
      <b>
            <b1>value of input element a/b/b1</b1>
            <b2>value of input element a/b/b2</b2>
      </b>
      <b>
            <b1>value of input element a/b/b1</b1>
            <b2>value of input element a/b/b2</b2>
      </b>
      <b>
            <b1>value of input element a/b/b1</b1>
      </b>
</a>
```

Agg

```
<a_out_root>
    <b_out>
        <b1_out>value of input element a/b/b1</b1_out>
        <b2_out>value of input element a/b/b2</b2_out>
    </b_out>
    <b_out>
        <b1_out>value of input element a/b/b1</b1_out>
        <b2_out />
    </b_out>
</a_out_root>
```

Context

The *Context* field in the *General* tab is an advanced feature designed to help control local looping. Applying a context to a group node in your output structure can solve looping complexity issues. It is especially useful in projects, which require you to make references to input nodes located in different places within the hierarchy of a structure. The *Context* specifies the location of the node where the block of input that is being used for looping begins. Generally, this should be set to the innermost group block from your input structure, which encapsulates all of the input data that must be used within this output group node.

To better understand how the Context feature works, you can examine one of the iWay sample XML projects, located in the following directory:

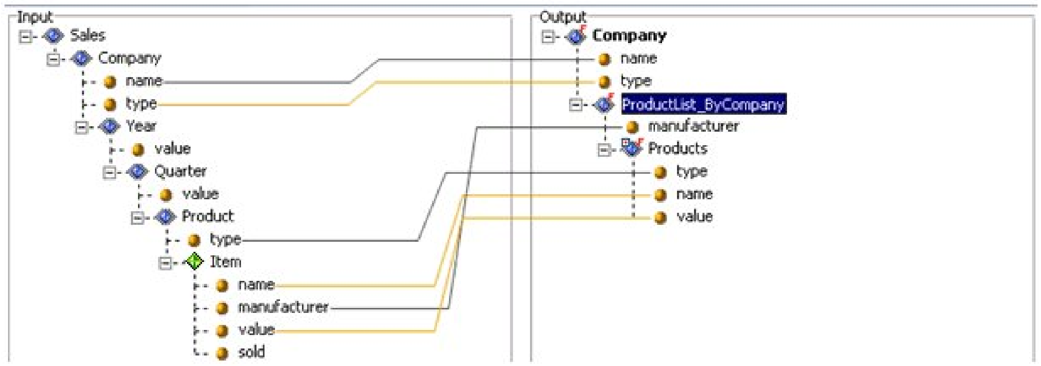
```
<iWaySMHome>\tools\transformer\samples\transform_projects\xml\XML_to_XML
\Context\Context.gxp
```

where:

<iWaySMHome>

Is the directory where iWay Service Manager was installed.

The input to output structure mapping is shown in the following image.



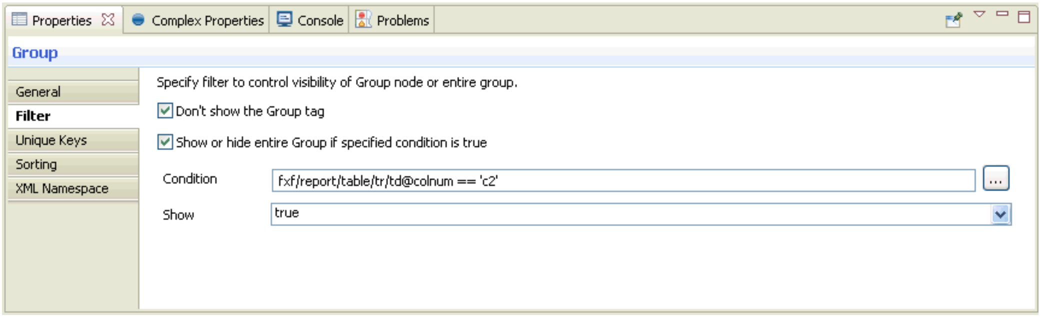
The input is an XML file that lists the details for the sales of a company item. The input information is organized by the year, quarter, product type, and the individual (brand). The output is an XML file containing the sales only for items from the specific manufacturer, for example, Cheapies. The output sales are arranged by year, quarter, and item.

Note that Product List has looping set to aggregate and context is set to Sales/Company/Year. Aggregation looping is required to enforce that only one parent node will exist in the output per each quarter. The reason why the context is set to Year is that the Year is the lowest group node in the input structure hierarchy that contains all of the information used within the Product List node. The input node Year opens the block of data in which the looping will occur.

When a context is applied to a group, the group icon in the Output pane is displayed as follows:

Filter Tab

The following image shows the *Filter* tab.



Specify filter to control visibility of Group node or entire group.

☑ Don't show the Group tag

☑ Show or hide entire Group if specified condition is true

Condition: fxf/report/table/tr/td@colnum == 'c2'

Show: true

The *Show or hide entire Group if specified condition is true* check box enables you to control the appearance of all the nested groups and elements, by defining the condition and resulting action on the group node and its contents. Therefore, each individual output block produced by that output loop will be displayed or hidden in the actual output depending on the state of this option.

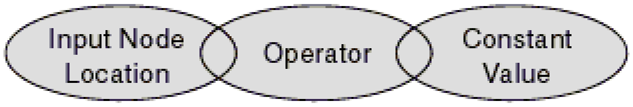
When a filter is applied, the group node icon in the Output pane is displayed as follows:

If this check box is selected and the Hide option is selected in the Action drop-down list, then the particular output block that satisfies the condition entered is omitted from being displayed in the output data.

When a group is hidden, the group icon in the Output pane is displayed as follows:

If the condition you enter is evaluated as true for an output loop and the Show option is selected from the Action drop-down list, then only that output block is displayed in the output. All other iterations of the group loop that do not match the condition you specified are suppressed from being displayed in the output data.

The condition you type must be in the following format:



For example:

Horses/Team/Years/Player/Goals == '30'

where:

Horses/Team/Years/Player/Goals

Is the location of the node in your input document structure.

30

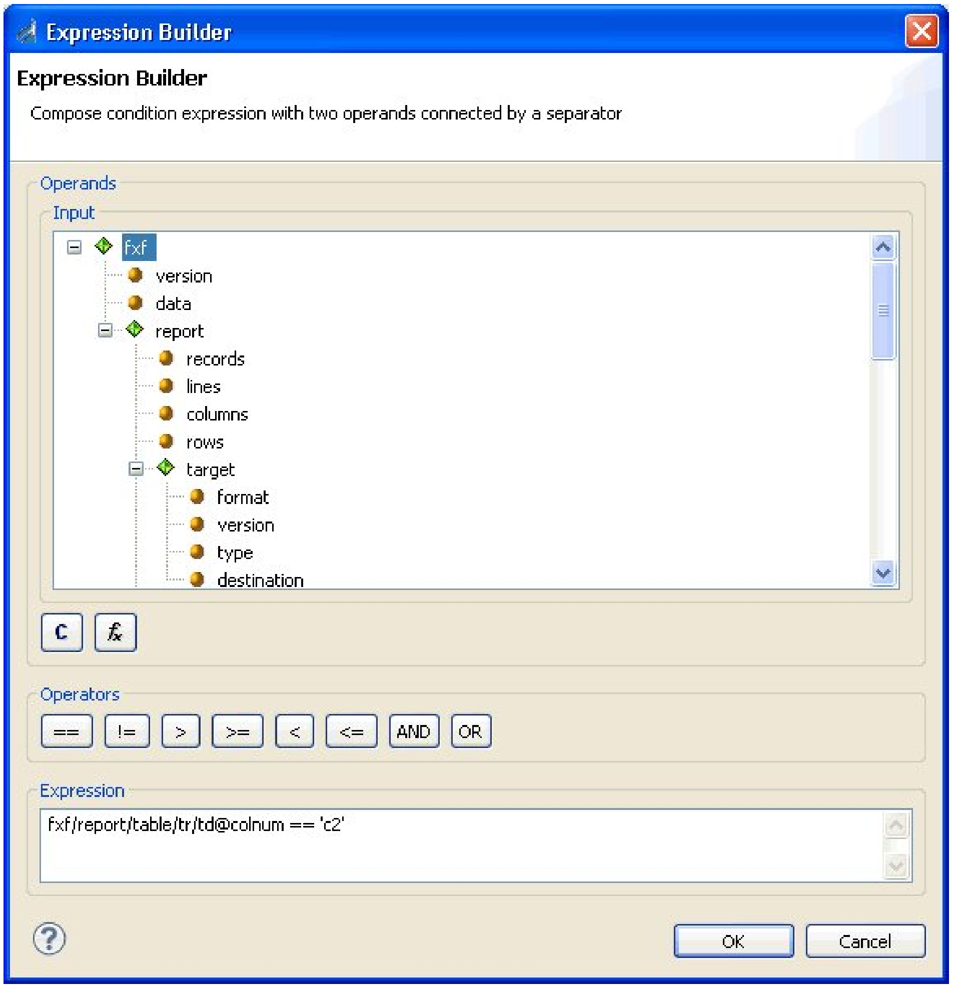
Is the desired alphanumeric value of that node. As the incoming document is processed by the Transformation Engine, if this condition is met in the particular data iteration, the action selected in the Action drop-down list is performed.

If you are using the Expression Builder to construct the condition, click the C icon to specify a constant value.

The following table lists the possible arguments (operators) for the block condition.

= =	equal to
!=	not equal to
>=	greater than or equal to
<=	less than or equal to
>	greater than
<	less than

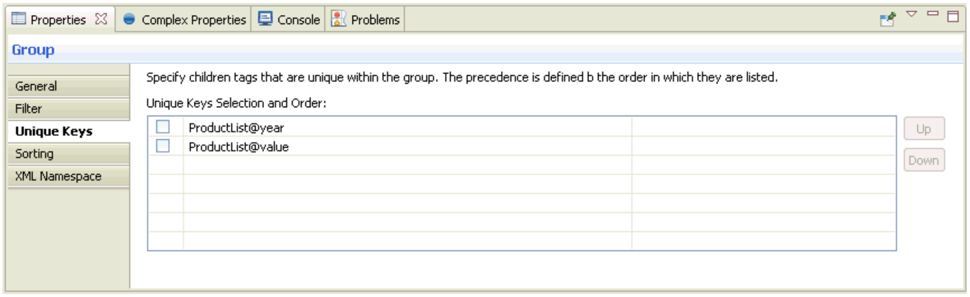
When you click the ellipsis button next to the Condition field, the Expression Builder window is displayed to help you build a condition intuitively.



```
fxf/report/table/tr/td@colnum == 'c2'
```

Unique Keys Tab

The following image shows the *Unique Keys* tab.



Specify children tags that are unique within the group. The precedence is defined b the order in which they are listed.

Unique Keys Selection and Order:

☐	ProductList@year	
☐	ProductList@value	

You can control the appearance of a group's output child nodes by defining nodes as unique in the *Unique Keys* tab. This will remove unnecessary data repetitions from your output group node. The child node you specify as the node will determine which loop repetitions will be suppressed. The output will display only the first unique occurrences of the key node. For example, assume that you generate the following non-unique XML output data:

```
<Product>
    <ProductID>XR281</ProductID>
    <Name>Green Rocket Vehicle</Name>
</Product>
<Product>
    <ProductID>SR71</ProductID>
    <Name>SR-71 Blackbird</Name>
</Product>
<Product>
    <ProductID>XR281</ProductID>
    <Name>Green Rocket Vehicle Again</Name>
</Product>
```

If you choose to make the output group node Product unique by specifying the output child node, ProductID, in the *Unique Keys* parent property, then your final output is the following:

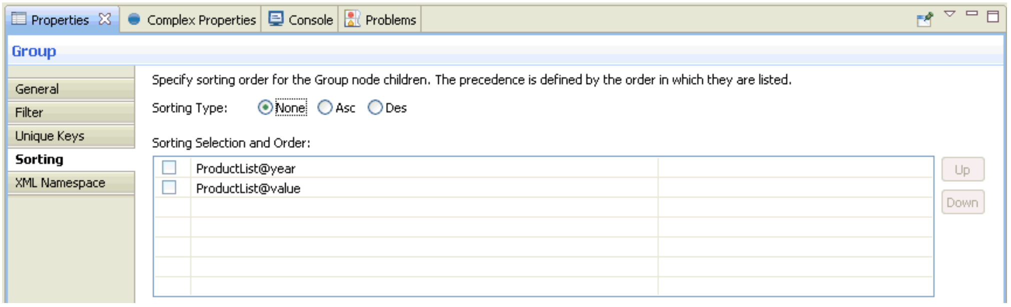
```
<Product>
    <ProductID>SR71</ProductID>
    <Name>SR-71 Blackbird</Name>
</Product>
<Product>
    <ProductID>XR281</ProductID>
    <Name>Green Rocket Vehicle</Name>
</Product>
```

Note: You can specify multiple unique keys if required.

The second instance of the XR281 product was removed because it was not unique to the output item *ProductID*.

Sorting Tab

The following image shows the *Sorting* tab.



Specify sorting order for the Group node children. The precedence is defined by the order in which they are listed.

Sorting Type: [✓ None] Asc Des

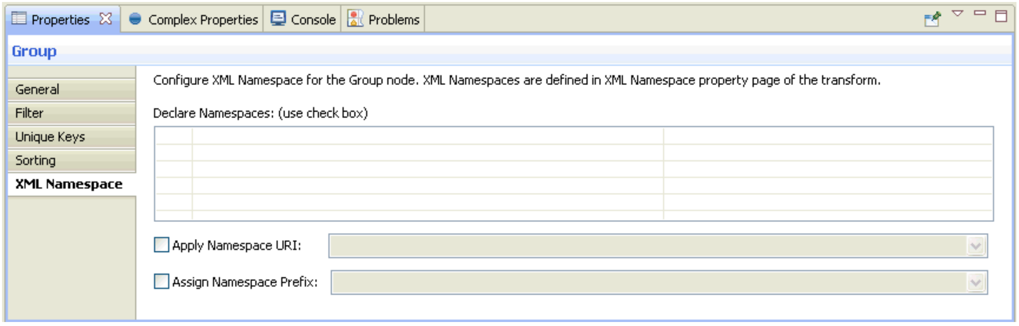
Sorting Selection and Order:

☐	ProductList@year	
☐	ProductList@value	

The *Sorting* tab provides you with the option to enable the sorting of the loop iterations in the blocks of data represented by the particular group node. You can choose a child node within the current group, by which to sort your output. The output can be sorted in ascending or descending alphanumeric order by the node or nodes you choose in the Sorting Order section. The Sorting Type option specifies how the output is sorted. Possible values are *None*, *Asc* (Ascending), or *Des* (Descending). Sorting Type defaults to *None*.

XML Namespace Tab

The following image shows the *XML Namespace* tab. This tab provides options to specify XML namespaces for the individual group node. It is similar to the XML Namespaces category found in the Project Properties dialog box.



Configure XML Namespace for the Group node. XML Namespaces are defined in XML Namespace property page of the transform.

Declare Namespaces: (use check box)

☐ Apply Namespace URI:

☐ Assign Namespace Prefix:

For more information on defining XML Namespaces, see Working With Namespaces on page 265.

Element

An element node is used to represent a basic data item in iIT Transformer. Element nodes are typically found inside the group node. Sometimes it is referred to as a leaf of the output tree because it cannot have any other child nodes, such as the nested groups or elements within it, except for the attribute nodes (for XML data format only).

Element Properties

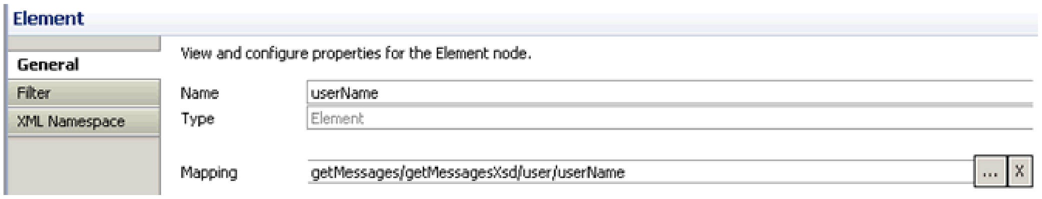
The set of element node properties is defined for an element output node. To view the properties of the element node in the Output pane, you must right-click an element node and then, select *Properties*. The Properties tab opens.

The following topics describe the tabs that are available from the Properties tab for the element node.

General Tab

The following image shows the Properties tab for the element node, which provides the following tabs:

- General
- Filter
- XML Namespace



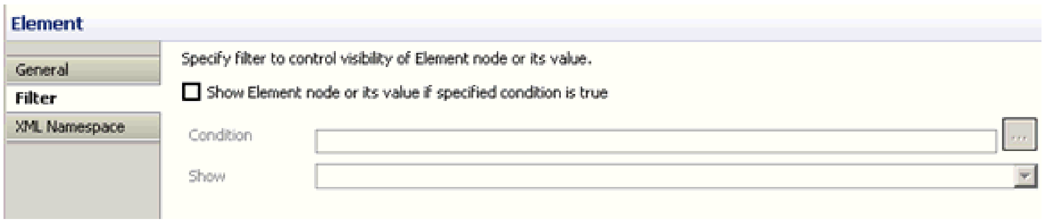
View and configure properties for the Element node.

Name userName
Type Element
Mapping getMessages/getMessagesXsd/user/userName

The Properties tab for the element node defaults to the *General* tab. Notice that the Type field contains a default value of Element, which indicates the type of mapping for which the properties are displayed.

Filter Tab

You can also apply a filter to an element. The following image shows the *Filter* tab for the element node.



Specify filter to control visibility of Element node or its value.

☐ Show Element node or its value if specified condition is true

Condition

Show

The *Show Element node or its specified value if specific condition is true* check box enables you to include all the element nodes or their values in the output, by defining the condition and resulting action on the child node.

The condition you type must be in the following format:

Input Node Location
Operator
Constant Value

For example:

Horses/Team/Years/Player/Goals == '30'

where:

Horses/Team/Years/Player/Goals

Is the location of the node in your input document structure.

30

Is the desired alphanumeric value of that node.

If you are using the Expression Builder to construct the condition, click the C icon to specify a constant value.

The following table lists the possible arguments (operators) for the block condition.

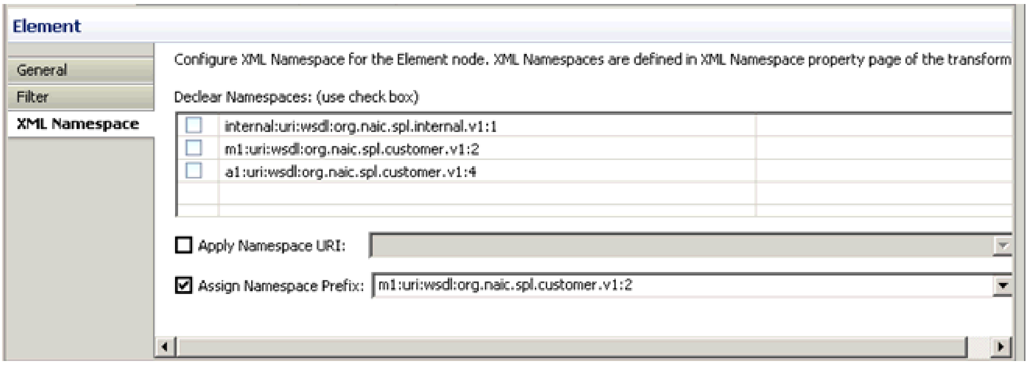
= =	equal to
!=	not equal to
>=	greater than or equal to

<=	less than or equal to
>	greater than
<	less than

When a filter condition is applied to an element node, the element icon in the Output pane is displayed as follows:

XML Namespace Tab

The following image shows the *XML Namespace* tab for the element node.



Configure XML Namespace for the Element node. XML Namespaces are defined in XML Namespace property page of the transform

Declear Namespaces: (use check box)

☐	internal:uri:wsdl:org.naic.spl.internal.v1:1
☐	m1:uri:wsdl:org.naic.spl.customer.v1:2
☐	a1:uri:wsdl:org.naic.spl.customer.v1:4

☐ Apply Namespace URI:

☑ Assign Namespace Prefix: m1:uri:wsdl:org.naic.spl.customer.v1:2

For more information on defining XML Namespaces, see *Working With Namespaces* on page 265.

Attribute

An attribute node contains a value associated with a specific group or element. It consists of a name, and an associated (textual) value. Attribute nodes are supported for XML format data only.

In the following example, *name* and *type* are attributes for the Company element:

```
<Company name="Video and Sound Card Express" type="Computer Parts">
```

Attribute Properties

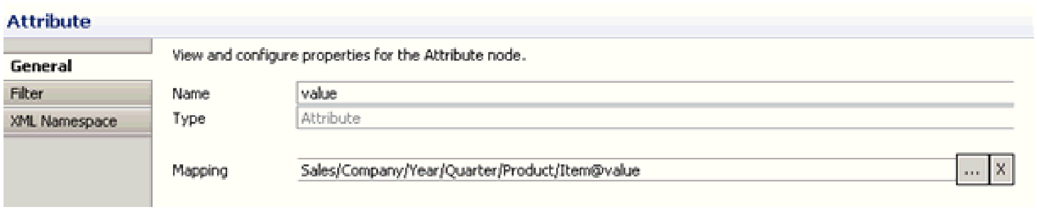
The set of various attribute properties is available for an attribute output node. To view or modify the properties of the attribute in the Output pane of the iIT Transformer workspace, you must right-click an attribute node and then, select *Properties*. The Properties tab opens.

The following topics describe the tabs available from the Properties tab for the attribute node.

General Tab

The following image shows the Properties tab for the attribute node, which provides the following tabs:

- ❑ General
- ❑ Filter
- ❑ XML Namespace



Attribute

General | Filter | XML Namespace

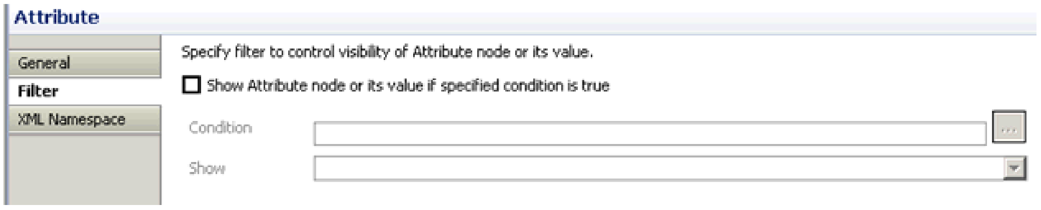
View and configure properties for the Attribute node.

Name: value
Type: Attribute
Mapping: Sales/Company/Year/Quarter/Product/Item@value

The Properties tab for the attribute node defaults to the General tab. Notice that the Type field contains a default value of Attribute, which indicates the type of mapping for which the properties are displayed.

Filter Tab

You can also apply a filter to an attribute. The following image shows the Filter tab for the attribute node.



Attribute

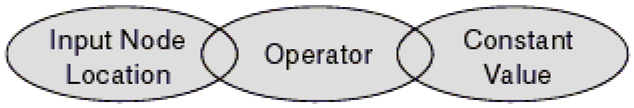
General | Filter | XML Namespace

Specify filter to control visibility of Attribute node or its value.

☐ Show Attribute node or its value if specified condition is true

Condition:
Show:

The *Show Attribute node if specific condition is true* check box enables you to alter the visibility of the attribute nodes or their values in the output, by defining the condition and resulting action on the child node. The condition you type must be in the following format:



For example:

```
Horses/Team/Years/Player/Goals == '30'
```

where:

`Horses/Team/Years/Player/Goals`

Is the location of the node in your input document structure.

`30`

Is the desired alphanumeric value of that node.

If you are using the Expression Builder to construct the condition, click the C icon to specify a constant value.

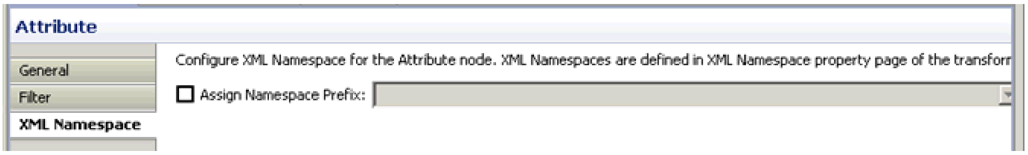
The following table lists the possible arguments (operators) for the block condition.

= =	equal to
!=	not equal to
>=	greater than or equal to
<=	less than or equal to
>	greater than
<	less than

When a filter condition is applied to an attribute node, the attribute icon in the Output pane is displayed as follows:

XML Namespace Tab

The following image shows the XML Namespace tab for the attribute node.



Attribute

General
Filter
XML Namespace

Configure XML Namespace for the Attribute node. XML Namespaces are defined in XML Namespace property page of the transforr

☐ Assign Namespace Prefix:

For more information on defining XML Namespaces, see *Working With Namespaces* on page 265.

Comment

A comment node can be used to embed comments into the output structure. Comment nodes are supported for XML output data only.

In the following example, the phrase *This is an XML Output sample* is a comment.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--This is an XML Output sample-->
```

Content

A content node is associated with a group node in the output document. Some groups have no content, in which case they are called empty. The content of a group node may include text, and it may include a number of sub-elements, in which case the node is called the parent of those subelements. Content nodes are supported for XML output data only.

CDATA

A CDATA node is used to indicate sections of data that you want the XML parser to ignore during validation. CDATA sections can include special characters that will not be parsed. CDATA nodes are supported for XML output data only.

In the following example, the CDATA tag (node) is wrapped around the *compare(a,b)* function, which contains special characters that are not allowed in XML syntax. As a result, these characters will not be parsed.

```
<![CDATA[
function compare(a,b)
{
  if (a > b) then
  {
    return 1
  }
}
}]]>
```

Variable

A Variable node (for XML format only) is used as a mapping value type, which represents an intermediate value used globally during the life of a transformation. Two types of variables can be defined, *dynamic* and *constant*.

Additional Key Terms

Additional key terms used to describe mapping relationships within the output document tree are:

- **Parent**

  A node A is the parent of a node B if node B is contained within the structure of node A, and belongs to the block of data that node A represents. The output has at least one parent group, which is referred to as the root group.

- **Child**

  A node A is called the child of a node B if and only if B is the parent of A.

- **Descendant**

  A node A is called a descendant of a node B, if either (1) A is a child of B, or (2) A is the child of some node C that is a descendant of B

- **Ancestor**

  A node A is called an ancestor of a node B, if and only if B is a descendant of A.

- **Sibling**

  A node A is called a sibling of a node B, if and only if B and A share the same parent. Node A is a preceding sibling if it comes before B in the output document tree. Node B is a following sibling if it comes after A in the output document tree.

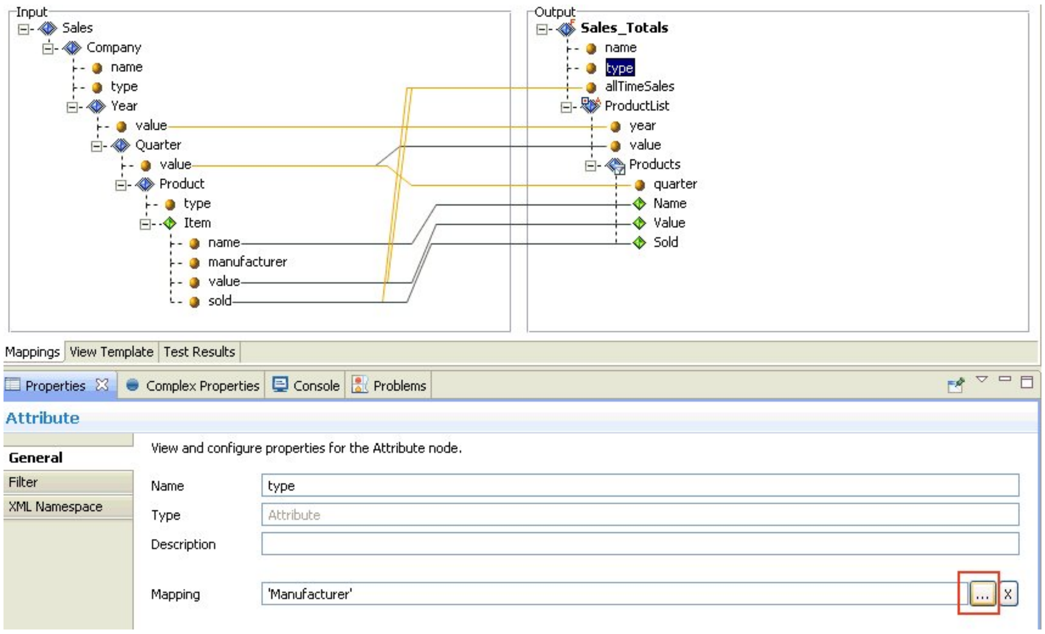
Mapping Values

Creating an output structure is usually not sufficient to perform a transformation. You typically must also specify the mapping values for each output node using the interface of the Mapping Values pane.

The following mapping values are supported.

- **Input Node:** Value of a node from an input structure.
- **Function:** Predefined or custom algorithm of calculation.
- **Constant:** Fixed value.
- **Expression:** Logical statement.

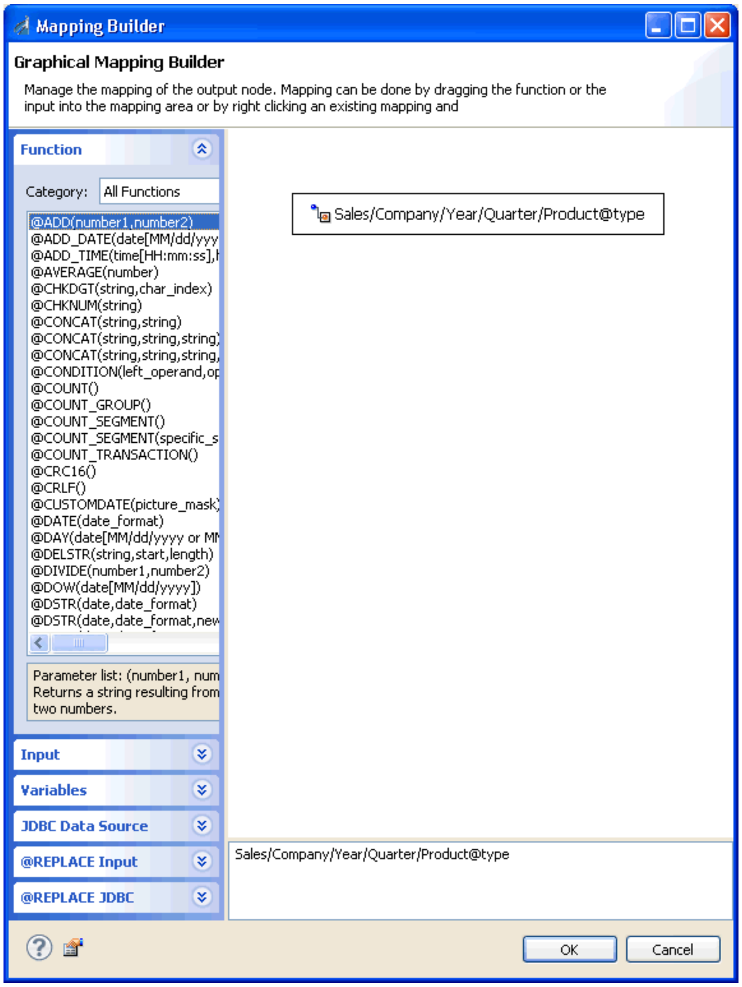
You can specify the value for each output node using the Mapping Builder. To access the Mapping Builder, highlight the output node on the Output pane, find the Mapping field on the properties pane of that node, and click the *Browse* button.



View and configure properties for the Attribute node.

Name: type
Type: Attribute
Description:
Mapping: 'Manufacturer'

The following image shows the Mapping Builder.



Manage the mapping of the output node. Mapping can be done by dragging the function or the input into the mapping area or by right clicking an existing mapping and

```
@ADD(number1,number2)
@ADD_DATE(date[MM/dd/yyy
@ADD_TIME(time[HH:mm:ss],
@AVERAGE(number)
@CHKDGT(string,char_index)
@CHKNUM(string)
@CONCAT(string,string)
@CONCAT(string,string,string)
@CONCAT(string,string,string,
@CONDITION(left_operand,op
@COUNT()
@COUNT_GROUP()
@COUNT_SEGMENT()
@COUNT_SEGMENT(specific_s
@COUNT_TRANSACTION()
@CRC16()
@CRLF()
@CUSTOMDATE(picture_mask)
@DATE(date_format)
@DAY(date[MM/dd/yyyy or MM
@DELSTR(string,start,length)
@DIVIDE(number1,number2)
@DOW(date[MM/dd/yyyy])
@DSTR(date,date_format)
@DSTR(date,date_format,new
```

Parameter list: (number1, num
Returns a string resulting from
two numbers.

Sales/Company/Year/Quarter/Product@type

Sales/Company/Year/Quarter/Product@type

The left pane of the Mapping Builder dialog box provides you with the option of switching between the following areas:

- Functions
- Input
- Variables
- JDBC Data Source

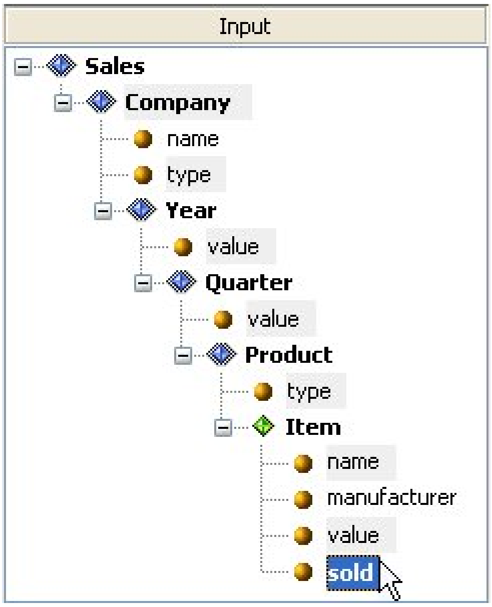
- @REPLACE Input
- @REPLACE JDBC

Click the name of the corresponding pane to expand its content. Click *New* to add new values to the Variables and JDBC Data Source panes.

This feature allows you to drag and drop any available configuration that was specified using the Project Properties dialog box into the workspace area of the Mapping Builder. You can switch between the available panes by clicking on the corresponding button, for example, *Variables*, *JDBC Data Source*, and so on.

Input Node (Context)

Input node represents a value from an input structure. Mapping to an Input node can be done using Input pane. This pane is typically located in the lower-left corner of the Mapping Builder. It allows you to select the location of the input node on which to base the value of the current output node.

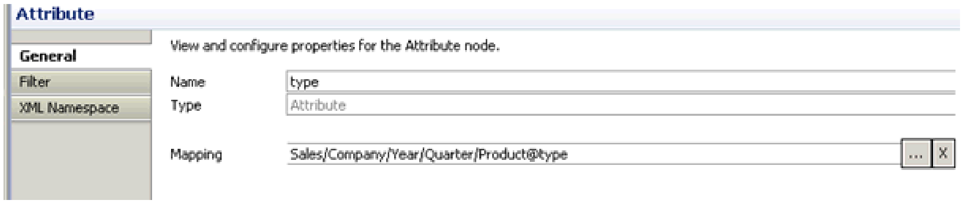


You can drag and drop the input node into the working area of the Mapping Builder. Depending on where in your workspace you drop the node, you can overwrite or update the existing mapping tree.

The Mapping Builder uses the following notation as the context property to define the location of the input node in the mapping formula:

```
Sales/Company/Year/Quarter/Product/Item@sold
```

This is the same notation that is used in the Mapping Values pane, as shown in the following image.

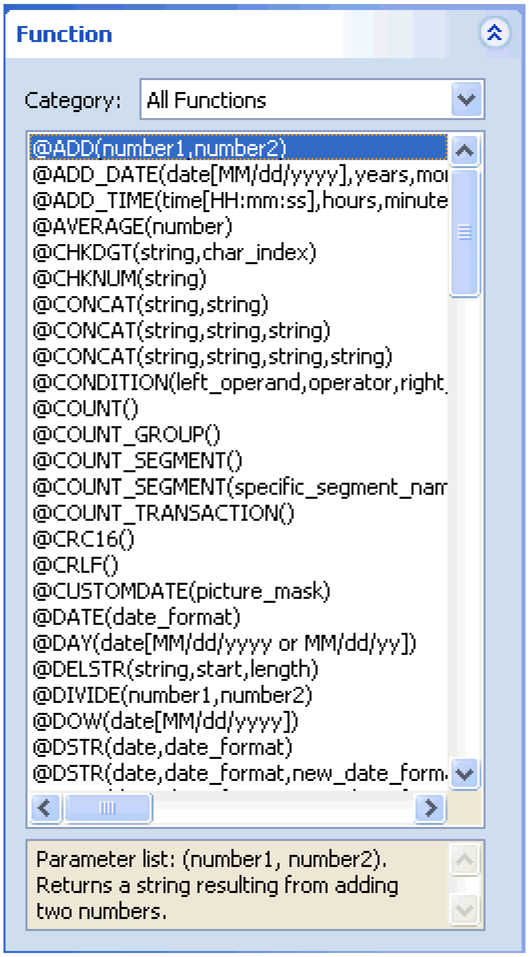


View and configure properties for the Attribute node.

Name: type
Type: Attribute
Mapping: Sales/Company/Year/Quarter/Product@type

Function

Function value can be from predefined or custom algorithm of calculation. Mapping to a Function can be done using Function pane. This pane is typically located in the upper-left corner of the Mapping Builder. It allows you to select predefined functions or custom functions that you defined to associate with an output node.



```
@ADD(number1,number2)
@ADD_DATE(date[MM/dd/yyyy],years,mo
@ADD_TIME(time[HH:mm:ss],hours,minute
@AVERAGE(number)
@CHKDGT(string,char_index)
@CHKNUM(string)
@CONCAT(string,string)
@CONCAT(string,string,string)
@CONCAT(string,string,string,string)
@CONDITION(left_operand,operator,right
@COUNT()
@COUNT_GROUP()
@COUNT_SEGMENT()
@COUNT_SEGMENT(specific_segment_nam
@COUNT_TRANSACTION()
@CRC16()
@CRLF()
@CUSTOMDATE(picture_mask)
@DATE(date_format)
@DAY(date[MM/dd/yyyy or MM/dd/yy])
@DELSTR(string,start,length)
@DIVIDE(number1,number2)
@DOW(date[MM/dd/yyyy])
@DSTR(date,date_format)
@DSTR(date,date_format,new_date_form
```

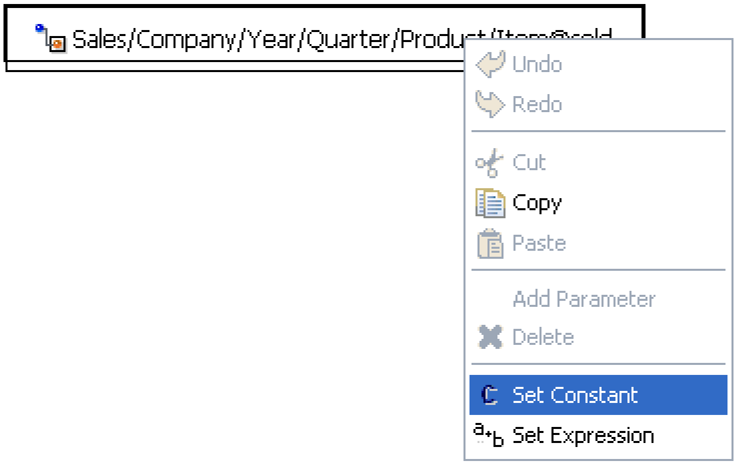
Parameter list: (number1, number2). Returns a string resulting from adding two numbers.

You can drag and drop the function into the working area of the Mapping Builder.

Note: In this version of iIT Transformer, you have an option to insert functions below the current function in the existing mapping tree or replace the current function.

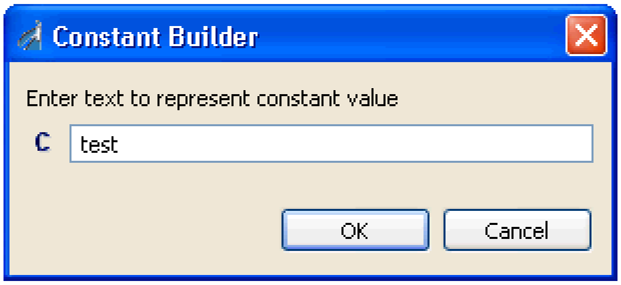
Constant

Allows you to type in a fixed value for a function or input node, by right-clicking the node and selecting *Set Constant* or double-clicking the node.



Make sure the *Set Constant* option is selected.

The Constant Builder dialog box opens, as shown in the following image.

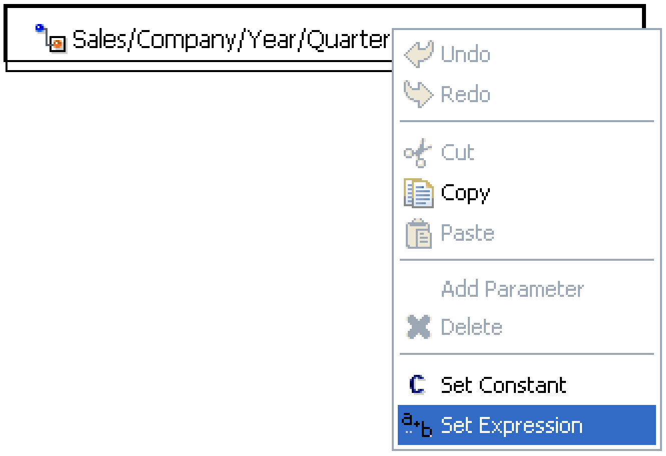


Type your value in the Constant field and click *OK*.

Expression

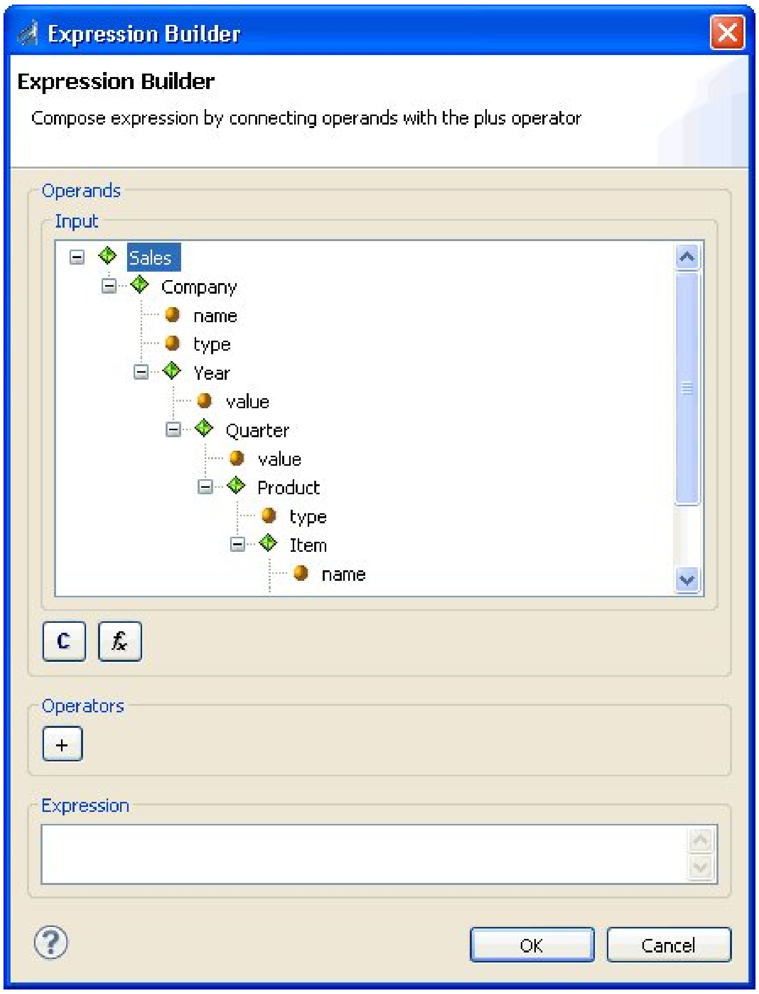
An expression is a logical statement where two or more input node values are combined into one output value using the basic operators and functions. In Mapping Builder, expressions are typically used for concatenation where input fields are joined by a plus sign "+" operator.

To set an expression, right-click the node and select *Set Expression* or double-click the node.



Make sure the *Set Expression* option is selected.

The Expression Builder dialog box opens, as shown in the following image.



Type your value in the Expression field and click *OK*.

You can also use the buttons that are available to build your expression graphically.

An expression can contain a concatenation of any number of input nodes and constant values in any order, in the following format:

```
<input item>+'constant'+...
```

For example:

```
'For '+tree/fruit+tree/fruit/year+': '+tree/fruit/
apple/yield+
' bushels, '+tree/fruit/apple/avgweight+' grams avg.'
```

Note: It is strongly encouraged to use the @CONCAT function instead of an expression to concatenate strings. For more information on using the @CONCAT function and its properties, see *Working With Functions* on page 269.

For more information on how to use the Mapping Builder, see *Using the Mapping Builder* on page 269.

Building and Altering Output Structures

For most output data formats, you can load an existing output structure from the metadata configuration of your project. The output structure will appear in the Output pane. Also, output structures typically can be altered once in the Output pane, but there are some that should never be altered.

It is *not* recommended to alter the output structure in the Output pane for the following formats:

- EDI HIPAA
- EDI X12
- EDIFACT
- IDOC

These output data formats require a mandatory metadata structure to be provided for the message. If you alter the output structure in the Output pane of the Mappings tab for any of these outputs, even by one character, the resulting transformation will not work properly.

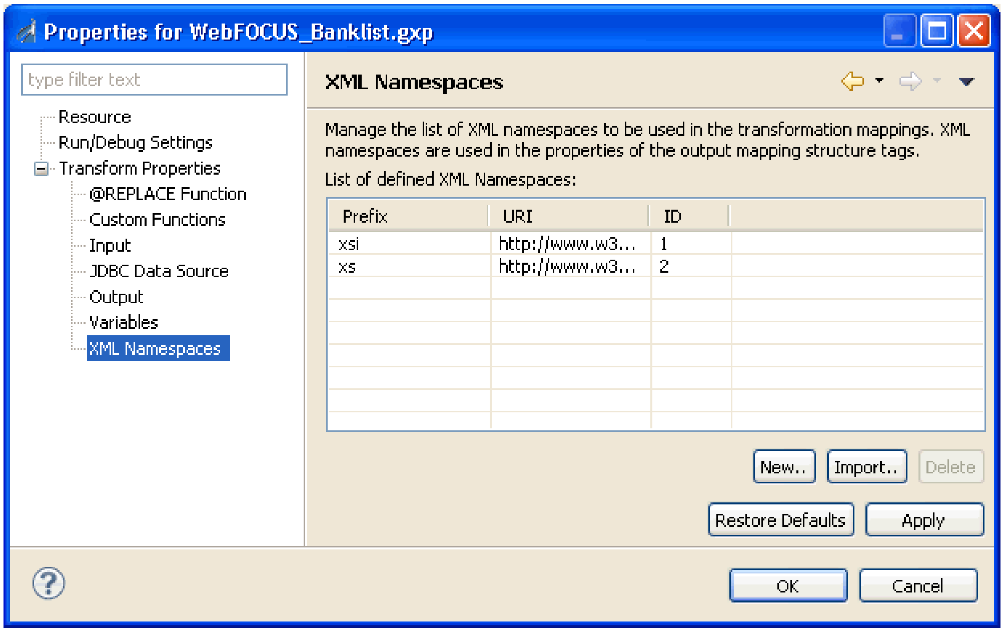
Working With Namespaces

Procedure: How to Load XML Namespaces From Another Transform Component

To load XML namespaces from another Transform component:

1. In the Project Properties dialog box, select the *XML Namespaces* category in the left pane.

The following image shows the XML Namespaces category in the Project Properties dialog box of a Transform component with areas for prefix, URI, and ID values.



XML Namespaces

Manage the list of XML namespaces to be used in the transformation mappings. XML namespaces are used in the properties of the output mapping structure tags.

List of defined XML Namespaces:

Prefix	URI	ID
xsi	http://www.w3...	1
xs	http://www.w3...	2

2. Click *Import*.

   The Open dialog box is displayed.

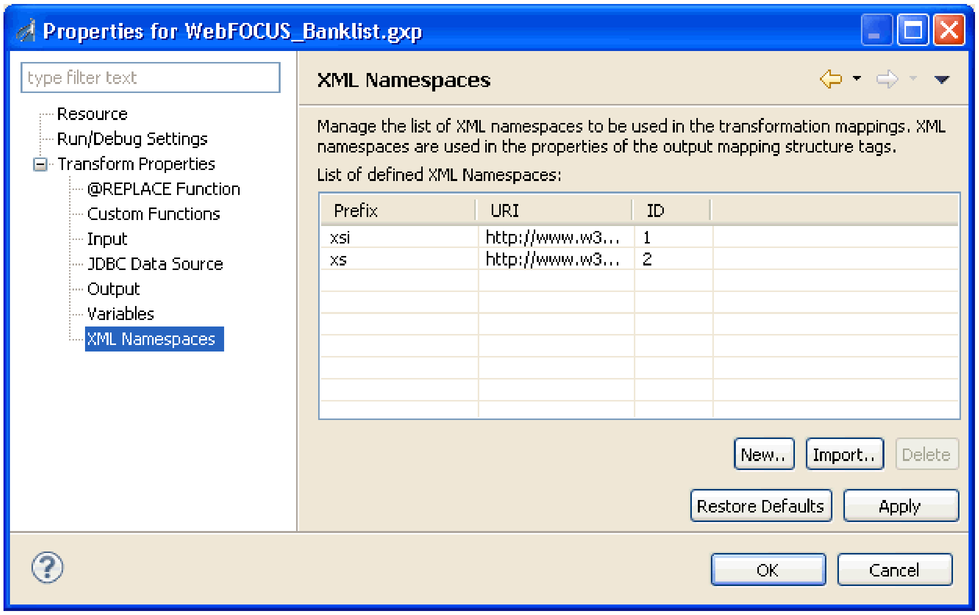
3. Find and select the Transform component whose namespace you wish to load and click *Open*.

Procedure: How to Create XML Namespaces for a Transform

To create XML namespaces for a Transform:

1. In the Project Properties dialog box, click the *XML Namespaces* category in the left pane.

The following image shows the XML Namespaces category with columns for prefix, URI, and ID values.



XML Namespaces

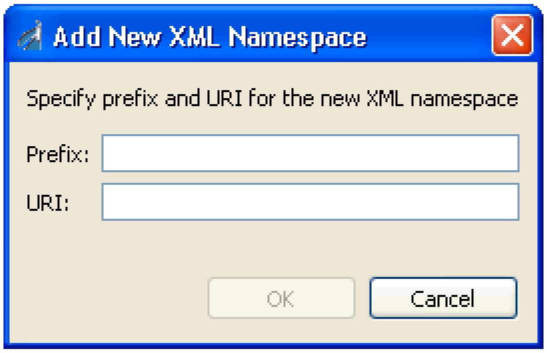
Manage the list of XML namespaces to be used in the transformation mappings. XML namespaces are used in the properties of the output mapping structure tags.

List of defined XML Namespaces:

Prefix	URI	ID
xsi	http://www.w3...	1
xs	http://www.w3...	2

2. Click *New*.

   The Add New XML Namespace dialog box opens.



Specify prefix and URI for the new XML namespace

Prefix:	
URI:	

3. In the Prefix field, type the prefix to be associated with the XML namespace.
4. In the URI field, type the URI associated with the prefix and use the URI to represent the stylesheet, reference, or value for the XML namespace.

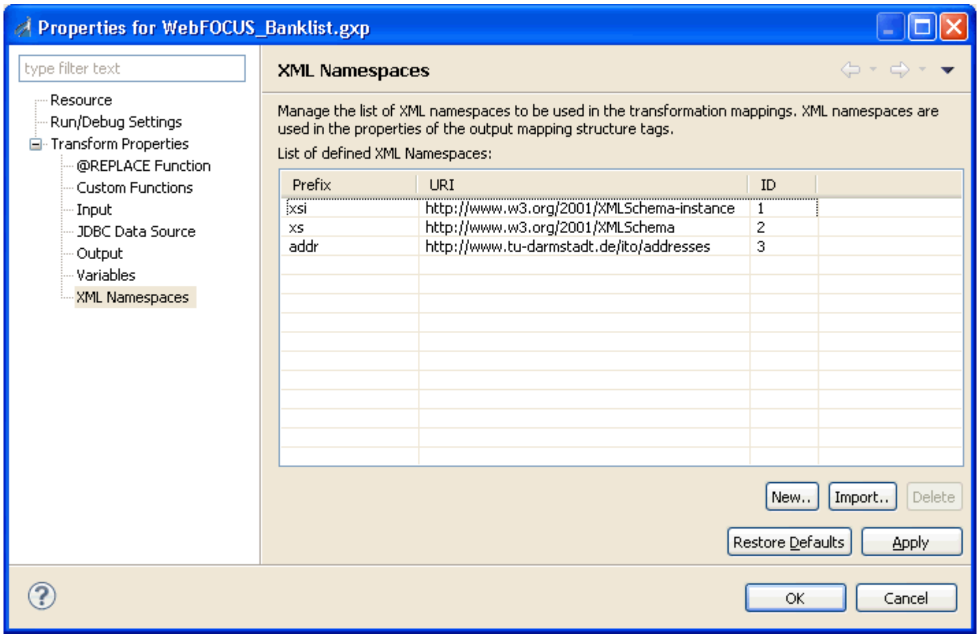
   For example, a prefix value of *addr* with the following URI:

   `http://www.tu-darmstadt.de/ito/addresses` is the equivalent of including the following in your actual data:

```
xmlns:addr="http://www.tu-darmstadt.de/ito/addresses"
```

5. Click *OK*.

   The new XML namespace is added to the list.



XML Namespaces

Manage the list of XML namespaces to be used in the transformation mappings. XML namespaces are used in the properties of the output mapping structure tags.

List of defined XML Namespaces:

Prefix	URI	ID
xsi	http://www.w3.org/2001/XMLSchema-instance	1
xs	http://www.w3.org/2001/XMLSchema	2
addr	http://www.tu-darmstadt.de/ito/addresses	3

An identification number for each XML namespace appears automatically set in the ID column. Identification numbers are unique identifiers used by iIT Transformer to manage your XML namespaces within the workspace.

6. Click *OK*.

   Note: To attach XML namespaces, left-click the appropriate node in the Output pane and select *Properties*. When the Output Node Properties dialog box opens, click the *XML Namespace* tab (if it is not already active) and then, select the *Apply Namespace URI* check box.

Working With Functions

The iIT Transformer interface offers an integrated set of useful functions, which are used to perform a variety of operations, such as calculation and manipulation of input data, or retrieving data from a database. You may use functions to modify the output node value to match your requirements. You can build complex expressions for output values using the combinations of functions via the Mapping Builder.

In addition, iIT Transformer enables you to implement your own algorithms for producing output values by adding custom functions according to the specific format. For more information, see *Custom Functions* on page 297.

Using the Mapping Builder

The Mapping Builder is a tool available in iIT Transformer as part of the Mapping Builder facility. It allows you to construct an output node using various methods and formulas implemented in iIT Transformer as functions. You can access the predefined functions and the custom functions that you defined, or build the statement containing several functions if needed.

In iIT Transformer, the predefined functions are subdivided into the following categories:

- All Functions
- Custom Functions
- EDI Functions
- Miscellaneous Functions
- Numeric Functions
- Processing Functions
- Runtime Functions
- SWIFT Functions
- Security Functions
- String Functions
- Time Functions

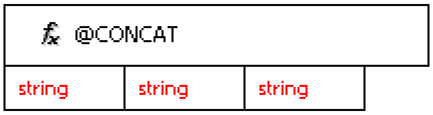
Setting Function Parameters

After selecting a specific function, typically you will need to define the set of parameters associated with it. The number and the names of the parameters for the particular function can be determined in either of the following ways:

- ❏ The function expression in the list of functions.

  @SUBTRACT (number1, number2)

- ❏ The function description area located below the list of functions.
- ❏ The right pane of the Mapping Builder. For example, the @CONCAT function shown in the following image accepts three parameters of type string.



For more details on the parameters to specify for each particular function, see *Predefined Functions* on page 102 for a complete reference. If dealing with custom functions, you may consult the author of the function.

Generally, the functions can take four types of values as their parameters:

- ❏ Input nodes
- ❏ Nested functions
- ❏ Constant values
- ❏ Expressions

A function of a parameter can take the value of a specified input node. For information on how to set an input node, see *How to Set an Input Node* on page 275.

You can nest functions within functions, which means the output result of a certain function can serve as the input parameter for another function. For more details, see *How to Nest Functions* on page 276.

You can also set a specific invariable value as a constant function parameter. For information on how to set a constant value, see *How to Set a Constant Value for a Function* on page 277.

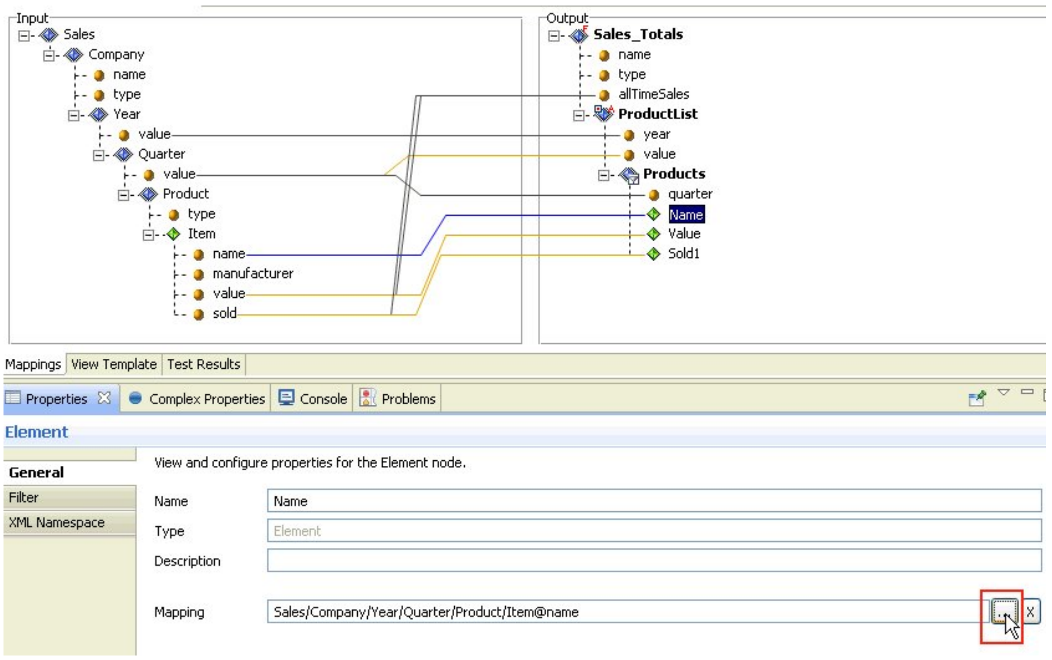
Another option for a function parameter type is an expression which is a basic combination of operators, input nodes, and/or constants. For information on how to set an expression, see *How to Set an Expression for a Function* on page 277.

Predefined Functions

The Mapping Builder enables you to set the output node values using a combination of functions and/or input nodes. Both custom and predefined functions can be utilized. When specifying a function through the Mapping Builder, you must know what the specific function does and how to specify the parameters for it. For more information, see *Predefined Functions* on page 102.

Procedure: How to Open the Mapping Builder

The following image shows the Mapping Builder interface in iIT Transformer.



View and configure properties for the Element node.

Name: Name

Type: Element

Description:

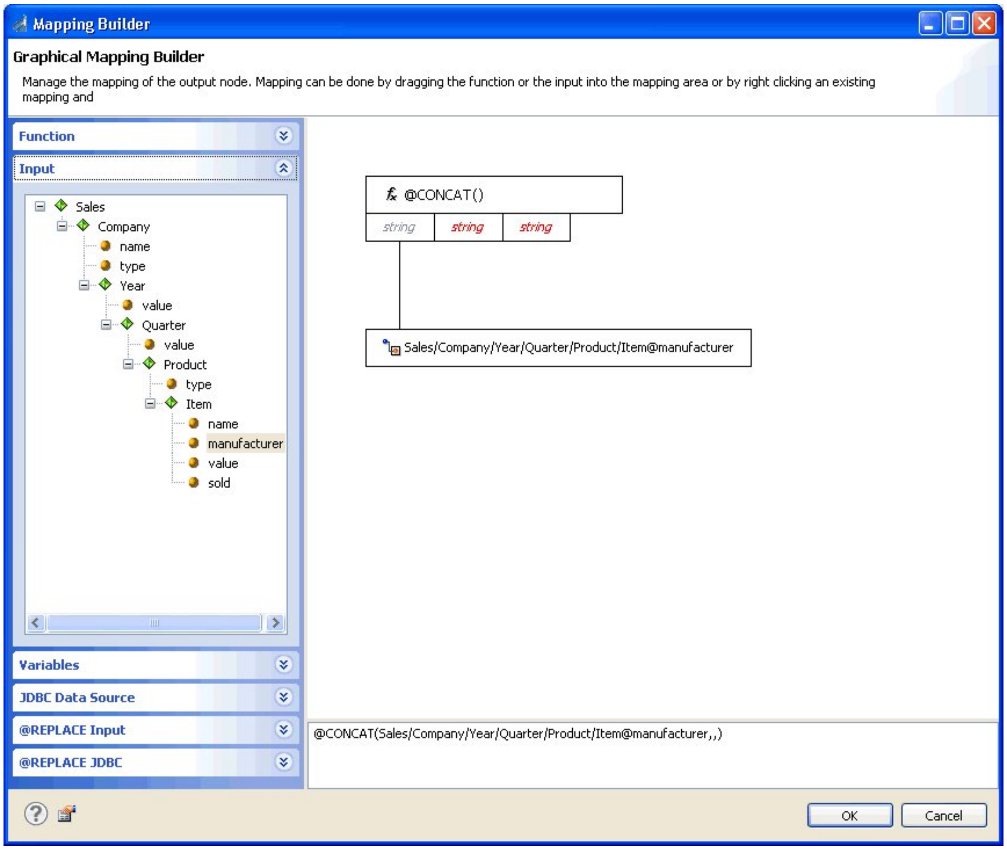
Mapping: Sales/Company/Year/Quarter/Product/Item@name

To open the Mapping Builder:

1. In the Mapping Values pane of the *Mappings* tab, select an output node.

2. Click the ellipsis ... button.

The Mapping Builder displays as shown in the following image.



Graphical Mapping Builder

Manage the mapping of the output node. Mapping can be done by dragging the function or the input into the mapping area or by right clicking an existing mapping and

```
@CONCAT(Sales/Company/Year/Quarter/Product/Item@manufacturer,,)
```

The left pane of the Mapping Builder dialog box provides you with the option of switching between the following areas:

- ❏ Functions (selected by default)
- ❏ Input
- ❏ Variables
- ❏ JDBC Data Source
- ❏ @REPLACE Input
- ❏ @REPLACE JDBC

Click the name of the corresponding pane to expand its content. Click *New* to add new values to the Variables and JDBC Data Source panes.

This feature allows you to drag and drop any available configuration that was specified using the Project Properties dialog box into the workspace area of the Mapping Builder. You can switch between the available panes by clicking on the corresponding button, for example, *Variables*, *JDBC Data Source*, and so on.

Procedure: How to Select a Function

To select a function:

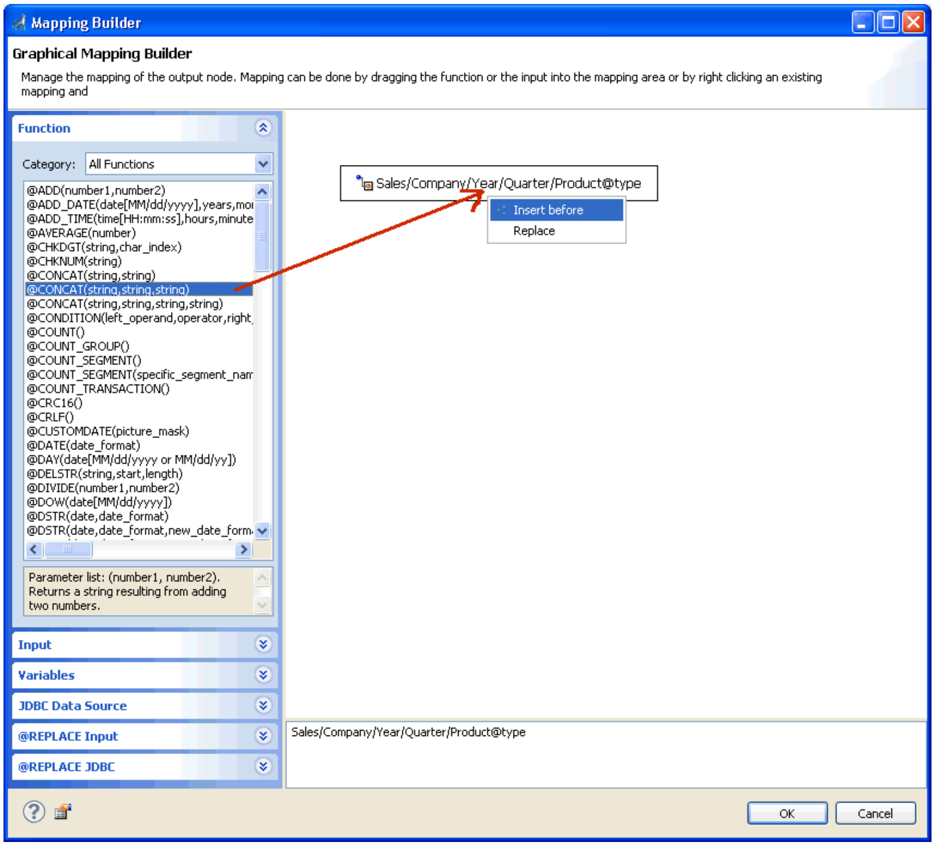
1. From the Category drop-down list in the Functions pane of the Mapping Builder, select the function category you want to use, for example, String.

   For a list of function categories, see *Using the Mapping Builder* on page 269.

   The following image shows String Functions selected in the Category drop-down list, which is located in the Functions pane.

   Optionally, you can skip to step 2 and select your function from the whole list of available functions.

2. Select the specific function you want to use, for example, @CONCAT(string, string, string), and drag it into the workspace area of the Mapping Builder as shown below:



Graphical Mapping Builder

Manage the mapping of the output node. Mapping can be done by dragging the function or the input into the mapping area or by right clicking an existing mapping and

Category: All Functions

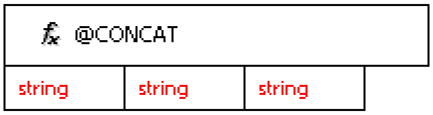
```
@ADD(number1,number2)
@ADD_DATE(date[MM/dd/yyyy],years,mo
@ADD_TIME(time[HH:mm:ss],hours,minute
@AVERAGE(number)
@CHKDGT(string,char_index)
@CHKNUM(string)
@CONCAT(string,string)
@CONCAT(string,string,string)
@CONCAT(string,string,string,string)
@CONDITION(left_operand,operator,right
@COUNT()
@COUNT_GROUP()
@COUNT_SEGMENT()
@COUNT_SEGMENT(specific_segment_nam
@COUNT_TRANSACTION()
@CRC16()
@CRLF()
@CUSTOMDATE(picture_mask)
@DATE(date_format)
@DAY(date[MM/dd/yyyy or MM/dd/yy])
@DELSTR(string,start,length)
@DIVIDE(number1,number2)
@DOW(date[MM/dd/yyyy])
@DSTR(date,date_format)
@DSTR(date,date_format,new_date_form
```

Parameter list: (number1, number2). Returns a string resulting from adding two numbers.

Sales/Company/Year/Quarter/Product@type

Note: You can drag and drop functions above or below the existing mapping tree in the workspace area as needed to create a combination of output values.

The function you choose becomes your working function.

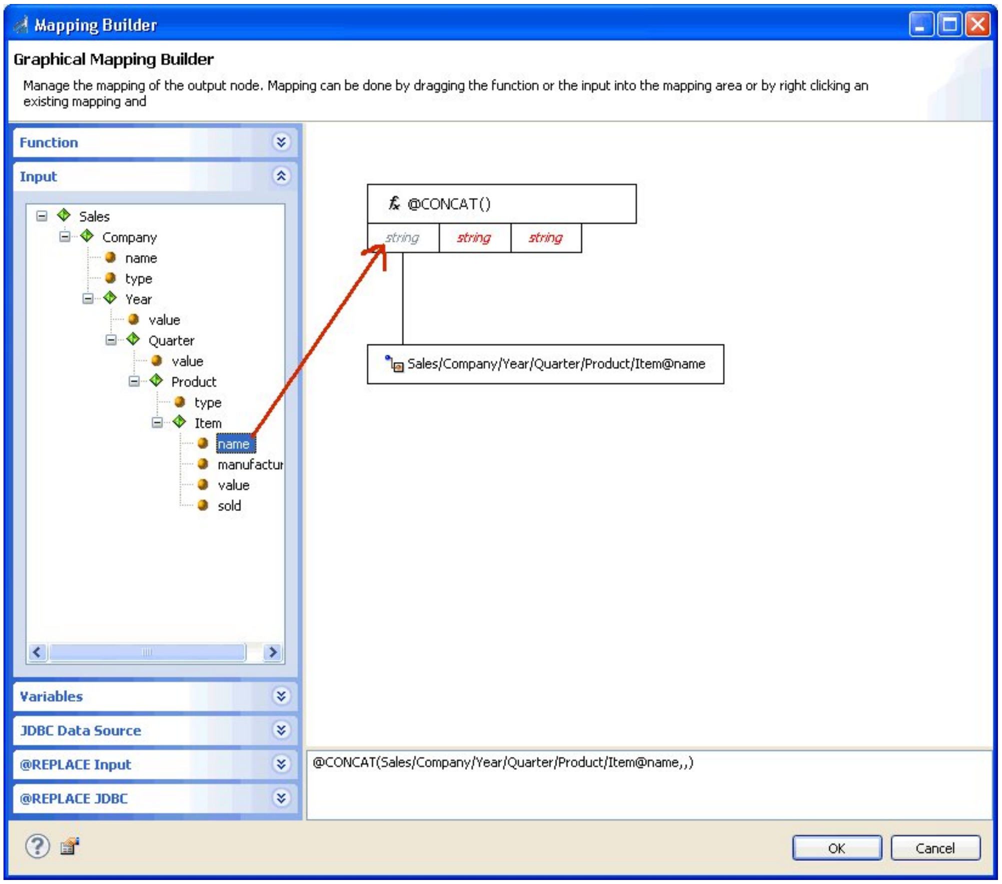


3. To replace the working function, drag a new function on the workspace and drop it directly onto the working function.

Procedure: How to Set an Input Node

To set an input node as the value for a function parameter:

1. Select an input node from the Input document tree area of the Mapping Builder.



Manage the mapping of the output node. Mapping can be done by dragging the function or the input into the mapping area or by right clicking an existing mapping and

```
@CONCAT(Sales/Company/Year/Quarter/Product/Item@name,,)
```

2. Drag the input node onto the appropriate parameter in the workspace area of the Mapping Builder.

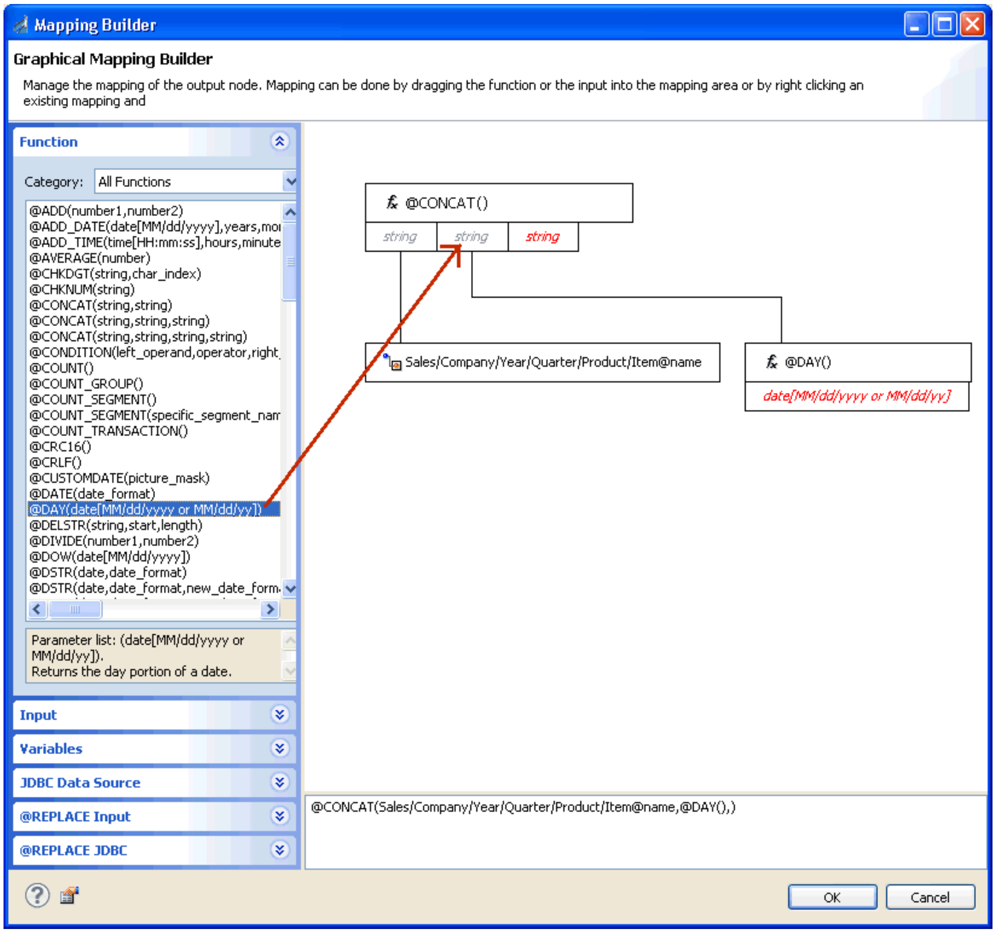
   An alternate way to perform this task is to use *Set Selected Input* option from the context menu, which appears when you right-click the parameter in the workspace area.

   Note: Another method for editing the mapping formula of your node is to type it directly in the preview area located on the lower-right corner of the Mapping Builder window and press *Enter*. The tree structure will be refreshed. However, this option is recommended for advanced users only. Using the drag and drop interface instead is strongly encouraged.

Procedure: How to Nest Functions

To specify another function as a parameter for your working function:

1. Select the function you want from the Functions pane of the Mapping Builder.



Manage the mapping of the output node. Mapping can be done by dragging the function or the input into the mapping area or by right clicking an existing mapping and

Category: All Functions

```
@ADD(number1,number2)
@ADD_DATE(date[MM/dd/yyyy],years,mo
@ADD_TIME(time[HH:mm:ss],hours,minute
@AVERAGE(number)
@CHKDGT(string,char_index)
@CHKNUM(string)
@CONCAT(string,string)
@CONCAT(string,string,string)
@CONCAT(string,string,string,string)
@CONDITION(left_operand,operator,right
@COUNT()
@COUNT_GROUP()
@COUNT_SEGMENT()
@COUNT_SEGMENT(specific_segment_nam
@COUNT_TRANSACTION()
@CRC16()
@CRLF()
@CUSTOMDATE(picture_mask)
@DATE(date_format)
@DAY(date[MM/dd/yyyy or MM/dd/yy])
@DELSTR(string,start,length)
@DIVIDE(number1,number2)
@DOW(date[MM/dd/yyyy])
@DSTR(date,date_format)
@DSTR(date,date_format,new_date_form
```

Parameter list: (date[MM/dd/yyyy or MM/dd/yy]).
Returns the day portion of a date.

@CONCAT() — string, string, string

Sales/Company/Year/Quarter/Product/Item@name

@DAY() — date[MM/dd/yyyy or MM/dd/yy]

```
@CONCAT(Sales/Company/Year/Quarter/Product/Item@name,@DAY(),)
```

2. Drag the function and drop it on the appropriate parameter of the working function on the workspace area of the Mapping Builder.

   An alternative way to perform this task is to use *Set Selected Function* option from the context menu, which appears when you right-click the parameter in the workspace area.

   You must set the parameters for the nested function as well.

   Note: You can nest functions multiple times.

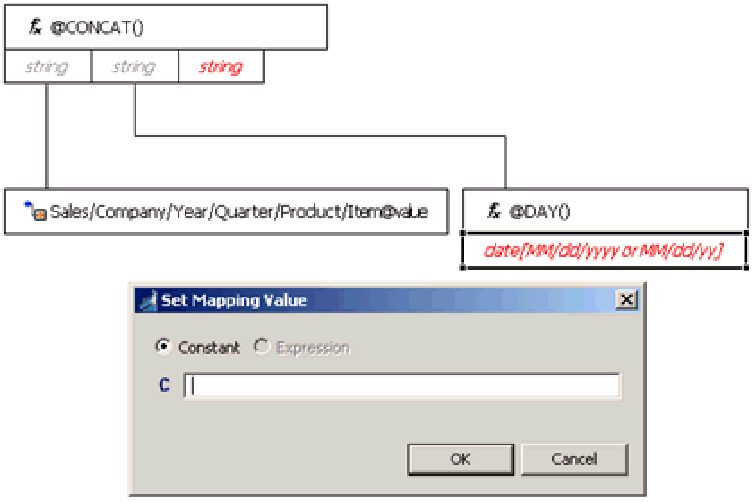
Procedure: How to Set a Constant Value for a Function

To set a constant value:

1. Right-click the appropriate parameter box in the workspace area of the Mapping Builder and select *Set Constant* from the context menu.

   You can also double-click the parameter box to set a constant.

   The following image shows the Constant option selected and a field where you can enter a value.



2. Type the value for your constant and click *OK*.

Procedure: How to Set an Expression for a Function

To set an expression:

1. Right-click the appropriate parameter box in the workspace area of the Mapping Builder and select *Set Expression* from the context menu.

   You can also double-click the parameter box and select the Expression radio button to set an expression.

2. Type the value for your expression and press *Enter*.

   An expression concatenation is typically used for any number of input items and constant values in any order, in the following format:

```
<input item>+'constant'+...
```

For example:

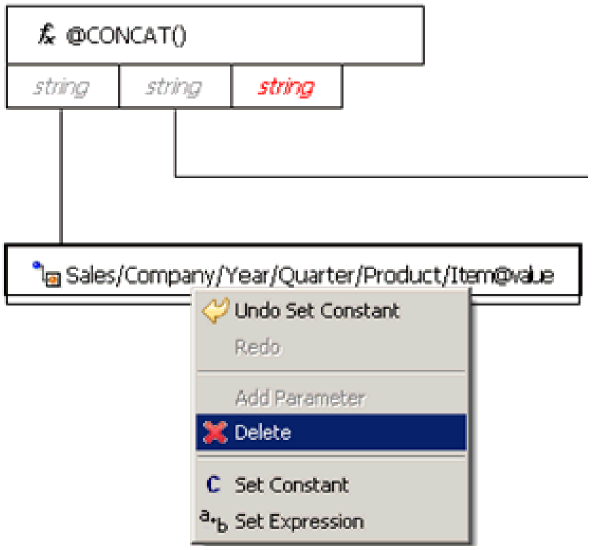
```
'For '+tree/fruit+tree/fruit/year+': '+tree/fruit/apple/yield+
' bushels, '+tree/fruit/apple/avgweight+' grams avg.'
```

Procedure: How to Remove a Function Parameter

To remove a value for a function parameter:

1. Right-click the parameter in the workspace area of the Mapping Builder.

   A context menu opens as shown in the following image.



2. Select *Delete*.

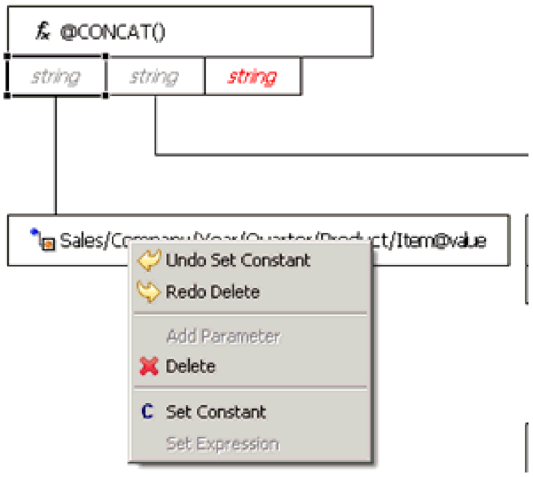
   The parameter is removed from the function.

Procedure: How to Cancel and Restore an Action

To cancel and restore an action within the Mapping Builder:

1. To cancel your last action, right-click any component in the workspace, for example, the working function.

A context menu opens as shown in the following image.

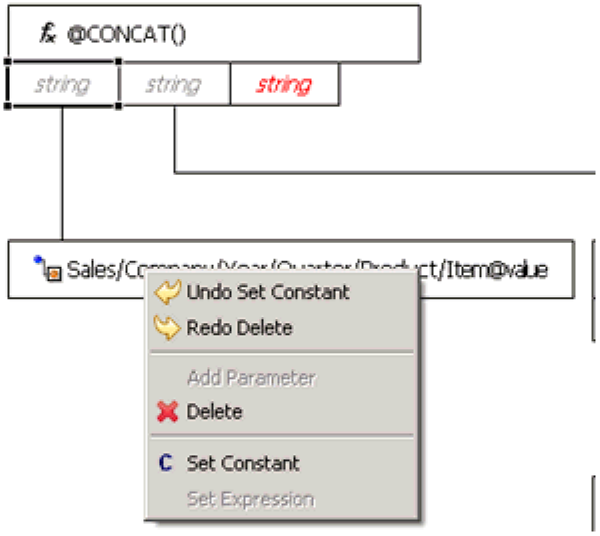


2. Select *Undo* or use the keyboard shortcut *Ctrl+Z*.

   The last action is removed from the workspace.

3. If you decide to restore an action, right-click any component in the workspace, for example, the output item.

   A context menu opens as shown in the following image.



4. Select *Redo* or use the keyboard shortcut *Ctrl+Y*.

   The last removed action is returned to the workspace.

   Note: Multiple levels of undo and redo are supported. You can select Undo/Redo or use the corresponding keyboard shortcut (Ctrl+Z/Ctrl+Y) multiple times until you have achieved the desired result.

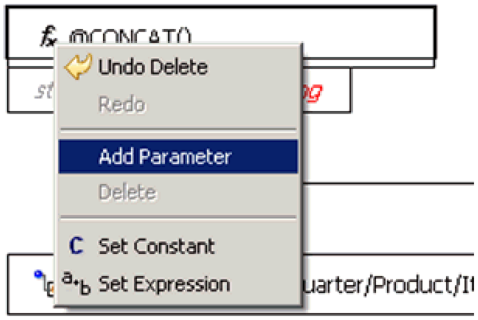
Customizing @CONCAT Functions

You can customize the signature of your @CONCAT function for a specific output node by adding or removing parameters as required using the Mapping Builder. This functionality adds flexibility to the design of mapping values, which involve complex string concatenations; it also eliminates the need to use expressions for the concatenation of strings in iIT Transformer.

Procedure: How to Add Parameters for @CONCAT Functions

To add parameters for @CONCAT functions:

1. In the Mapping Builder, right-click the @CONCAT function in the workspace area.



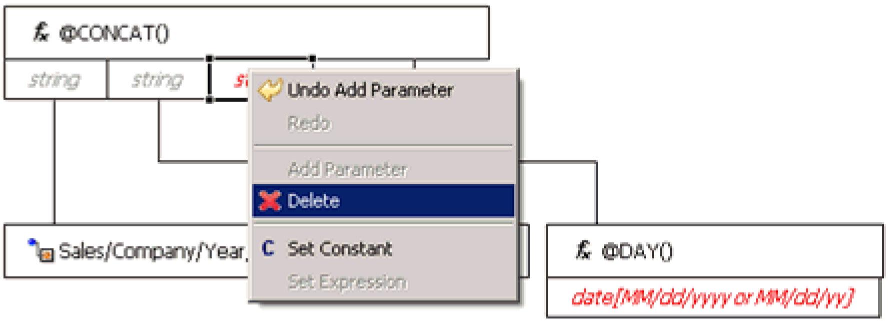
2. Select *Add Parameter* from the context menu.

   The parameter is added to the parameter list, and the number of parameters for this function definition will increase by one.

Procedure: How to Remove Parameters for @CONCAT Functions

To remove parameters for @CONCAT functions:

1. In the Mapping Builder, right-click a parameter to delete from the @CONCAT function in the workspace area.



2. Select *Delete* from the context menu.

   The parameter is removed from the parameter list, and the number of parameters for this particular function definition will decrease by one.

Configuring Properties for JDBC

A JDBC replace function is a replace function in which the match and replace values are taken from columns of the specified database table or from the results of an SQL statement run on the specified database. For example, you can configure iIT Transformer to use a database customer information table, reading the value from the "Full Name" column and replacing it with the value from the "Nickname" column. Unlike the regular replace functions where the match and replace pairing is static, the JDBC replace functions read the data from the specified columns of the next database row each time the transformation processing loops past the JDBC replace function while producing your output document.

Replace functions are similar to custom functions in that you must first define the function and then, apply it to the output node mapping value definition that you want to affect. In the case of replace functions, this can be accomplished using the @REPLACE and @SIMPLE_REPLACE functions.

For more advanced database functionality, such as matching across multiple input nodes or running complex SQL queries, see *Using the @JDBCLOOKUP Function* on page 286.

The process of defining a particular replace function accounts for setting up the value to match, but not the mapping of it. You must map the replace functions to the specific node using the Mapping Builder. For more information, see *Using the Mapping Builder* on page 269.

JDBC Replace Functions as Input Data Sources

Since a JDBC replace function can be defined and used no matter what the data format of your input is, by using JDBC replace functions you can access multiple input data sources. For example, if the input data format of your transform is XML, your incoming document is likely an XML file. By defining and using JDBC replace functions, you can read additional input data from the database. In fact, if you define multiple JDBC replace functions, you can use as many database sources as you like.

Procedure: How to Define a JDBC Lookup

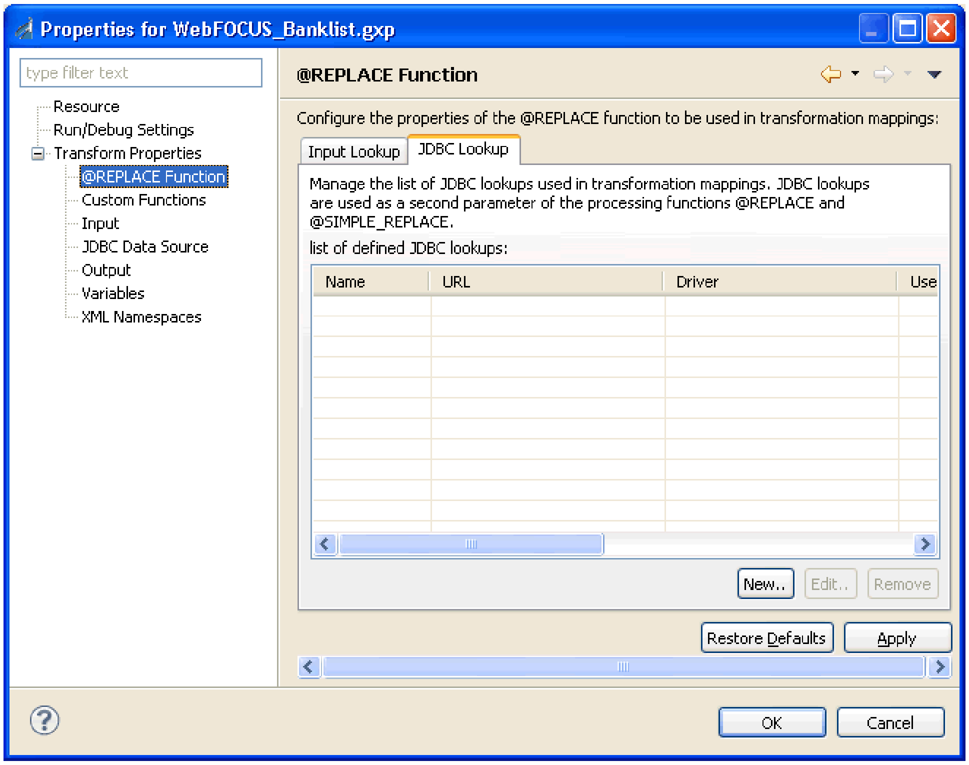
To define a JDBC lookup:

1. Select the Transform component and then, *Properties* from the Integration Project context menu.

   The Properties dialog box opens.

2. Expand *Transform Properties* in the left pane and then select the *@REPLACE Function* category.

The following image shows the Project Properties dialog box with the @REPLACE Functions category selected in the left pane.



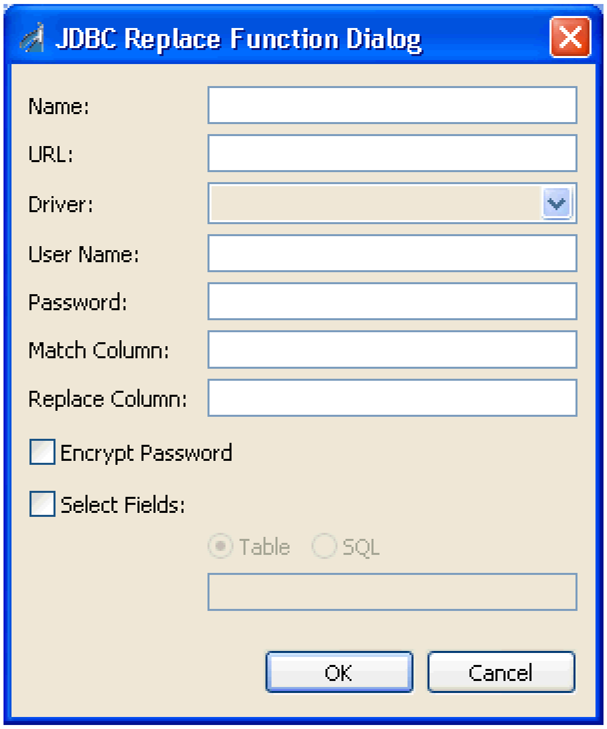
Configure the properties of the @REPLACE function to be used in transformation mappings:

Manage the list of JDBC lookups used in transformation mappings. JDBC lookups are used as a second parameter of the processing functions @REPLACE and @SIMPLE_REPLACE.

list of defined JDBC lookups:

3. Click the *JDBC Lookup* tab.
4. Click *New*.

The JDBC Replace Function Dialog opens.



To access the particular database, you must provide the necessary information that is required for your JDBC lookup.

a. In the Name field, specify a name for the JDBC lookup.

b. In the URL field, specify the database URL.

   Note: To define a predefined function (for example, @SREG) as the URL, click the *Browse* button to the right of the URL field, which opens the Mapping Builder.

c. From the Driver drop-down list, select an available JDBC driver.

   Note: iIT Transformer does not ship with any of the predefined drivers. The driver must be registered with iIT Transformer.

d. Specify a valid user name and password for accessing the database.

e. In the Match Column field, type the name of the source match column.

f. In the Replace Column field, specify either a database table name or an SQL statement to run on the database.

g. From the Select Fields drop-down list, select one of the following methods to acquire the match and replace values:

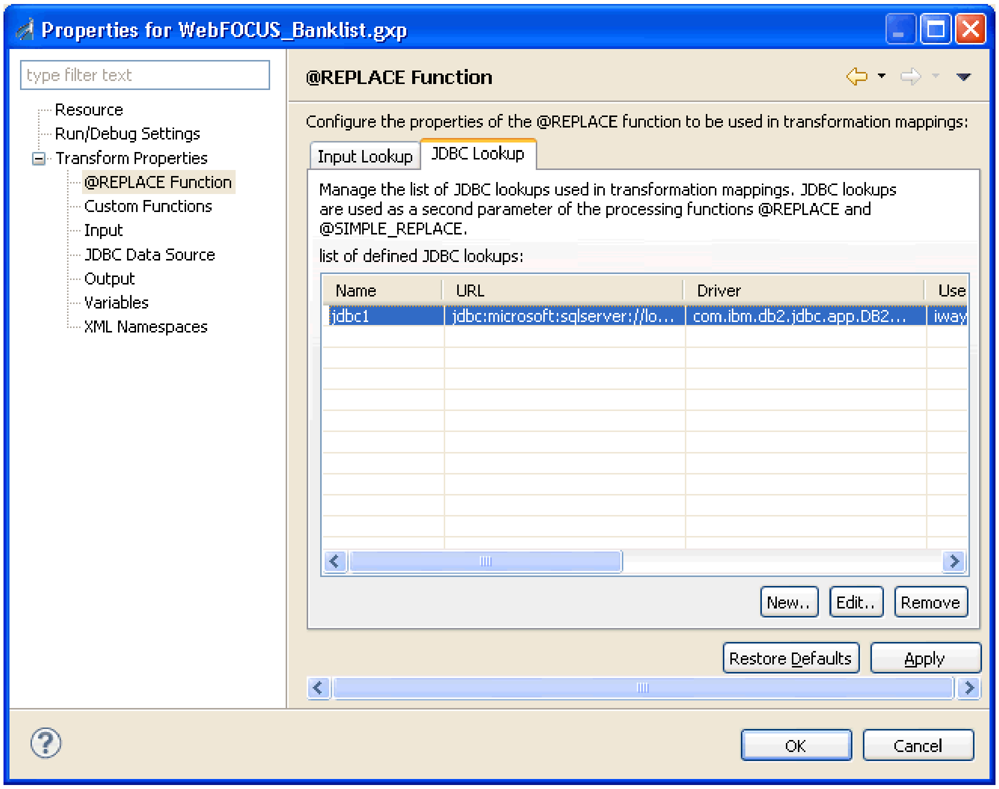
If you select *From Table*, specify a database table name in the Table Name field.

If you select *From SQL Query*, specify an SQL statement to run on the database in the SQL Query field.

If you specify an SQL statement, the match and replace values are taken from the result of the SQL statement.

Procedure: How to Modify a JDBC Lookup

To change or update any information in your JDBC lookup:



Configure the properties of the @REPLACE function to be used in transformation mappings:

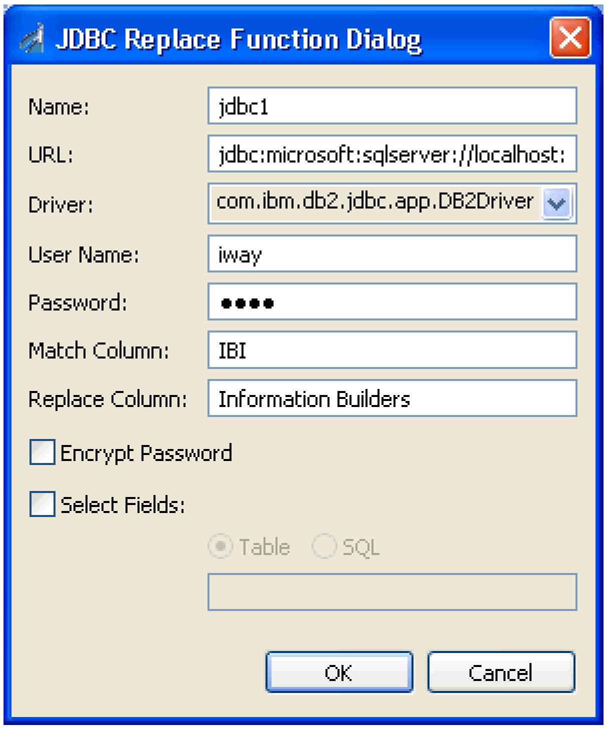
Manage the list of JDBC lookups used in transformation mappings. JDBC lookups are used as a second parameter of the processing functions @REPLACE and @SIMPLE_REPLACE.

list of defined JDBC lookups:

Name	URL	Driver	Use
jdbc1	jdbc:microsoft:sqlserver://lo...	com.ibm.db2.jdbc.app.DB2...	iway

1. Select the JDBC lookup, for example, jdbc1, and click *Edit*.

The JDBC Replace Function Dialog opens.


JDBC Replace Function Dialog

Name:	jdbc1
URL:	jdbc:microsoft:sqlserver://localhost:
Driver:	com.ibm.db2.jdbc.app.DB2Driver
User Name:	iway
Password:	●●●●
Match Column:	IBI
Replace Column:	Information Builders

☐ Encrypt Password

☐ Select Fields:

● Table ○ SQL

2. Click *OK* once you have completed the necessary modifications.

Procedure: How to Delete a JDBC Lookup

To delete a JDBC lookup, select the JDBC lookup, for example, JDBC1, and click *Remove*.

The JDBC lookup is removed from the list.

Using the @JDBCLOOKUP Function

The @JDBCLOOKUP function returns a matched value from a database using an SQL statement. It takes the JDBC data source name and SQL statement in string form as parameters. The SQL statement can be set dynamically based on the input from other functions within iIT Transformer. If more than one value is matching the query, then the last matching value is returned.

When no result is found in the database, an empty string is returned. When more than one result is found, the last matching result is returned. Regardless of the number of columns in the result set, the value of the first column is always returned. During the transformation with multiple occurrences of @JDBC lookup, if the SQL statements are the same, then the returned result will be the same.

Parameters:

JDBC_data_source_name: String that represents a globally defined JDBC connection configuration.

SQL_statement: SQL statement that can be created using SQL Builder.

Example:

```
@JDBCLOOKUP('LOOKUP_TEST', {'SELECT field1 FROM LOOKUP_TABLE WHERE field2 '
= '+@QUOTE(Customer/Person/Name)})})
```

where:

`LOOKUP_TEST`

Is the name of a JDBC connection configuration.

`{'SELECT field1 FROM LOOKUP_TABLE WHERE field2 ' = '+@QUOTE(Customer/Person/Name)}`

Is an SQL statement.

Procedure: How to Create an SQL Statement Using SQL Builder

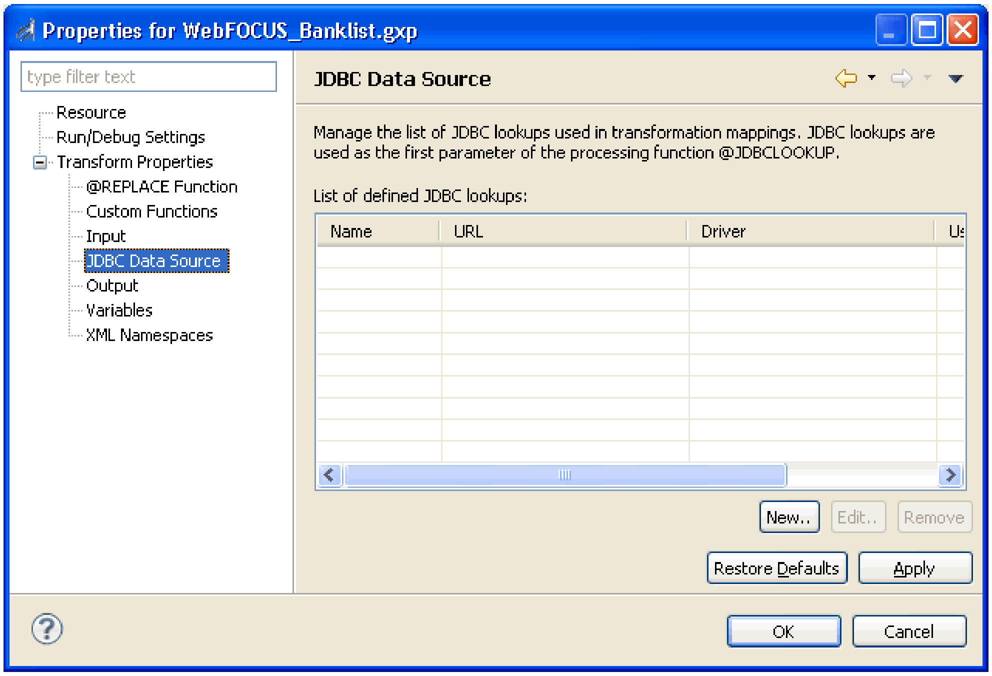
To create an SQL statement, which can be used for the @JDBCLOOKUP function, you must follow the instructions below:

1. From the menu bar, click *Project* and select *Properties*.

   The Properties dialog box opens.

2. Click the *JDBC Data Source* category in the left pane.

The following image shows the Properties dialog box with the JDBC Data Source category selected in the left pane.

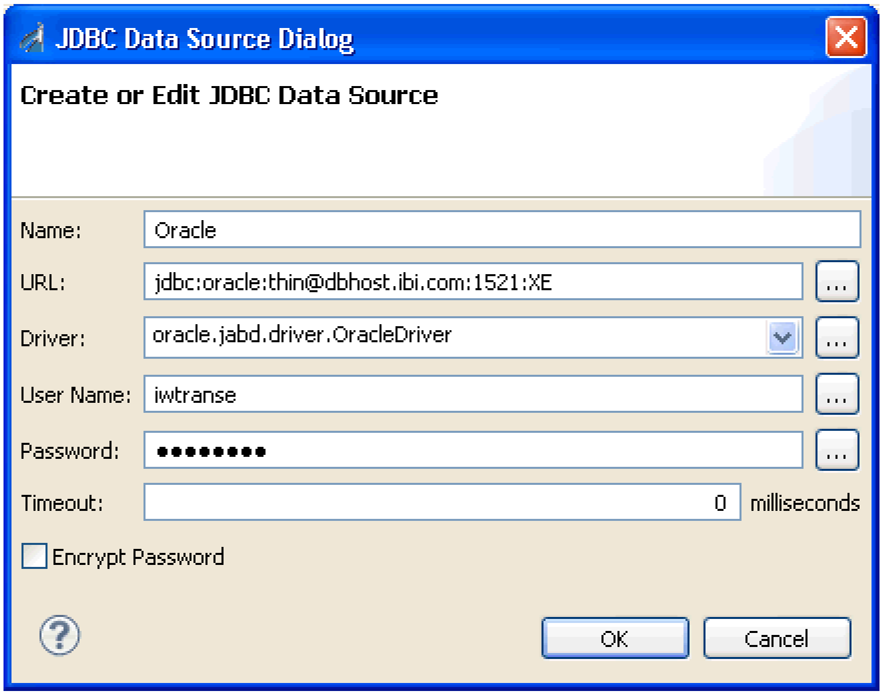


Manage the list of JDBC lookups used in transformation mappings. JDBC lookups are used as the first parameter of the processing function @JDBCLOOKUP.

List of defined JDBC lookups:

3. Click *New*.

The JDBC Data Source Dialog opens.



JDBC Data Source Dialog

Create or Edit JDBC Data Source

Name:	Oracle
URL:	jdbc:oracle:thin@dbhost.ibi.com:1521:XE
Driver:	oracle.jabd.driver.OracleDriver
User Name:	iwtranse
Password:	●●●●●●●●
Timeout:	0 milliseconds

☐ Encrypt Password

Perform the following steps:

a. In the Name field, provide a name for the JDBC connection.

   This name is used as a string value for the *JDBC_data_source_name* parameter in the @JDBCLOOKUP function.

b. In the URL field, type the URL to the database.

   Note: To define a predefined function (for example, @SREG) as the URL, click the *Browse* button to the right of the URL field, which opens the Mapping Builder.

c. From the Driver drop-down list, select an available JDBC driver that is registered with iIT Transformer.

   Note: iIT Transformer does not ship with any of the predefined drivers.

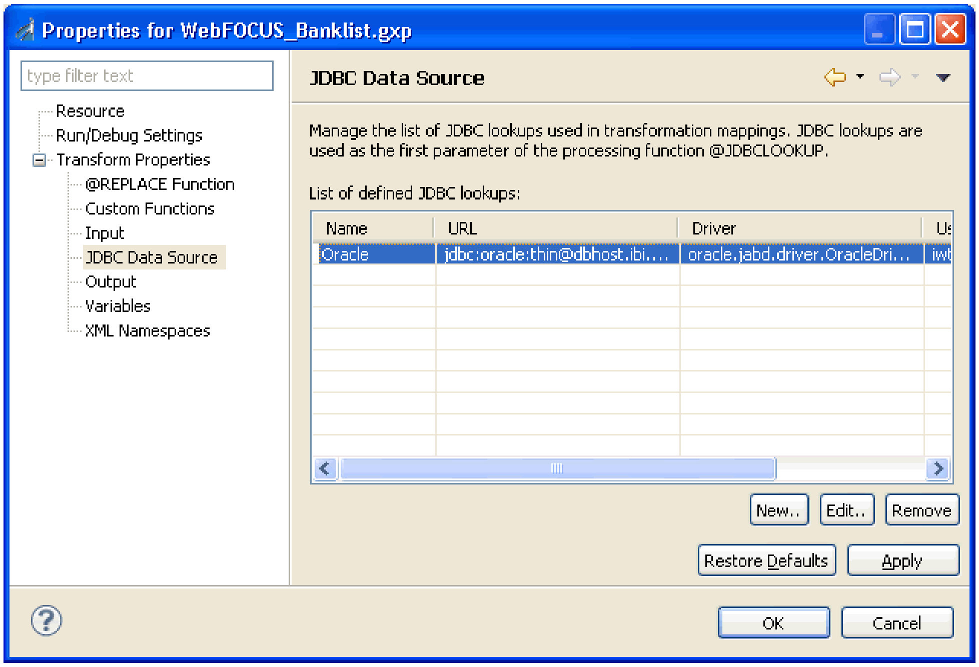
d. Type a valid user name and password to connect to the database.

e. Specify a timeout value in milliseconds.

   Note: If the timeout value is omitted, or the value is 0 (default), then timeout is not enabled.

4. Click *OK*.

   The JDBC lookup is added to the list of defined JDBC lookups.



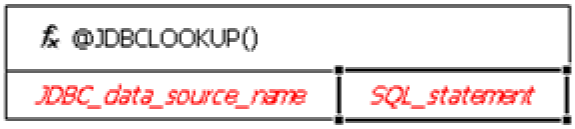
Manage the list of JDBC lookups used in transformation mappings. JDBC lookups are used as the first parameter of the processing function @JDBCLOOKUP.

List of defined JDBC lookups:

Name	URL	Driver	Us
Oracle	jdbc:oracle:thin@dbhost.ibi....	oracle.jabd.driver.OracleDri...	iwt

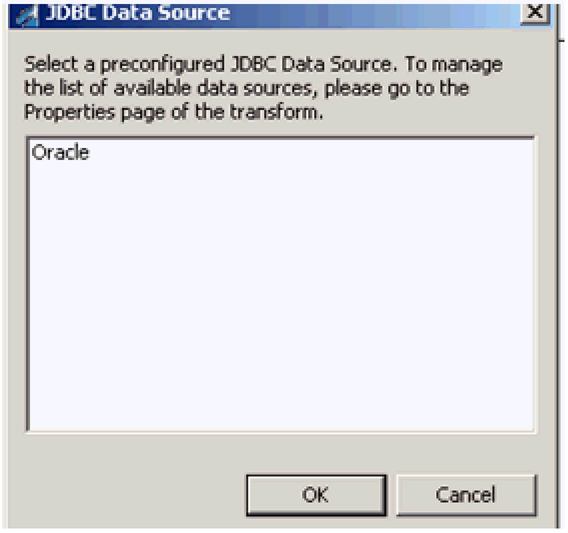
5. Click *OK*.
6. In the Mapping Values pane of the Mappings tab, select an output node.
7. Click the ellipsis ... button.

   The Mapping Builder opens.
8. From the Category drop-down list in the Functions pane of the Mapping Builder, select the *Processing Functions* category.
9. Select the @JDBCLOOKUP function, and drag it onto the workspace area of the Mapping Builder.



10. Double-click the *JDBC_data_source_name* parameter.

The JDBC Data Source dialog box opens as shown in the following image.

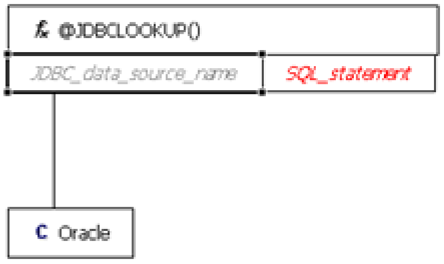


Select a preconfigured JDBC Data Source. To manage the list of available data sources, please go to the Properties page of the transform.

Oracle

11. Select the name of a defined JDBC lookup, for example Oracle, and click *OK.*

    The @JDBCLOOKUP function is updated as shown in the following image.



12. Double-click the *SQL_statement* parameter.

    Since the SQL statement is an expression, the *Where* clause can contain the following:

    - Constant
    - Transformer Function
    - Input Node Context
    - Unlimited AND & OR

    Note: In the Column Name field, you can also type * or Count (*) to return more advanced data sets.

13. Click *OK* once you have finished creating your SQL statement.

Configuring Properties for Replace

Replace functions are used to match and replace specific values of the input data. For example, suppose your input data has a node called "car_age" with a value of either "new" or "used". If you decide to substitute the term "used" and want to use "pre-owned", you can define a replace function with match and replace value pairing where the match value would be specified as "used" and the replace value is "pre-owned."

Replace functions work similar to custom functions in that you must first define the function itself and then, assign it in the output node mapping value which needs to be modified. This task can be accomplished with the @REPLACE and @SIMPLE_REPLACE functions.

The process of defining replace functions accounts for setting up the values to match and replace, but not the mapping. You must map the replace functions using the Mapping Builder. For more information, see *Using the Mapping Builder* on page 269.

Procedure: How to Define A Replace Function

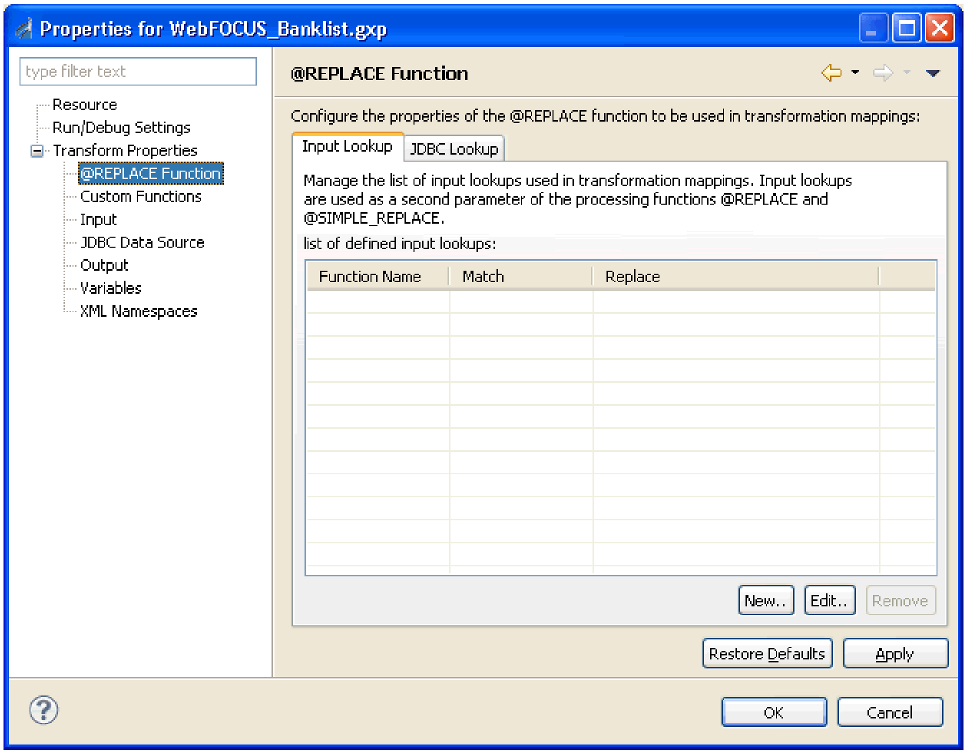
To define a replace function:

1. Right-click a Transform component and select *Properties* from the context menu.

   The Properties dialog box opens.

2. Expand *Transform Properties* in the left pane and then select the *@REPLACE Function* category.

The following image shows the Project Properties dialog box with the @REPLACE Functions category selected in the left pane.



Configure the properties of the @REPLACE function to be used in transformation mappings:

Manage the list of input lookups used in transformation mappings. Input lookups are used as a second parameter of the processing functions @REPLACE and @SIMPLE_REPLACE.

list of defined input lookups:

The *Input Lookup* and *JDBC Lookup* tabs contain the lists for the two types of replace functions available in iIT Transformer.

3. To define the replace functions in which you define the permanent or constant match and replace values, click the *Input Lookup* tab. For more information, see *How to Define an Input Lookup* on page 293.

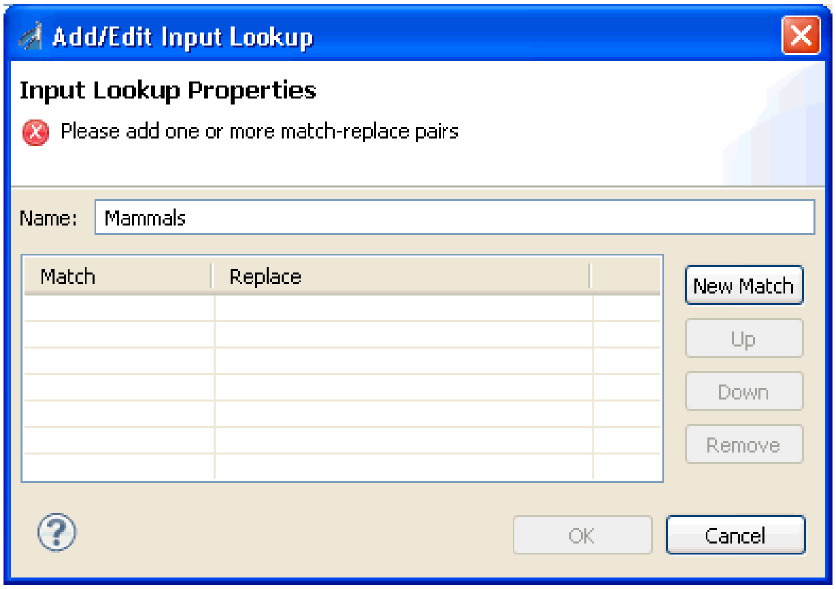
   To define replace functions in which the match and replace values are taken from the columns of the specified database table, enabling you to insert the input values from the different databases, use the *JDBC Lookup* tab.

Procedure: How to Define an Input Lookup

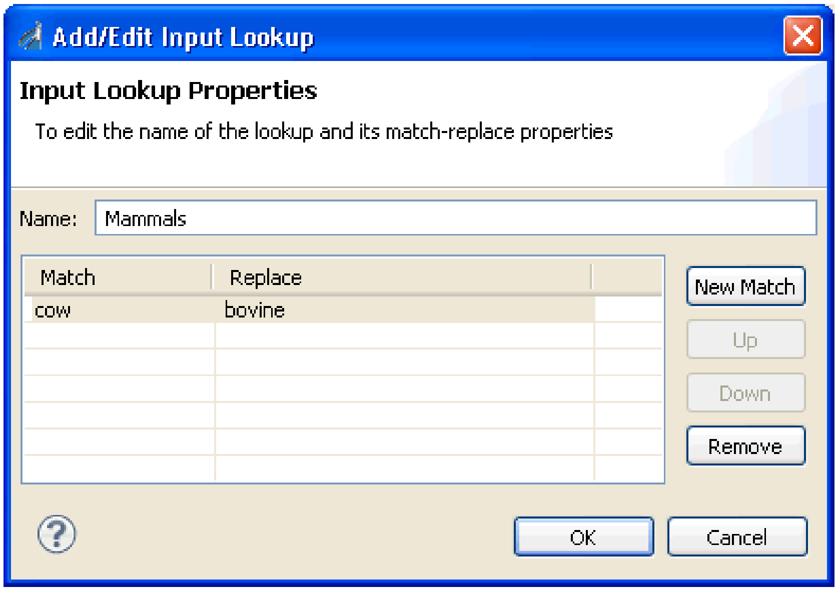
To define an input lookup for a replace function:

1. Click *New* in the Input Lookup tab.

The Add/Edit Input Lookup dialog box opens.



2. Type a name, for example, Mammals, for the new input lookup group.
3. Click *New Match*.



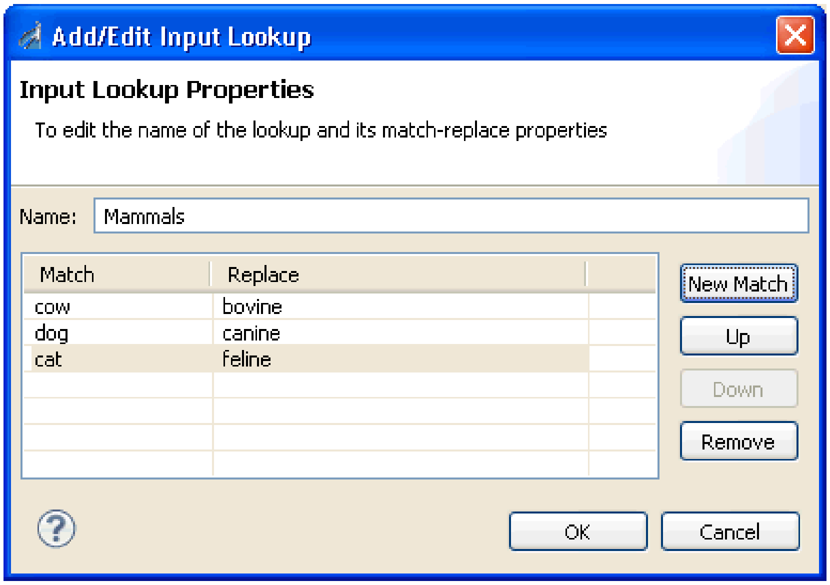
4. Specify the match and replace values in the corresponding fields accordingly.

5. Click *OK*.

   You are returned to the previous Add New Input Lookup dialog box.

   You can add multiple input lookups if required.

   For example, the following image shows a replace function named "Mammals." The lower pane of the Add New Input Lookup dialog box has columns for Match and Replace node values for various mammals, for example, “cow” to "bovine.”



Add/Edit Input Lookup

Input Lookup Properties

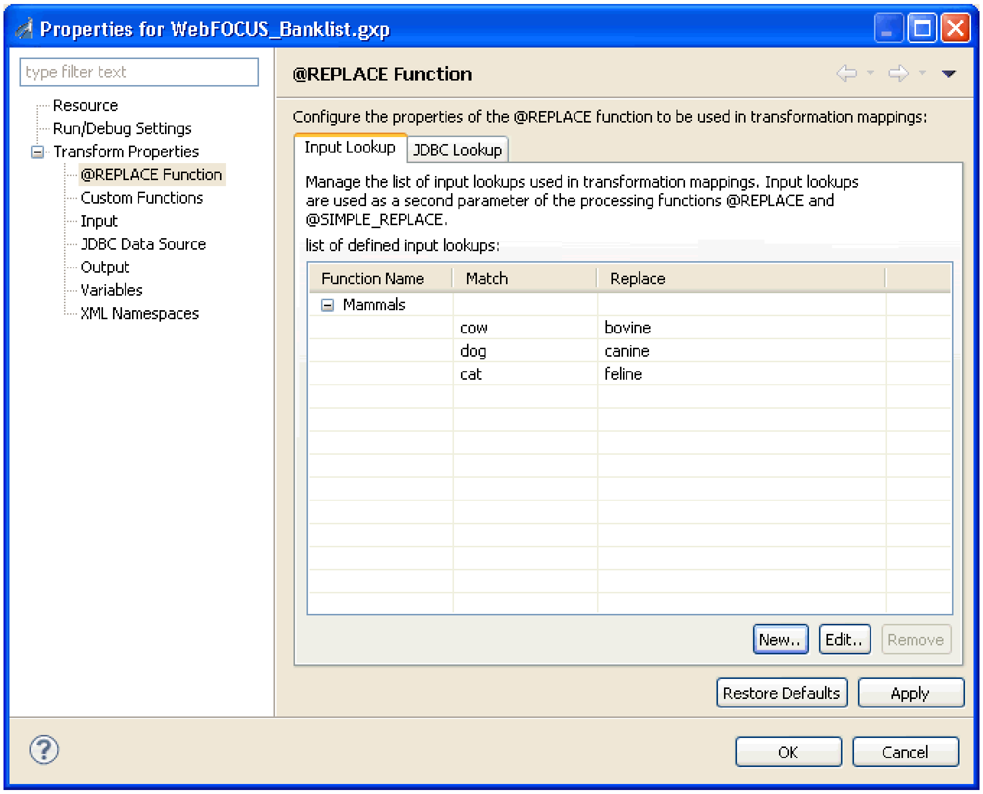
To edit the name of the lookup and its match-replace properties

Name: Mammals

Match	Replace
cow	bovine
dog	canine
cat	feline

Procedure: How to Modify an Input Lookup

To modify an input lookup:



Configure the properties of the @REPLACE function to be used in transformation mappings:

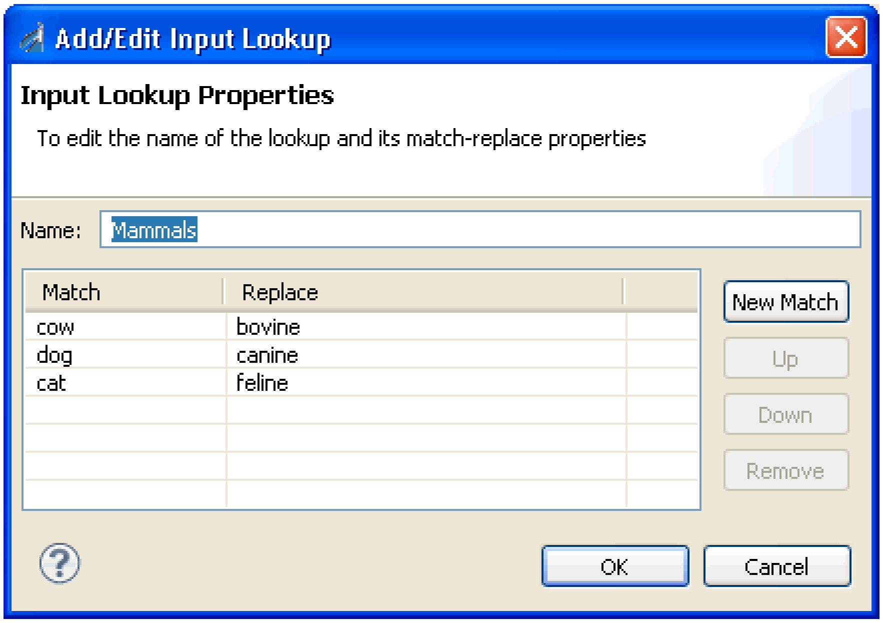
Manage the list of input lookups used in transformation mappings. Input lookups are used as a second parameter of the processing functions @REPLACE and @SIMPLE_REPLACE.

list of defined input lookups:

Function Name	Match	Replace
Mammals		
	cow	bovine
	dog	canine
	cat	feline

1. Select the input lookup, for example, Mammals, and click *Edit*.

The Input Lookup Properties dialog box opens.


Add/Edit Input Lookup

Input Lookup Properties

To edit the name of the lookup and its match-replace properties

Name: Mammals

Match	Replace
cow	bovine
dog	canine
cat	feline

You can perform either one of the following modifications to an input lookup group:

- Define additional match and replace pairings by clicking *New*.
- Delete an existing match and replace pairing by selecting it and clicking *Remove*.
- Modify the values of an existing match and replace pairing by selecting it and typing the new value.

2. Click *OK* once you have completed making the necessary modifications.

Procedure: How to Delete an Input Lookup

To delete an input lookup, select the input lookup, for example, Mammals, and click *Remove*.

The input lookup is removed from the list.

Custom Functions

Writing Custom Functions

The custom functions that can be used with iIT Transformer must be written in Java code, complying with the rules discussed in the following section. Custom functions can be stored as a class file or a jar file, either as part of your iIT project, or on the file system.

Note: The following section is written assuming that you are familiar with the Java programming language.

Each custom function you write must be a subclass of the AbstractFunction super class.

You must implement a constructor for your class and the two abstract methods, execute() and getReturnType(), which are inherited from the AbstractFunction class. You may also choose to add your own methods. The Java code for your custom function must conform to the following format:

```
import com.iwaysoftware.transform.common.function.AbstractFunction;
public class MY_FUNCTION_NAME extends AbstractFunction
{
     public MY_FUNCTION_NAME()
     {
        setName("MY_FUNCTION_NAME");
        setDescription("This is what my function does...");
     }
     public Object execute() throws Exception
     {
        // perform function's execution here
        // and return desired output value
     }
     public Class getReturnType()
     {
        return MY_FUNCTION'S_RETURN_TYPE;
     }
}
```

where:

MY_FUNCTION_NAME

Is the name of your custom function.

MY_FUNCTION'S_RETURN_TYPE

Is the data type of your custom function's return value.

Import Statement

You must include the following import statement in the beginning of your custom function implementation:

```
import com.iwaysoftware.transform.common.function.AbstractFunction;
```

This enables you access to the methods required for your custom function implementation.

Note: Optionally, you can utilize the logging capabilities of iIT Transformer in the code of your custom function by including the following import statements in the beginning of your custom function implementation:

```
import com.iwaysoftware.transform.common.log.LogUtil;
import com.iwaysoftware.transform.common.log.TransformLogger;
```

Class Declaration

As stated previously, a custom function is a subclass of the AbstractFunction super class. Therefore, you can declare your custom function as follows:

```
public class MY_FUNCTION_NAME extends AbstractFunction {
...
}
```

where:

MY_FUNCTION_NAME

Is the name of your custom function.

Constructor

There are two methods you must call within the constructor of your custom function. They are the setName(*String name*) and setDescription(*String description*) methods, which are predefined in the AbstractFunction super class. The values provided as parameters for these methods are utilized when you access your custom function within iIT Transformer.

The name of the function used in iIT Transformer is the parameter that you set using the setName method, and the description of the function used by iIT Transformer is the message you type as the parameter in setDescription.

execute()

The method execute(), is one of two abstract methods declared in the AbstractFunction super class, which is supposed to contain the function procedures and instructions. Your custom function, which is a subclass of AbstractFunction, must therefore, implement the execute() method as follows:

```
public Object execute() throws Exception {
...
}
```

When you are writing the body of this method, you can follow the same style that is used to write any method in Java.

Getting Arguments

Retrieving arguments passed into your custom function is achieved through the methods getValueInstance(), getArgument(*int index*), and getFunctionContext(), which are the predefined methods from the functions package that your custom function is a part of. To store two arguments as String variables *argOne* and *argTwo*, write the following:

```
String argOne =
(String)getFunctionContext().getArgument(0).getValueInstance();

String argTwo =
(String)getFunctionContext().getArgument(1).getValueInstance();
```

Other methods that you may find useful for your custom function implementation are:

- getNumberOfArguments()

  Returns an integer indicating the number of arguments passed to your function by the user, which is used in conjunction with method, getFunctionContext(), as follows:

  ```
  int num = getFunctionContext().getNumberOfArguments();
  ```

- getArguments()

  Returns an array containing all of the arguments passed to your function by the user. To store your arguments in an array of objects called myArray, use the following format:

  ```
  Object[] myArray = getFunctionContext().getArguments();
  ```

getReturnType()

getReturnType() is one of the abstract methods declared in the AbstractFunction class. Therefore your custom function, which is a subclass of AbstractFunction, must implement this method. The value returned by getReturnType() indicates the type of value you return in method execute(). The choices for your return value are:

- java.lang.Boolean.class
- java.lang.Character.class
- java.lang.Double.class
- java.lang.Float.class
- java.lang.Integer.class
- java.lang.Long.class

- java.lang.String.class
- java.lang.StringBuffer.class

Associated Object Map

The custom function interface of iIT Transformer is now updated with the ability to utilize the data structure of customizable global parameters. This data structure is implemented as a Java Map object, which is an object that maps keys to values. This object can be accessed by any custom function within your project. As a result, it can contain global reusable data or information that can be cached, such as your database connection string. The Associated Object Map can also be used to track the usage of the custom functions in your project, if required. Use the following format to access the Associated Object Map:

```
Map objectMap = super.getAssociatedObjectMap();
```

Sample Custom Function

The following snippet of Java code is the implementation for a custom function that returns a string resulting from adding 25 to the input value of the function.

```
import java.math.BigDecimal;
import java.util.Map;
import com.iwaysoftware.transform.common.function.AbstractFunction;
import com.iwaysoftware.transform.common.log.LogUtil;
import com.iwaysoftware.transform.common.log.TransformLogger;
public class ADD25
            extends AbstractFunction
{
      private static final BigDecimal BD25 = new BigDecimal(25);
      public ADD25()
      {
           /* specifies the function name to be used within iAM */
           setName("ADD25");
           /* specifies the function description to be used within iAM */
           setDescription("Return a string resulting from adding 25 to input value.");
      }
      public Object execute()
                  throws Exception
      {
           /*
            * retrieves the first argument and assigns it to Double variable
            * 'input'
            */
           // arguments are String always
           String inputArg = (String)
             getFunctionContext().getArgument(0).getValueInstance();
           BigDecimal input;
           try
           {
               input = new BigDecimal(inputArg);
           }
           catch (Exception e)
           {
               input = new BigDecimal(0);
           }
           // Gets the instance of the logger
           TransformLogger logger = super.getLogger();
           // Gets the debug flag
           boolean debug = LogUtil.canDebug(logger);
           // Fetches associate objects
           Map objectMap = super.getAssociatedObjectMap();
            /* creates a variable to hold the value that will get returned */
           String answerString;
           if (objectMap.containsKey(input))
           {

                          // return stored result
                          String result = (String) objectMap.get(input);
                          if (debug)
                          {
                                   LogUtil.debug(false, logger, "returning CACHED
           result: ");
                          }
                          return result;
                      }
                      else
                      {
                                  answerString = input.add(BD25).toString();
                          objectMap.put(input, answerString);
                          if (debug)
                          {

                         LogUtil.debug(false, logger, "returning the value : "
                     + answerString);
               }
               // return string
               return answerString;
           }
    }
    public Class getReturnType()
    {
           /* specifies that method execute() returns a String */
           return java.lang.String.class;
    }
```

Defining Custom Functions

If no predefined iIT Transformer function or their combination is sufficient to perform the task you require, you can write your own custom function containing the implementations of your task and returning the desired value. The custom functions must be written in Java code and compiled to produce a .class file, which then must be stored in the following directory to be available for use in design time with iIT Transformer:

```
<iWaySMHome>/tools/transformer/custom_functions
```

where:

<iWaySMHome>

Is the directory where iWay Service Manager was installed.

Before using a custom function within iIT Transformer, test it to ensure it works properly. For more information on creating or writing custom functions, see *Writing Custom Functions* on page 297.

Defining a custom function in iIT Transformer creates a link to the actual custom function .class file stored in the *custom_functions* directory. Custom functions are defined using the Project Properties dialog box.

You can also import the custom functions from the other Transform components. For more information, see *How to Import a Custom Function in iIT Transformer* on page 309.

Procedure: How to Define a Custom Function in iIT Transformer

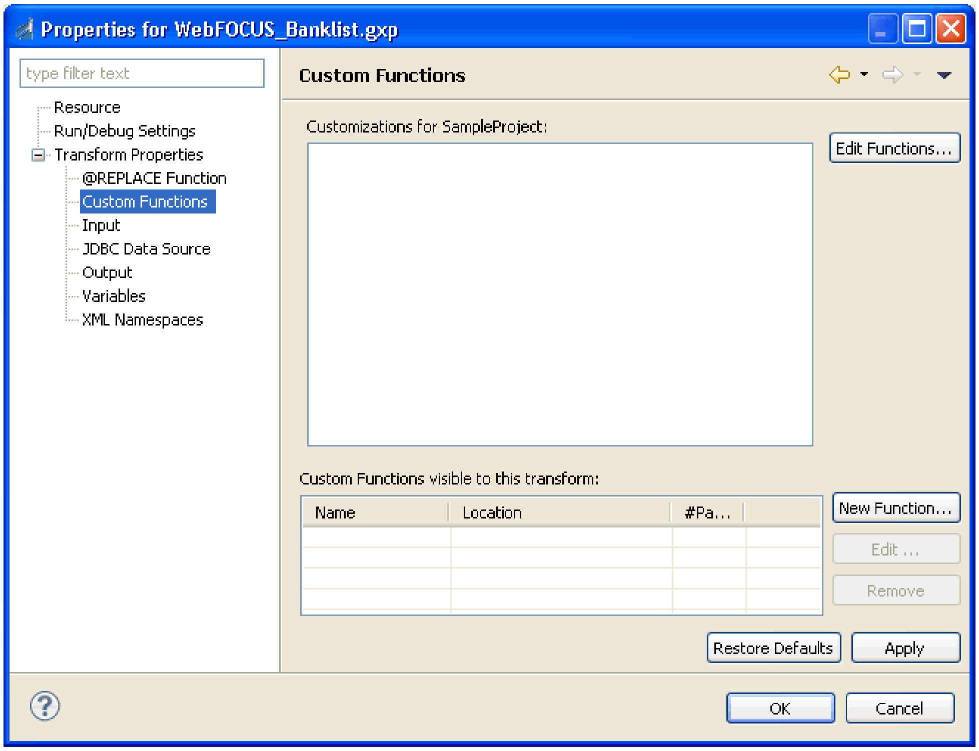
To define a custom function:

1. Right-click a Transform component and select *Properties* from the context menu.

   The Properties dialog box opens.

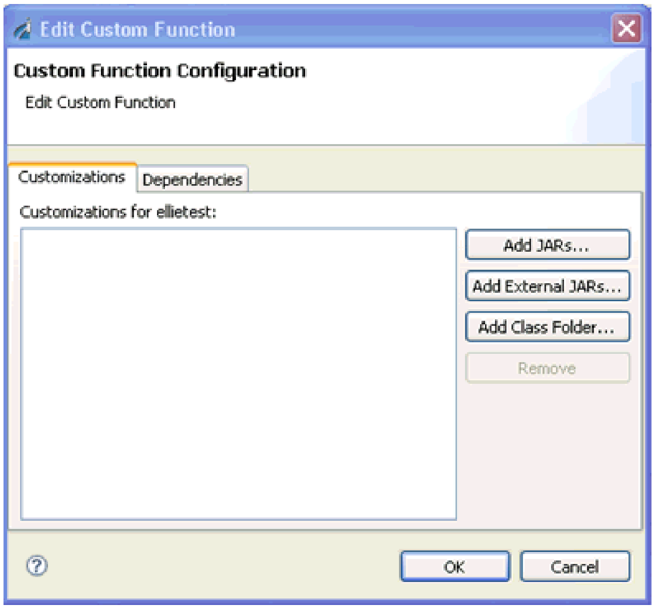
2. In the left pane, expand *Transform Properties* and select *Custom Functions*.

The following image shows the Properties dialog box with the Custom Functions category selected in the left pane.



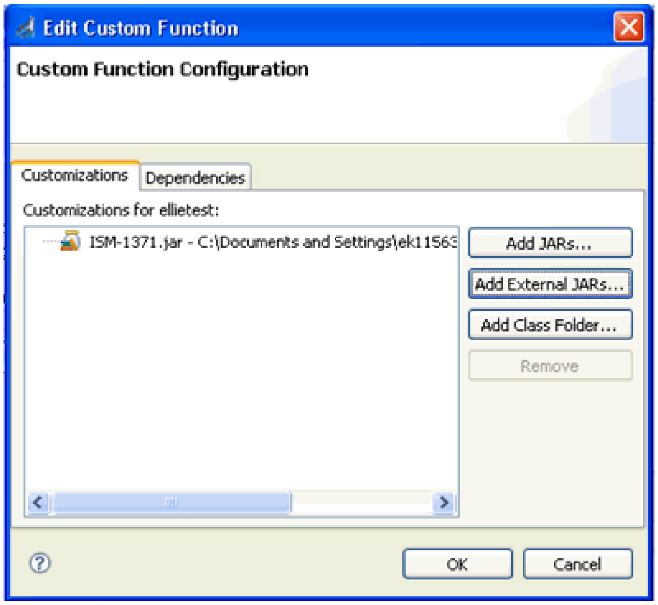
3. Click *Edit Functions*.

The Edit Custom Function dialog box opens.

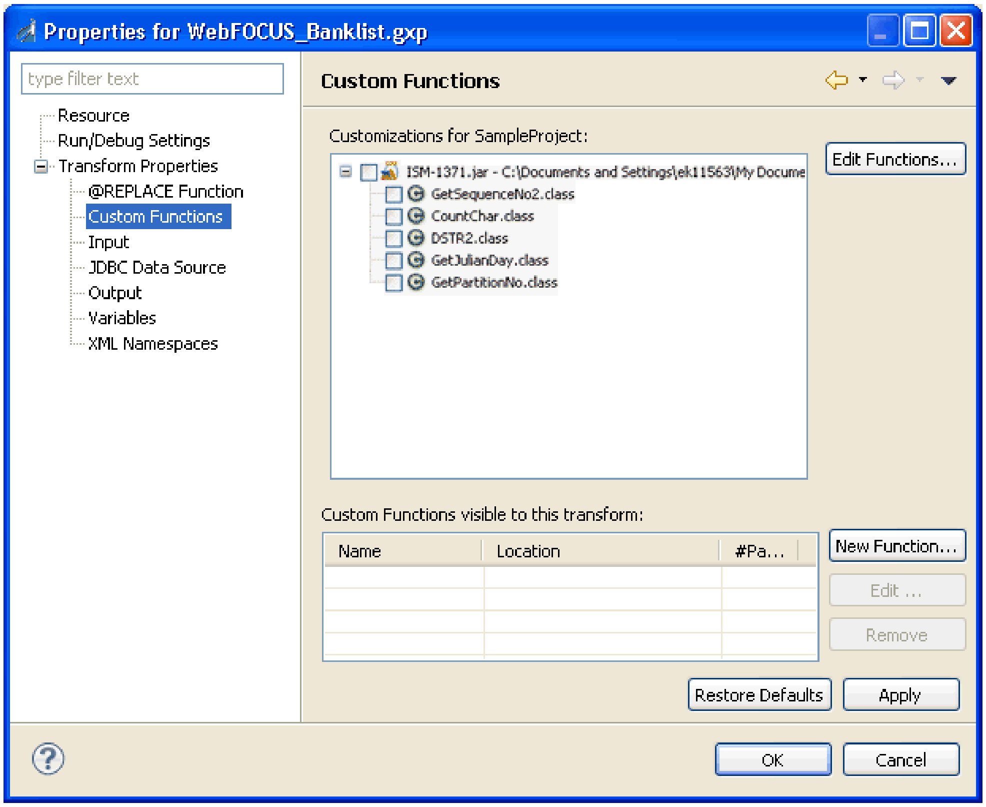


Several options are provided to add one or multiple custom functions if they reside in a .jar file.

4. Click *Add External JARs*, select the location of your .jar file, and then click *OK*.



Note that your .jar file has been parsed and the list of custom functions contained in the .jar file now appears in the Customizations area, as shown in the following image.



Customizations for SampleProject:

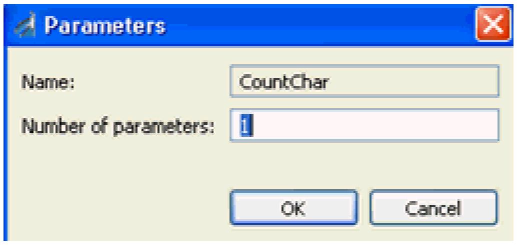
- ISM-1371.jar - C:\Documents and Settings\ek11563\My Docume
  - GetSequenceNo2.class
  - CountChar.class
  - DSTR2.class
  - GetJulianDay.class
  - GetPartitionNo.class

Custom Functions visible to this transform:

Name	Location	#Pa...

You can make any of the custom functions listed in the Customizations area available in Mapping Builder by clicking the check box next to it, for example, *CountChar.class*.

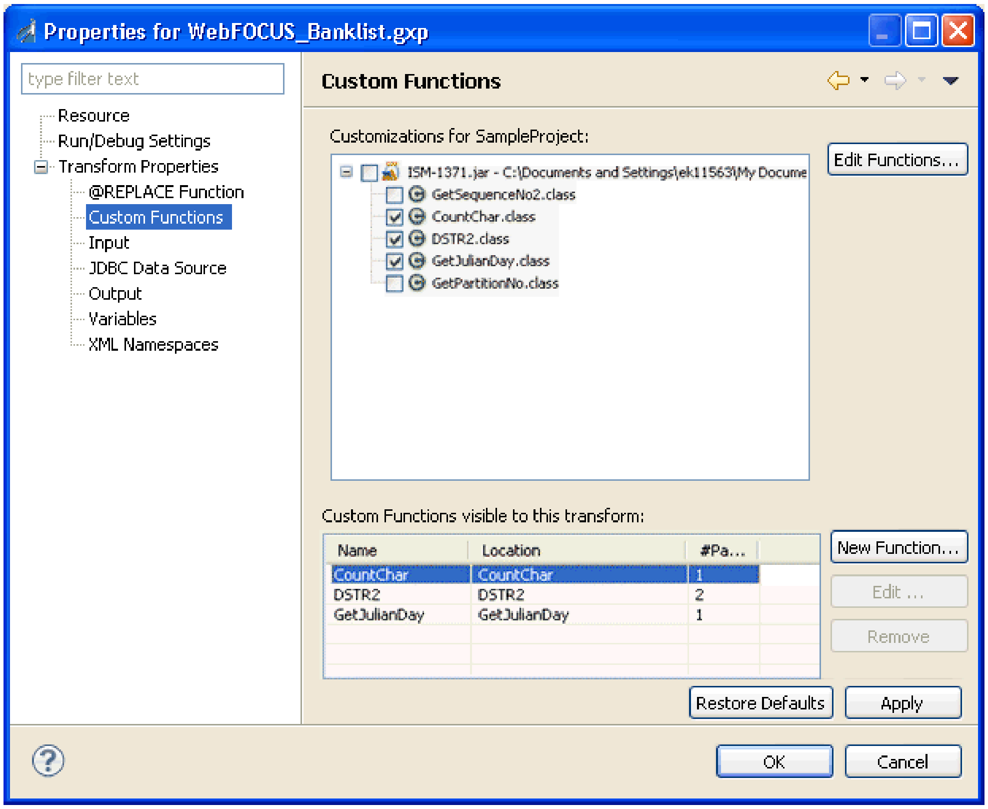
The Parameters dialog box opens, which prompts you to verify the name of the custom function. Enter the number of parameters that the custom function accepts as shown in the following image, and click *OK*.



Name: CountChar

Number of parameters: 1

Note that every custom function that was defined by selecting the corresponding check box in the Customizations area is now included in the Custom Functions visible to the transform area, as shown in the following image.



Customizations for SampleProject:

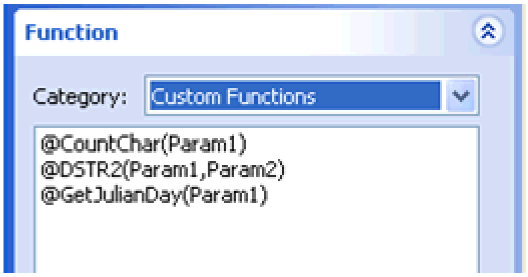
- [ ] ISM-1371.jar - C:\Documents and Settings\ek11563\My Docume
  - [ ] GetSequenceNo2.class
  - [✓] CountChar.class
  - [✓] DSTR2.class
  - [✓] GetJulianDay.class
  - [ ] GetPartitionNo.class

Custom Functions visible to this transform:

Name	Location	#Pa...
CountChar	CountChar	1
DSTR2	DSTR2	2
GetJulianDay	GetJulianDay	1

5. Click *OK*.

The custom functions now appear in the list of defined custom functions of the Mapping Builder. The name of the function is listed in the Function Signature column, and the location of the function is listed in the Location column.



@CountChar(Param1)
@DSTR2(Param1,Param2)
@GetJulianDay(Param1)

Procedure: How to Import a Custom Function in iIT Transformer

To import or define a custom function, or to define a JDBC Lookup:

1. Right-click a Transform component and select *Properties* from the context menu.

   The Properties dialog box opens.

2. In the left pane, expand *Transform Properties* and select *Custom Functions*.
3. Click *New Function*.

Compiling Your .java File

In order to make your custom function available for use within iIT Transformer, you need to compile the completed Java code of your custom function, integrating it with the Transformer engine code to create a .class file.

To compile custom functions, the `%IWAY_HOME%/lib` path must be part of your building environment. For example, you can use the following script to compile your custom function(s) with JDK version 1.6:

```
set IWAY_HOME=%IWAY7%
set LIB="%IWAY_HOME%lib/*"
set CLASS_FILES=%IWAY_HOME%etc/manager/transformations/custom_functions
set SOURCE_FILES=C:/temp/iway_custom_functions/src/*.java
javac -classpath %LIB%; -d %CLASS_FILES% %SOURCE_FILES%
@pause
```

where:

`C:/temp/iway_custom_functions/src/`

Is the temporary location for your custom function(s) source files on your hard drive.

Migrating Custom Functions

In early versions of Transformer, for example, *5.2.104*, the Java code for a custom function had a format similar to the following:

```
import com.xmlglobal.goxmltransform.engine.functions.*;
public class MY_FUNCTION_NAME extends AbstractFunction
{
  public MY_FUNCTION_NAME()
  {
    setName("MY_FUNCTION_NAME");
    setDescription("This is what my function does...");
  }
  public Object execute() throws Exception
  {
    // perform function's execution here
    // and return desired output value
  }
  public Class getReturnType()
  {
    return MY_FUNCTION'S_RETURN_TYPE;
  }
}
```

where:

`MY_FUNCTION_NAME`

Is the name of your custom function.

`MY_FUNCTION'S_RETURN_TYPE`

Is the data type of your custom function's return value.

The following procedure describes how to migrate custom functions created for the early versions of Transformer, for example, *5.2.104*, to be called by your modified .xch file.

Note: Before you compile the migrated custom functions, verify that the iwtranse.jar file is in your classpath.

Procedure: How to How to Migrate Custom Functions

To migrate custom functions, modify your import statements as follows:

1. Comment out xmlglobal package(s), for example:

   ```
   //import com.xmlglobal.goxmltransform.engine.functions.*;
   ```

2. Add the iwaysoftware package, which must be imported, for example:

   ```
   import com.iwaysoftware.transform.common.function.AbstractFunction;
   ```

Using Custom Functions at Run Time

In iIT Transformer, the functionality of custom functions for run time purposes is the same as the design time functionality. For more information on configuring custom functions, see *Custom Functions* on page 297.

Chapter 5

iWay Transformer Tips and Tricks

This section provides useful tips and tricks that are related to iWay Transformer.

In this chapter:

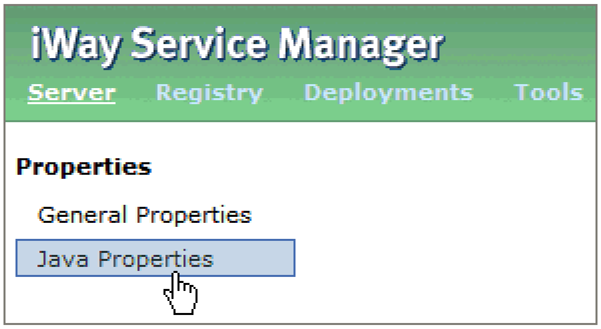
XSLT Runtime Processing

Since iWay Service Manager (iSM) does not include a native XSLT engine for iWay Transformer, iSM invokes external libraries for XSLT runtime processing.

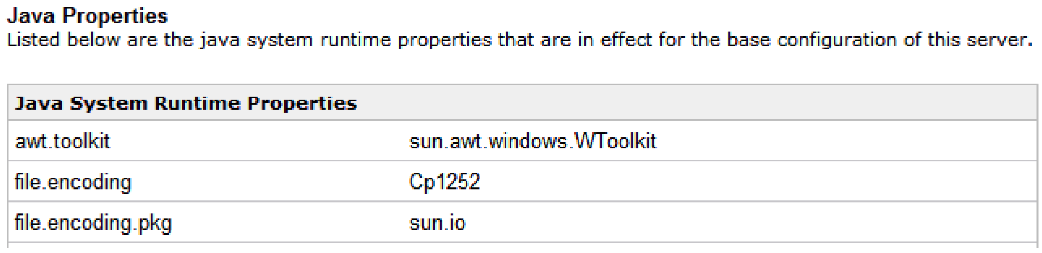
By default, iSM uses an XSLT engine for iWay Transformer that is compatible in Sun/Oracle and IBM Java Virtual Machines (JVMs). However, you can set any available XSLT engine for runtime processing to be used by iWay Transformer. You must ensure that the Java runtime property is assigned as described and the implementation is added to the classpath.

Note: The default XSLT engine that is set by iSM points to a processing library that supports XSLT Version 1.0. To enable support for XSLT Version 2.0, you must point to the processing library that is identified in this topic (org.apache.xalan.processor.TransformerFactoryImpl).

To view the current library that is used by iSM for XSLT runtime processing, logon to the iSM Administration Console, and click *Java Properties* in the left pane, as shown in the following image.



The Java Properties page opens, as shown in the following image.

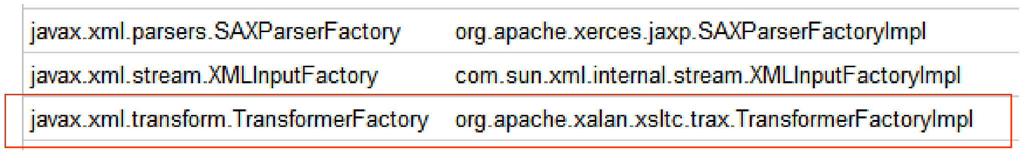


Java Properties

Listed below are the java system runtime properties that are in effect for the base configuration of this server.

Java System Runtime Properties	
awt.toolkit	sun.awt.windows.WToolkit
file.encoding	Cp1252
file.encoding.pkg	sun.io

Scroll down the list until you see the *javax.xml.transform.TransformerFactory* property and the corresponding value that is currently being referenced. In the following image, *javax.xml.transform.TransformerFactory* is set to *org.apache.xalan.xsltc.trax.TransformerFactoryImpl*.



javax.xml.parsers.SAXParserFactory	org.apache.xerces.jaxp.SAXParserFactoryImpl
javax.xml.stream.XMLInputFactory	com.sun.xml.internal.stream.XMLInputFactoryImpl
javax.xml.transform.TransformerFactory	org.apache.xalan.xsltc.trax.TransformerFactoryImpl

If no value is assigned to the *javax.xml.transform.TransformerFactory* runtime property, the default behavior of iSM is to check for the *com.ibm.xtq.xslt.jaxp.compiler.TransformerFactoryImpl* library, which is used in the IBM implementation of Java. if this XSLT engine is not found, iSM sets the runtime property to *org.apache.xalan.xsltc.trax.TransformerFactoryImpl* by default.

To point to a different library that will be used by iWay Transformer for XSLT runtime processing, you can use one of the following options:

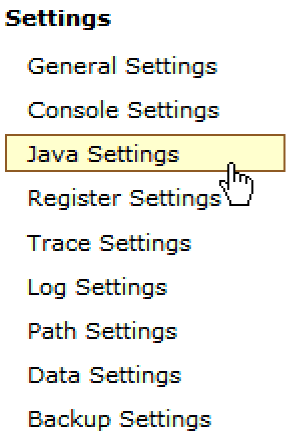
- ❑ If iSM is running as a service, then use the iSM Administration Console to add the setting.

  Note: This is the recommended option to use.

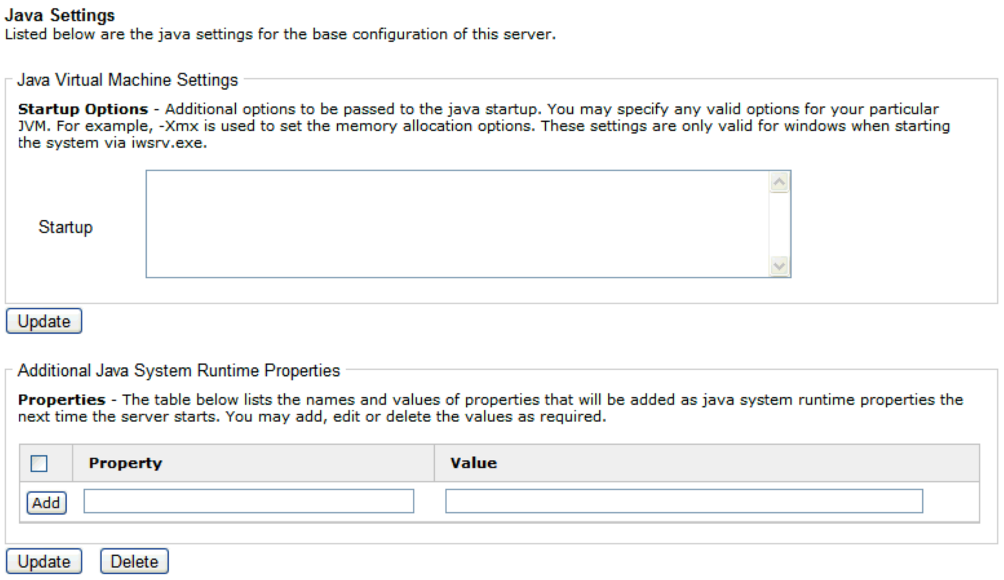
- ❑ If iSM is running as an application, then add the setting to the iway7.cmd build script.

Option 1: Running iWay Service Manager as a Service

In the iSM Administration Console, click *Java Settings* in the left pane, as shown in the following image.



The Java Settings page opens, as shown in the following image.



Java Settings
Listed below are the java settings for the base configuration of this server.

Java Virtual Machine Settings

Startup Options - Additional options to be passed to the java startup. You may specify any valid options for your particular JVM. For example, -Xmx is used to set the memory allocation options. These settings are only valid for windows when starting the system via iwsrv.exe.

Startup

Additional Java System Runtime Properties

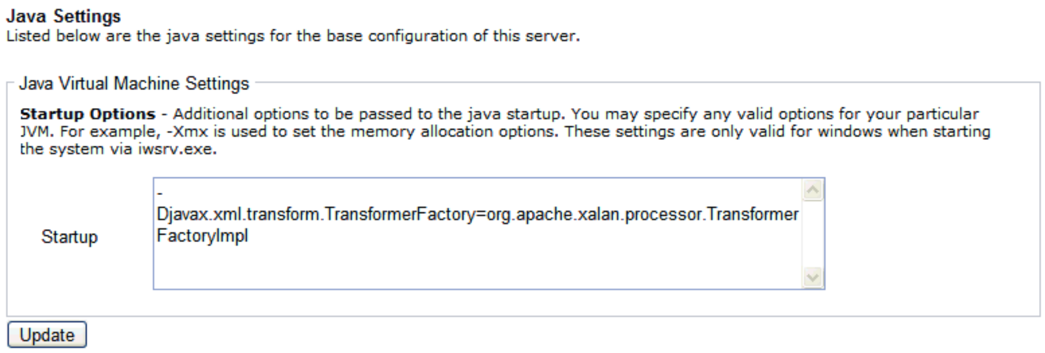
Properties - The table below lists the names and values of properties that will be added as java system runtime properties the next time the server starts. You may add, edit or delete the values as required.

☐	**Property**	**Value**
Add		

In the *Java Virtual Machine Settings* section, specify the following in the Startup field:

```
-
Djavax.xml.transform.TransformerFactory=org.apache.xalan.processor.Transform
erFactoryImpl
```

For example:


Java Settings
Listed below are the java settings for the base configuration of this server.

Java Virtual Machine Settings

Startup Options - Additional options to be passed to the java startup. You may specify any valid options for your particular JVM. For example, -Xmx is used to set the memory allocation options. These settings are only valid for windows when starting the system via iwsrv.exe.

Startup: -Djavax.xml.transform.TransformerFactory=org.apache.xalan.processor.TransformerFactoryImpl

In this example, *javax.xml.transform.TransformerFactory* is now being set to *org.apache.xalan.processor.TransformerFactoryImpl*.

Click *Update* when you are finished. As a best practice, restart iSM as well.

Option 2: Running iWay Service Manager as an Application

Edit the iway70.cmd build script and add the following command to the Java settings:

```
-
Djavax.xml.transform.TransformerFactory=org.apache.xalan.processor.Transform
erFactoryImpl
```

Save the iway70.cmd build script and restart iSM when you are finished.

Feedback

Customer success is our top priority. Connect with us today!

Information Builders Technical Content Management team is comprised of many talented individuals who work together to design and deliver quality technical documentation products. Your feedback supports our ongoing efforts!

You can also preview new innovations to get an early look at new content products and services. Your participation helps us create great experiences for every customer.

To send us feedback or make a connection, contact Sarah Buccellato, Technical Editor, Technical Content Management at *Sarah_Buccellato@ibi.com.*

To request permission to repurpose copyrighted material, please contact Frances Gambino, Vice President, Technical Content Management at *Frances_Gambino@ibi.com.*

Information Builders

iWay

iWay Integration Tools Transformer User's Guide

Version 7.0.x and Higher

DN3502273.0418

Information Builders, Inc.
Two Penn Plaza
New York, NY 10121-2898