Financial Message Designer for AquaLogic Service Bus User's Guide

Version 3.4

Financial Message Designer for AquaLogic Service Bus

Financial Message Designer for AquaLogic Service Bus is an integrated environment used for creating and managing a cartridge, which is a model that captures message formats, message mappings and message flows. Once a cartridge design is completed, it can be generated into platform specific code and deployed into Runtime. Once deployed, the Runtime becomes capable of performing the transformations defined by that cartridge.

See Also:

[Designer User Interface](#)
[Cartridge](#)
[Message](#)
[Message Mapping](#)
[Formula](#)
[Function Definition](#)
[Resources](#)
[Code Generation](#)
[Simulator](#)
[Debugging](#)
[Working With Cartridge Designer](#)

Designer User Interface

The Designer user interface consists of a set of windows, menus and tools for developing and managing cartridges. It can be used to create, open, update, build and deploy cartridges.

The Designer main window is shown below.
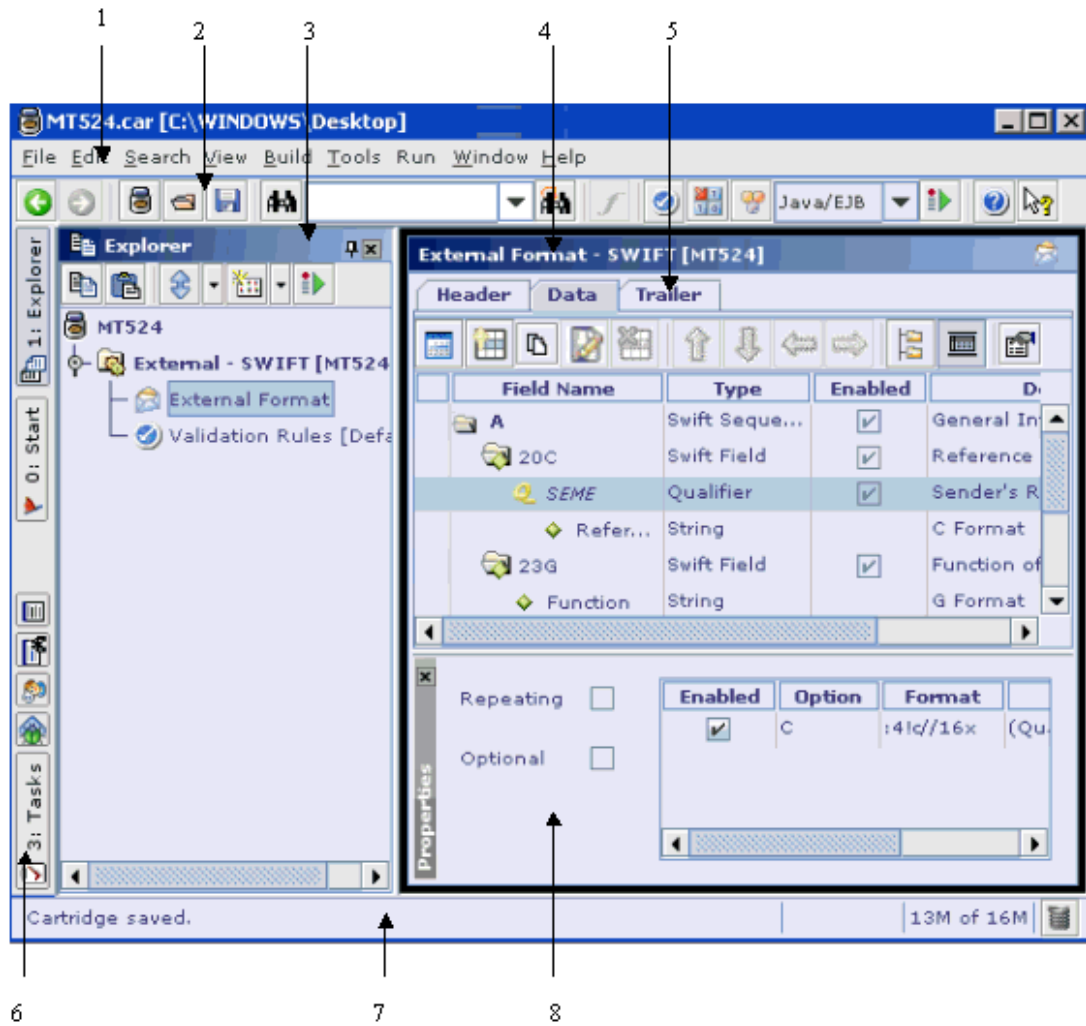
1. [Menu Bar](#)
2. [Tool Bar](#)
3. [Explorer](#)
4. [Element Heading](#)
5. [Design Element UI Pane](#)
6. [Speed Bar](#)
7. [Status Bar](#)
8. [Properties](#)

Menu Bar

The Menu bar in **Designer** is shown below.

See Also:

[File Menu](#)
[Edit Menu](#)
[Search Menu](#)
[View Menu](#)
[Build Menu](#)
[Tools Menu](#)
[Run Menu](#)
[Window Menu](#)
[Help Menu](#)
[Tool Bar](#)

File Menu

[New Cartridge](#) (Ctrl+N)	Creates a new cartridge.
[Open Cartridge](#) (Ctrl+O)	Opens an existing cartridge in a new Designer window. This allows opening multiple cartridges in a Designer session.
[Save](#) (Ctrl+S)	Saves the current cartridge.
Save As	Creates a copy of the current cartridge or saves the newly created cartridge.
[Close](#)	Closes the currently open cartridge without closing Designer.
[Settings](#)	Allows the user to set the option to reopen last file on startup and to switch between the different color schemes viz., Blue, Green, Gray, Kunststoff and Windows Look & Feel and to choose the required font & size.
[Recent](#)	Displays a list of the most recently used cartridges.
[Exit](#)	Exits Designer.

See Also:

[Menu Bar](#)
[Tool Bar](#)

Edit Menu

Copy (Ctrl+C)   Copies the current design element definition into clipboard as XML.

Paste (Ctrl+V)   Pastes the copied design element definition to the current design element.

Import (Ctrl+Shift+I)   Imports the definition of a design element from the selected file.

Export (Ctrl+Shift+E)   Exports the definition of a design element into a file.

Back (Ctrl+Alt+Left)   Takes you to the last design element you have visited before selecting the current design element.

Next (Ctrl+Alt+Right)   Takes you to the design element that you have visited before selecting the Back action.

Reformat Formula (Alt+F8)   Auto formats formulas in the current node/all nodes in the cartridge.

See Also:

Menu Bar
Tool Bar

Search Menu

Find (Ctrl+F)   Displays the **Find** dialog box, which is used to search for an item in the currently selected design element based on the specified search criteria.

Find Next (F3)   Resumes the recent search operation.

Find in Path   Displays the **Find in Path** dialog box, which is used to search for design elements that contain the search text in one of the search items specified in that dialog box.

Incremental Search (Ctrl+I)   Displays the **Search for** pop up to quickly search occurrence of text in the current location

View Menu

[Status Bar](#) (Alt+4)   Hides or displays the status bar.

[Tool Bar](#) (Alt+5)   Hides or displays the main window Tool Bar.

[Element Heading](#)   Hides or displays the Element heading bar.

[Start](#) (Alt+0)   Hides or displays the **Start** window

[Explorer](#) (Alt+1)   Hides or displays the **Explorer** window.

[File Explorer](#)   Hides or displays the **File Explorer** window

[Message](#) (Alt+2)   Hides or displays the **Message** window.

[Tasks](#) (Alt+3)   Hides or displays the **Tasks** window.

[Search Results](#) (Alt+6)   Hides or displays the **Search Results** window.

[Change Log](#) (Alt+7)   Hides or displays the **Change Log** window

[Debug](#) (Alt+8)   Hides or displays the **Debug** window

Build Menu

[Validate](#) (Ctrl+L)   Invokes the validation operation for the current design element.

[Validate All](#) (Ctrl+T)   Invokes the validation operation for all design elements of the cartridge.

[Generate Cartridge](#) (F7)   Invokes cartridge generation in the default platform.

[Code Generation](#)   Displays the Code Generation Settings dialog

[Settings](#)	corresponding to the default platform.
[Select Default Platform](#) ...	Select a required platform (Java/EJB) and set it as default.
Java/EJB > [Generate Cartridge](#) (Java/EJB)	It first invokes the validation operation for all design elements of the current cartridge. It then generates the cartridge into Java/EJB components based on the current code generation settings.
Java/EJB > [Code Generation Settings](#) (Java/EJB)	Displays the **Java/EJB Code Generation Settings** dialog box that allows you to specify the information used in generating the Java/EJB components. It allows you to specify component names, bind the reference class names with the concrete Java classes and specify references from one component to other component(s).

See Also:

[Menu Bar](#)
[Tool Bar](#)

Tools Menu

[Diff Cartridges](#)	Used for comparing two cartridges
[Cartridge Publisher](#)	Generates HTML docs for the cartridge and its child design elements into the specified directory.
[Deployer](#)	Used to deploy/undeploy the generated components under the platform specific Runtime. The supported Runtime is EJB.
SQL Tools	Displays a submenu of database tools.
SQL Tools > Schema Generator	This can be used to generate db independent XML schema files as well as db specific SQL schema files for the tables in a data source.
SQL Tools > SQL Generator	This can be used to generate db specific SQL commands for the given XML schema file, which is a db independent way of specifying SQL commands.
SQL Tools > Execute SQL	This tool executes a set of SQL commands against the specified data source, based on the given schema file.

	This tool supports both XML schema files and SQL schema files.
SQL Tools > SQL Console	This tool can be used to execute SQL commands against the specified data source just like Oracle SQL*Plus.
XML Tools > DTD Generator	Generates a DTD file from the structure of an XML file.

See Also:

Menu Bar
Tool Bar

Run Menu

Simulator (F5)	Launches Simulator corresponding to the default platform
Toggle Breakpoint... (F9)	Adds a new Breakpoint or Deletes an existing Breakpoint
Step Over (F10)	Runs the current statement as a unit and steps to the next statement
Step Into (F11)	Runs the current statement, steps Into any sub functions in it and then advances to the next statement
Step Out (Shift+F11)	It advances past the remainder of the code to be executed in the current location and then advances to the statement immediately following the one that called the current location
View Debug Window (Alt+8)	Hides or Displays the **Debug** window

See Also:

Menu Bar
Tool Bar

Window Menu

| New Window (Ctrl+W) | Opens a new Designer window |

<Cartridge window name>	Activates the selected cartridge window (brings the cartridge to front).
<Simulator window name>	Activates the selected simulator window.

See Also:

[Menu Bar](#)
[Tool Bar](#)

New Window

Designer provides an easy way for opening new designer window and navigating to other open designer and simulator windows using **New Window** feature. When we select New Window menu item from Window menu bar, a new Designer window will open.
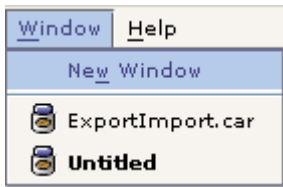
Selecting the cartridge name from the list will navigate to the corresponding cartridge ie, the particular Cartridge gets activated (brought to front). When you open Designer for the first time, 'Untitled' will appear as sub Menu in Window Menu, when the cartridge is saved, Untitled will be replaced by the cartridge name. When you double click a cartridge from Windows Explorer the cartridge opens in a new instance of Designer and Cartridge name will appear as sub Menu in Window Menu.
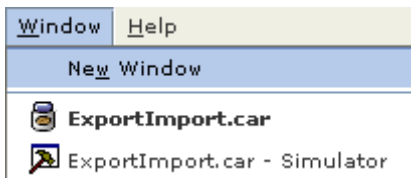
As you can see the window menu lists all cartridges and simulator windows that are currently open.

See Also:

[Menu Bar](#)
[Tool Bar](#)

Help Menu

Designer	Opens the help document for **Designer**
Java Runtime Help	Opens the help document for **Java Runtime**
[Version Info](#)	Displays Designer **version** and Plug-in versions. It also displays versions of Simulator and Code Generator.
About Designer	Displays the **version, build and copyright information** about Designer.

See Also:

[Menu Bar](#)
[Tool Bar](#)

Tool Bar

The tool bar contains the following icons for accessing frequently used operations.

	The Back button takes you to the last design element you have visited before selecting the current design element.
	The Next button takes you to the design element that you have visited before selecting the Back action.
	[Creates a new cartridge](#).
	[Opens an existing cartridge](#).
	Opens Repository client
	Saves the current cartridge.
	[Initializes the search operation](#).
amount ▼	Allows [searching](#) for an item in the currently selected design element based on the search text.
	[Resumes the recent search operation](#).
	Displays the [Edit Formula](#) (F4) dialog box that helps in specifying a formula by providing easy access to the formula functions and data fields.
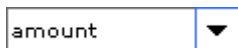
⊘	Invokes the [validation](#) operation for all design elements of the cartridge.
🔢	Invokes [cartridge generation](#) in the default platform.
🔧	Displays the [Code Generation Settings dialog](#) corresponding to the default platform.
Java/EJB ▼	[Allows selecting the default platform](#)
▶	Launches [Simulator](#) corresponding to the default platform.
❓	Displays the CHM for [Designer](#)
▸?	Shows the Help Tracker icon and opens the [Help](#) file for the selected UI
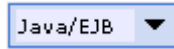
See Also:

[Menu Bar](#)

Explorer

The **Explorer** window displays the contents of a cartridge in the form of a tree structure. It displays the cartridge design element and all of its child elements such as internal/external message, persistence designer, mapping, message flow and so on. The tree structure coupled with expand/collapse functionality makes moving between various design elements intuitive.

The Explorer window is shown below.
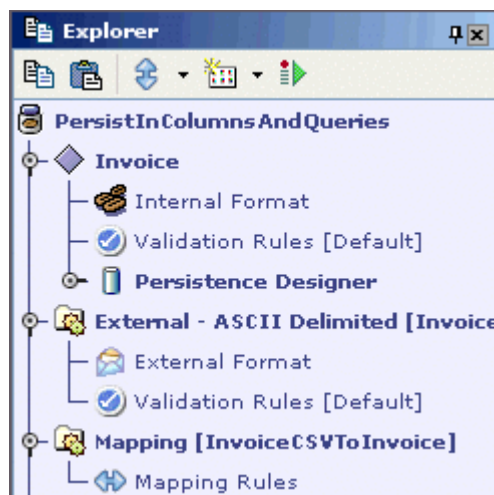
The user can show/hide the Explorer window either by selecting the Explorer menu item of the View menu or by clicking on the Explorer button on the Speed Bar.

When the Explorer window is open, it can be closed by clicking on the Close button present in the right corner of its title bar.

The 'Auto Hide' mode ('Pinned' mode) of the Explorer window can be enabled or disabled, by toggling the 'Auto Hide' button present in its title bar. When the 'Auto Hide' mode is enabled (indicated by the ⊨ icon), the Explorer window hides itself when it loses focus. When the 'Auto Hide' mode is disabled (indicated by the ⊤ icon), the Explorer window remains visible even when it loses focus.

Clicking on the Copy icon ▤ present in the Explorer window toolbar copies the definition of the selected element into clipboard. See the Copy/Paste a Design Element section for more information.

Clicking on the Paste icon ▤ present in the Explorer window toolbar imports the recently copied design element definition. See the Copy/Paste section for more information.

The **Move Node** ⬍ drop down menu pops up a menu as shown below and the items present in it can be used to rearrange the position of the design element nodes in the Explorer tree. See the Arranging Nodes section for more information.
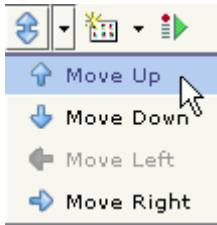
The **New Item** 📅 drop down menu pops up a separate menu and the items present in it can be used to create new folders, internal/external messages, mappings, message flows, Resources and function definitions.

Select 'Mount Directory' menu item from the 'New Item' drop down in Explorer toolbar, to mount one or more file system directories so that you can manage external cartridge resources such as sources files, XML, DTD, etc. from within Designer.

Clicking on the **Simulator** ▶ tool icon launches Simulator in default platform. The design element selected in Explorer at the time of launching Simulator is automatically selected for execution.

The icons present in **Explorer** tree convey the type of design element.

[Cartridge](#)

[Internal Message](#)

[Internal Format](#)

[Processing Rules](#)

[Persistence Designer](#)

[Database Table Design](#)

[Queries](#)

[Database Mapping](#)

[External Message](#)

[External Format](#)

[Validation Rules](#)

[Mapping Rules](#)

WebForms Designer

WebForm

Message Flow

[Reference](#)

[Folder](#)

See Also:

[Arranging Nodes](#)
[Delete Node](#)
[Designer User Interface](#)

Arranging Nodes

The **Move Node** drop down menu present in the Explorer toolbar pops up a separate menu as shown below and the items present in it can be used to rearrange the position of the design element nodes in the Explorer tree. This is especially useful in cartridges making use of Folder nodes to organize other design element nodes.
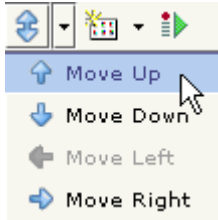
Once the required design element node is selected, the user can click on these items to rearrange the position of that node within the Explorer tree.

The following table describes the icons used to arrange the nodes of the Explorer tree.

Icon	Purpose	Description
	Move Up	This moves the selected node up by one position within the same Folder/cartridge node. This icon is disabled when the top of that Folder/cartridge node is reached.
	Move Down	This moves the selected node down by one position within the same Folder/cartridge node. This icon is disabled when the bottom of that Folder/cartridge node is reached.
	Move Left	This promotes the selected node by one level in the explorer tree hierarchy. This means that the selected node is promoted to the same level that of its previous parent node. The node is positioned at the bottom of the new parent node.
	Move Right	This demotes the selected node by one level in the explorer tree hierarchy. This icon is enabled only when the selected node is positioned immediately after a Folder node, which is at the same level. Clicking on this icon makes the selected node a child of that Folder node. The node is positioned at the bottom of that

		Folder node.

Delete Node

To remove any node from the cartridge right click the node and select 'Delete' menu item. 'Confirm Delete' dialog will be displayed.
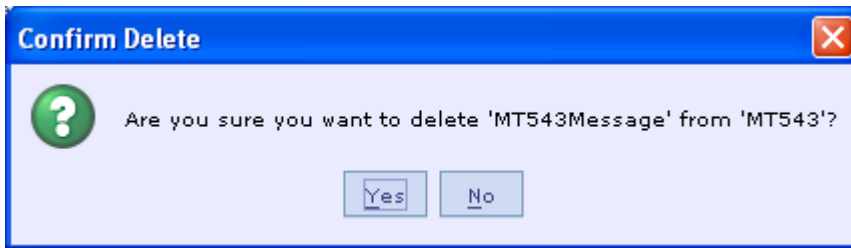
Click 'Yes' to delete the selected node.

File Explorer

The File Explorer displays files in the file system as a tree. Double clicking a file opens it using the associated application. Double clicking a cartridge opens it in Designer.

1. Select the 'View Menu'.

2. Select 'File Explorer' menu item. The 'File Explorer' is displayed.
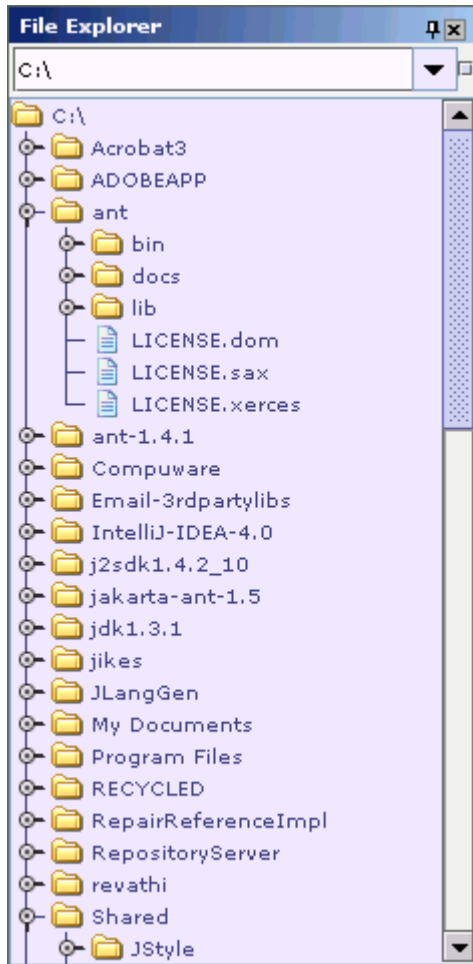
Open File

Select a file in the File Explorer. Right Click and select 'Open File' menu. The selected file is opened in windows explorer.

Explore as Root

Select a directory in the File Explorer. Right Click and select 'Explore as Root' menu. The selected directory is set as the root directory for the file explorer.

File Explorer Filters

The file explorer filter can be used to filter the files that are displayed in the explorer. Select the filter button in the explorer toolbar. The following dialog will be displayed.
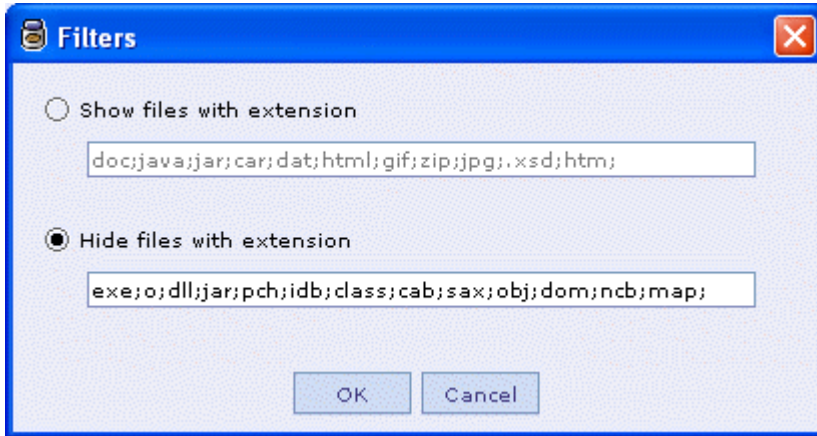
To hide files with a certain extension select 'Hide files with extension' option button and add the file extension (e.g. xml) to be ignored in the text box. Press ok. The files with the specified extension will not be displayed in the explorer.

To show only files with a certain extension select the 'Show files with extension' option button and add the extension in the text field. Only files with the specified extension will be displayed in the explorer.

If multiple extensions are specified they should be separated by ';' character.

See Also:

Designer User Interface

Getting Started Pane
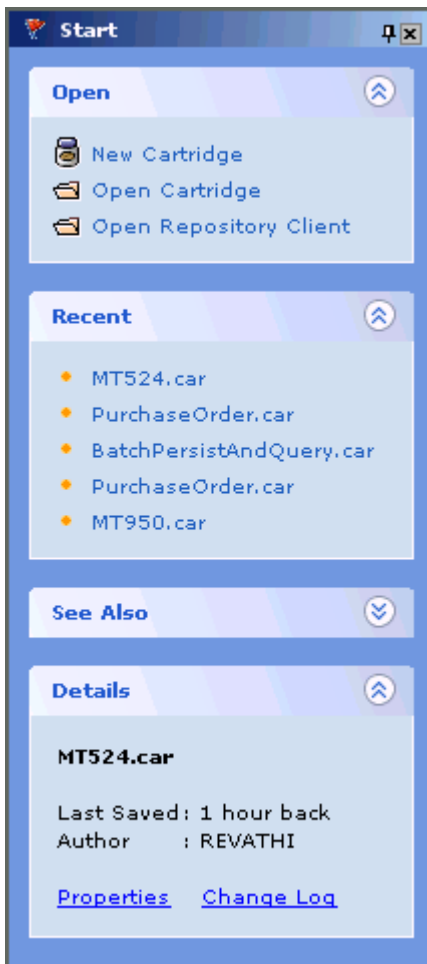
This pane provides convenient links to get started. The recently accessed cartridges names are displayed. Clicking the name opens the cartridge. Links for creating a new cartridge or opening a cartridge that is not in the recent file list are also provided in this pane. Link to Help Index is provided.

See Also:

Designer User Interface

Design Element UI Pane

When a design element node is selected in the Explorer pane, the user interface of the corresponding design element is displayed in this pane. It usually consists of a toolbar that can be used in defining/updating that design element. The Internal/External message design element UI contains 'Tree View' and 'Properties' buttons in tool bar.

See Also:

Designer User Interface
Tree View
Properties

Tree View
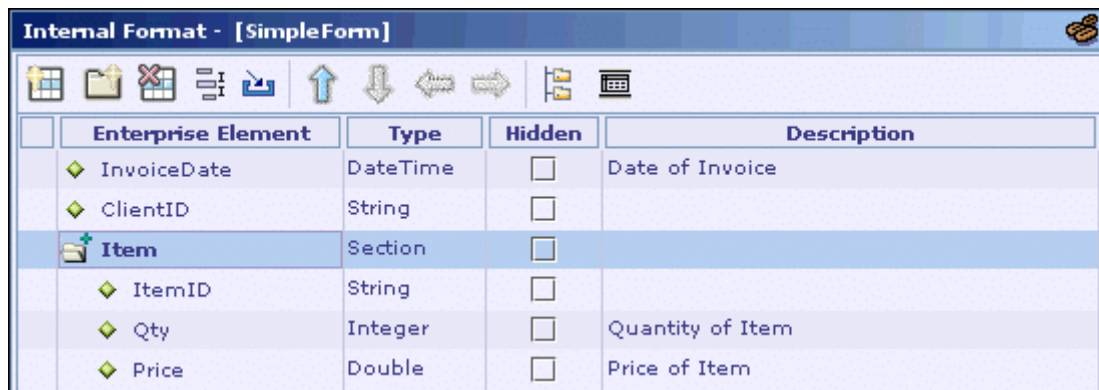
The Tree View helps user to view the table contents of a message in a tree structure. The button '⌨' when enabled, displays the tree structure for the fields/section in the table.

All operations like adding a field/section, deleting a field/section etc., related to a message can be performed in the Tree structure.

For an internal message the root node name in the Tree structure is the name of the internal message and child nodes are the 'fields' or 'sections'. For an external message, the tree view depends on the layer selected. The root node in a tree structure is 'Header' for a Header layer, 'Data' for a Data layer and 'Trailer' for a Trailer layer and child nodes are the fields' or 'sections'.

Consider the following Internal message with fields and a section with sub fields.

The 'Toggle Tree View' icon that is present in the tool bar when enabled '⌨', displays the corresponding tree structure for the field/section in table view.
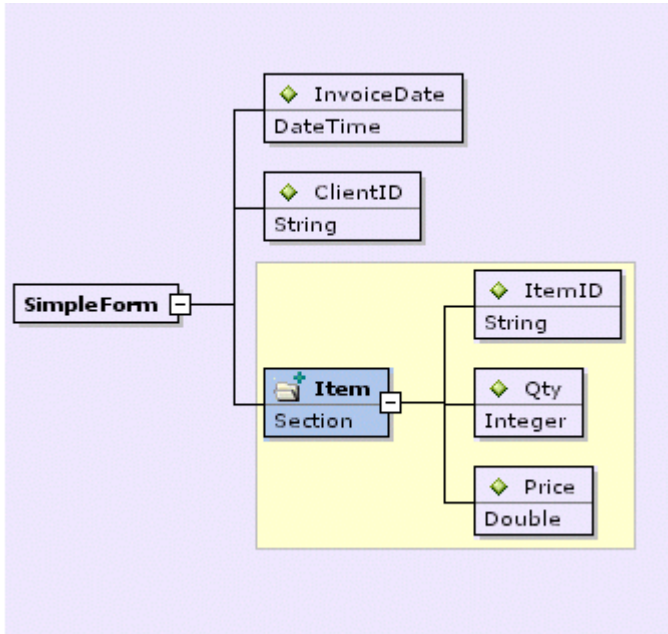
The field/section that was highlighted in the table structure appears as selected in the corresponding tree structure. In a tree structure, the name of the field/section or its data type is editable. Any section with sub fields is collapsible in the tree structure. The occurrence property of a field or a section in table view can also be viewed in the tree structure.
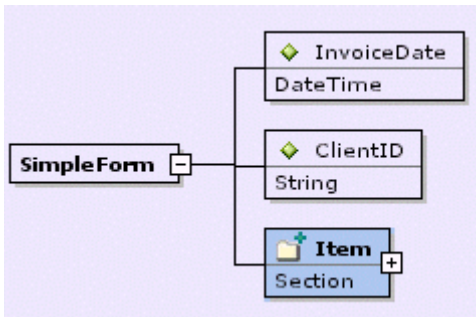
In the above diagram, the section 'Item' appears with the occurrence property 'Mandatory, Repeating'. Right clicking inside the tree structure displays a pop up with context menu items **Show Annotation** and **Show Type**.
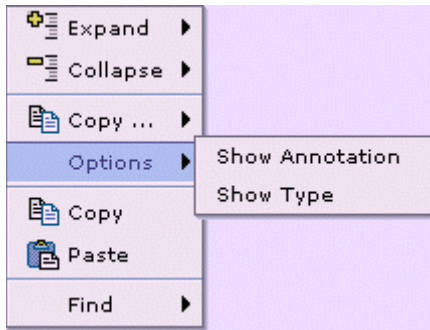
Show Annotation menu item when selected shows the description of all the fields/section and hides the description when selected again. The **Show Type** menu item is selected by default. The 'Show Type' menu item when toggled hides the data type of all the fields/section in the UI.
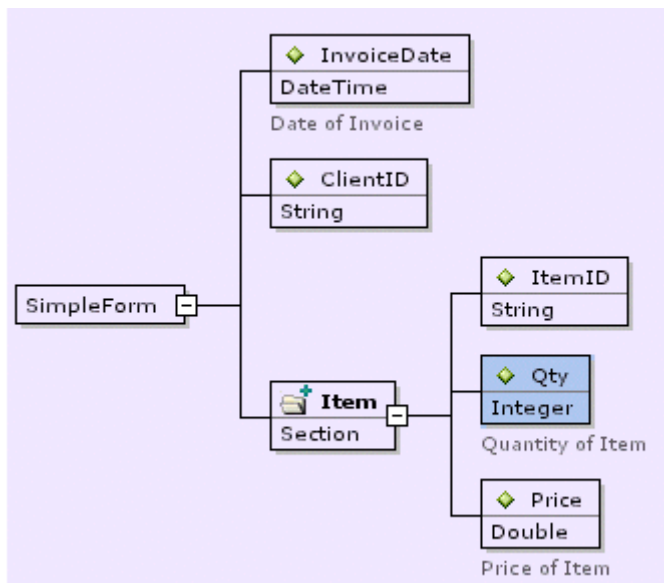
See Also:

Properties

Properties

The 'Properties' button '⊞' enables the user to show or hide the 'Properties' window. The 'Properties' button is 'On' by default, click ⊞ to toggle it. Enabling it would display the Properties window in the bottom of the Designer UI.

When the Properties window is open, clicking on the close button in its title bar can close it.

Right clicking in the title bar of Properties window, displays a pop up with check box 'Properties'. Deselecting the 'Properties' check box would hide the Properties window from the Designer UI.
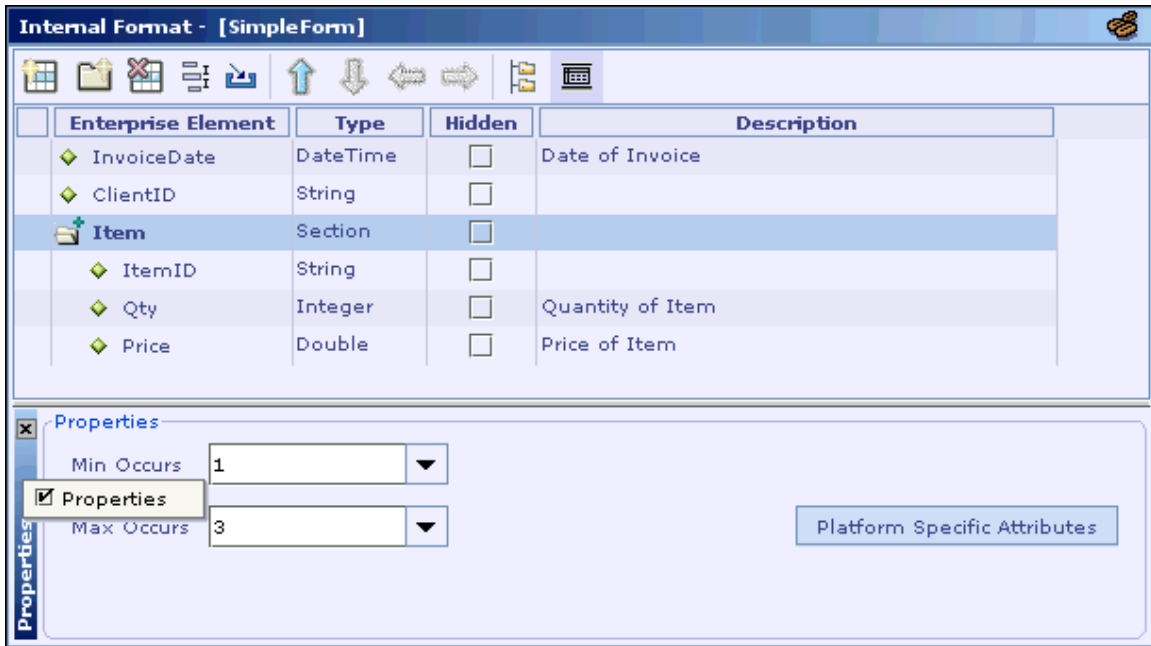
As can be seen in the above picture, the properties window consists of properties corresponding to the Field/Section selected in the message.

See Also:

Field Properties
Section Properties
Design Element UI Pane

Message Window

The Message window displays messages generated during validation and code generation as shown below.
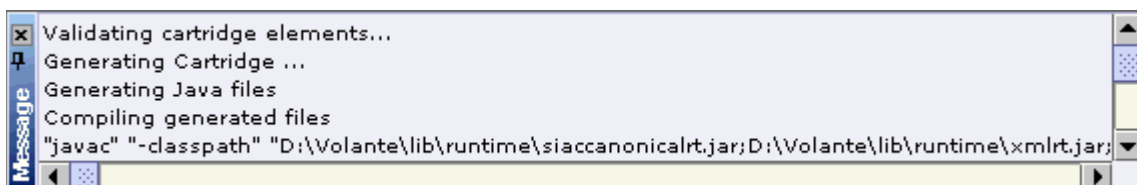
When the Message window displays a list of error messages as shown in the following picture, the user can double click on a particular error message to display the corresponding design element user interface. The selected error message is also displayed in the status bar.
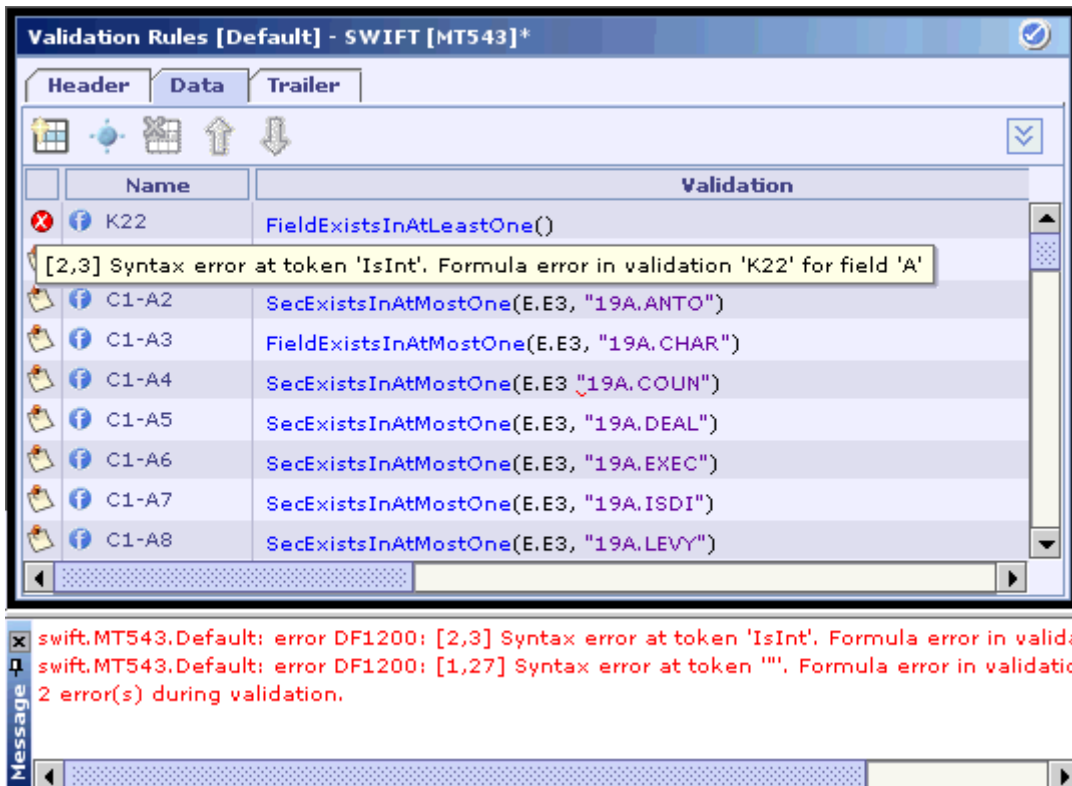
As can be seen in the above picture, moving the mouse pointer over the error icon displays the corresponding error message as tool tip.

The user can show/hide the Message window either by selecting the Message menu item of the View menu or by clicking on the Message button on the Speed Bar.

When the Message window is open, it can be closed by clicking on the Close button present in its title bar.

The Auto Hide mode (Pinned mode) of the Message window can be enabled or disabled, by toggling the Auto Hide button present in its title bar. When Auto Hide mode is enabled (indicated by the ⊨ icon), the Message window hides itself when it loses focus. When Auto Hide mode is disabled (indicated by the ⊩ icon), the Message window remains visible even when it loses focus.

Right clicking inside the Message window displays a popup menu with two items, **Clear** and **Copy**. While the **Clear** menu item clears the content of the Message window, the **Copy** menu item copies the selected text of the Message window to the clipboard.

See Also:

Designer User Interface

Task Window

When a design element is validated, validation errors and warnings are listed as tasks in the Tasks window. New tasks are added to this list whenever new errors/warnings occur. A task is removed from this list only when the corresponding error/warning is rectified.

In the following picture, the Tasks window (the bottom pane) is shown with a task selected.
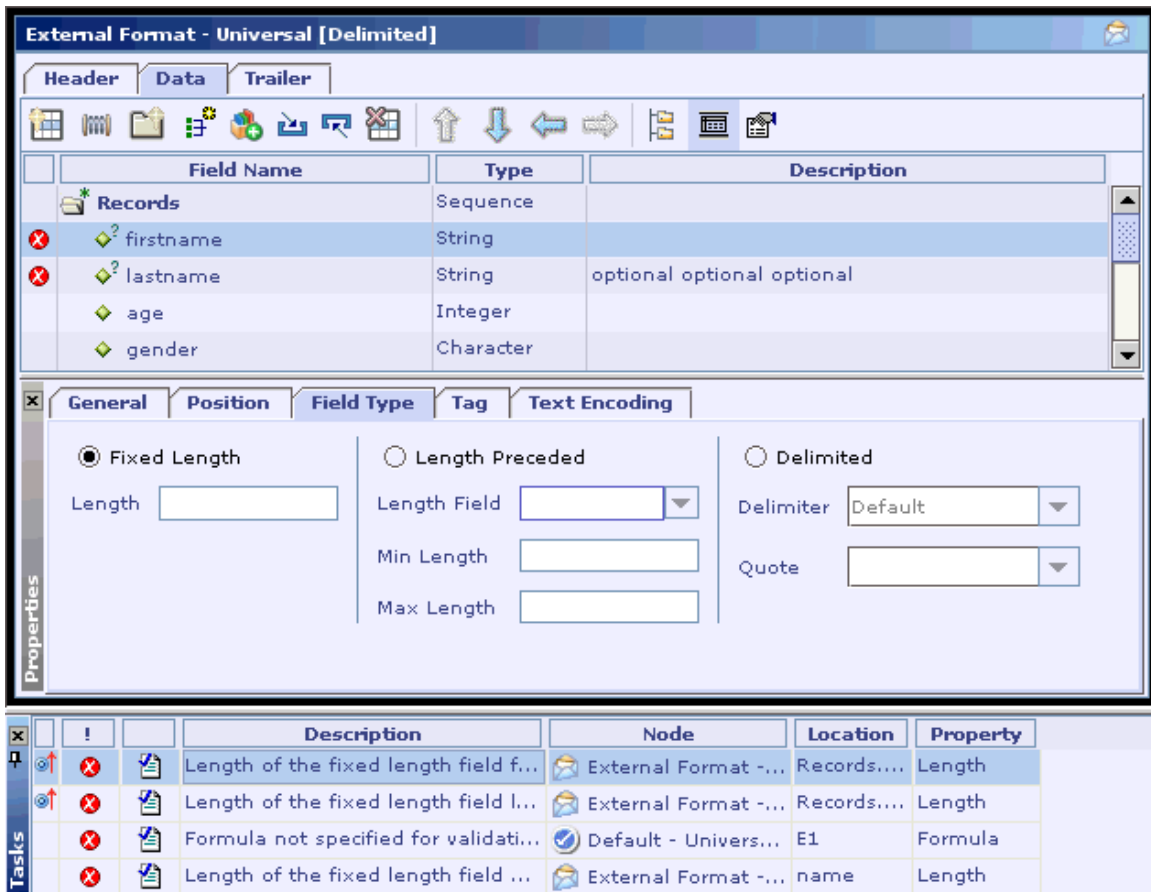
In the Tasks window, the Description column displays the error message.

As can be seen in the picture given below, if you move the mouse pointer over the description of a task, it turns into a hyperlink. If clicked, it takes you to the item causing the validation error such as a field/section, a validation rule or a mapping rule.

A tool tip with full error message is also displayed when the mouse pointer is moved over the description of a task.

The Node column displays the name of the design element node that contains the item causing the error/warning.

The Location column displays the name of the item causing the error/warning.

Tasks that correspond to the same design element as that of the currently selected task are identified by the **Tasks in active node** icon ⊗⊤ displayed in the first column.

Error codes corresponding to the errors/warnings can be displayed by selecting the Code menu item from the short-cut menu displayed when right-clicking inside the column title row.
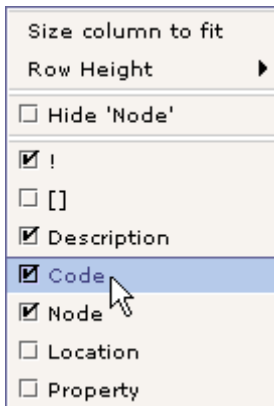
The following picture shows the Tasks table with error code column.

Another useful feature while working with the Tasks table is generating an HTML page for it. The HTML page is a sort of quick view of the current table information. The structure of the generated HTML page will be similar to the current appearance

of the table. If you have hidden a column or reordered the columns, it will be reflected in the generated HTML page.

To generate the HTML page, select the View As HTML menu item from the short-cut menu of the Tasks table, as shown in the above picture. Right-clicking anywhere inside the table, with the exception of the column title row, displays the short-cut menu.

See Also:

Designer User Interface

Search Results Window

The items that match the criteria specified while performing 'Find In Path' or 'Fin Usage' operations are listed in the Search Results auxiliary window as shown below.

The window displays the search results in four columns

Node. The node where a match for the search has occurred. The node type (e.g. External Format - FIX) and the node name (Order41IN) are displayed.

Location. Displays the name of the matched item.
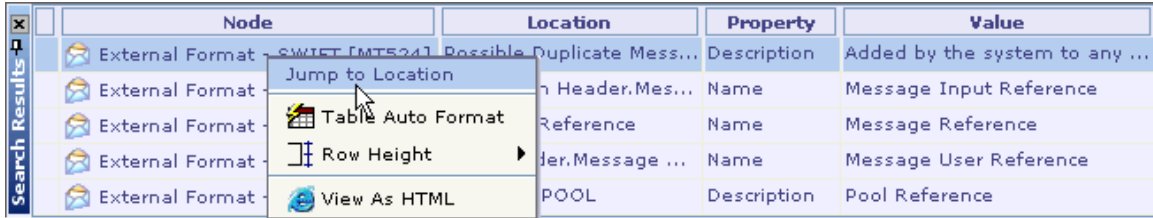
Property. Displays the matched attribute of the item. (E.g. Name, Description)

Value. Displays the matched text. If for e.g. the match has occurred for a field in its description property, the value of the description is displayed in this column.

To navigate to the location where the match has occurred, move mouse over the Node column. A hyperlink is displayed. Click it to navigate to the location.

Jump To Location

To navigate to the location where the match has occurred the user can also select the row in the search results table, right click it and select 'Jump to Location' menu item.

Node	Location	Property	Value
External Format - SWIFT [MT524] Possible Duplicate Mess...	Description	Added by the system to any ...
External Format - ...	Jump to Location	n Header.Mes...	Name	Message Input Reference
External Format - ...	Table Auto Format	Reference	Name	Message Reference
External Format - ...	Row Height ▸	der.Message ...	Name	Message User Reference
External Format - ...	View As HTML	POOL	Description	Pool Reference

View As HTML

To view the search results in the HTML format right click the search results window and select 'View As HTML' menu item. The search results are displayed in HTML form.

See Also:

[Find in Path](#)
[Find Usage](#)

Status Bar

The status bar displays status of the current operation. It also displays the error/warning message currently selected in the Message window.

The status bar is shown below.

See Also:

[Designer User Interface](#)

Speed Bar

Speed Bar displays buttons corresponding to auxiliary windows. The user can show/hide an auxiliary window by toggling the corresponding button.

See Also:

[Designer User Interface](#)

Change Log Window

This feature allows the user to maintain a change log of a Cartridge. To edit/view the 'Change Log' window, click on the Change Log link displayed in the 'Getting Started' pane of Designer when a Cartridge is open.

To add a log entry, click on the 'Add Change Log Entry' tool icon displayed on the left of the 'Change Log' window. Once a new row is added to the 'Change Log' window, you can edit the content each column. To remove a log entry, select the row to be removed in the 'Change Log' window and then click on the 'Remove Change Log Entry' tool icon .

See Also:

Designer User Interface

Element Heading

It is the title bar of the Design Element UI pane. It displays the type and name of the current design element.

See Also:

Designer User Interface

Designer Settings

Designer settings that can be specified are

Option to reopen last file on startup and to
Color schemes viz., Blue, Green, Gray, Plastic Look & Feel and Windows Look &
Feel
Required font & size for all text in designer and
Required font & size for codes in designer.

1. Click on the **File** menu.

2. Select the **Settings** menu item.

    The **Settings** dialog box appears.
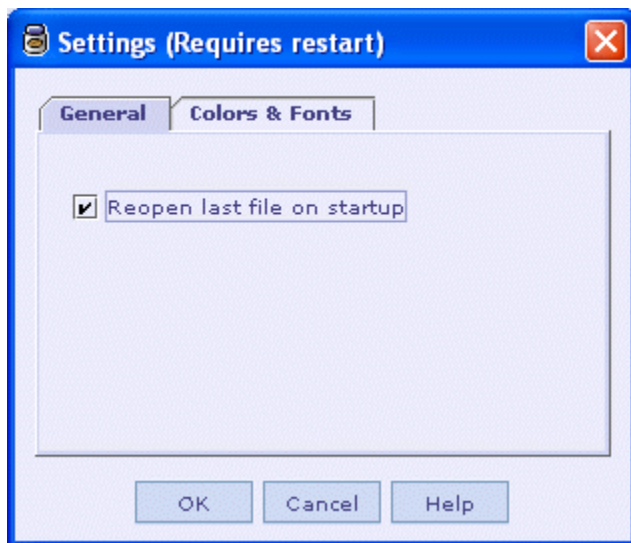
You can specify the designer settings in this dialog.

Note:

The designer must be restarted after making changes to this dialog for the changes
to take effect.

See Also:

Designer User Interface

Recent File List

The recent file list provided by Designer gives easy access to the recently opened cartridges.

Selecting the **File > Recent** menu item displays a submenu consisting of the most recently used cartridges. The user can select one to open the corresponding cartridge. The shortcut key **Ctrl-J** can be used to open the most recently used cartridge.

See Also:
Tables
Navigation Features
Diff
Cartridge Publisher
Cartridge Read-Only Mode
Verify Integrity
Export/Import a Design Element
Copy/Paste
Validating Design Elements
Find
Drag and Drop

Exiting Designer

Follow the steps given below to exit Designer.

1. Click on the **File** menu.

2. Select the 'Exit' menu item.

3. The save cartridge dialog is displayed.
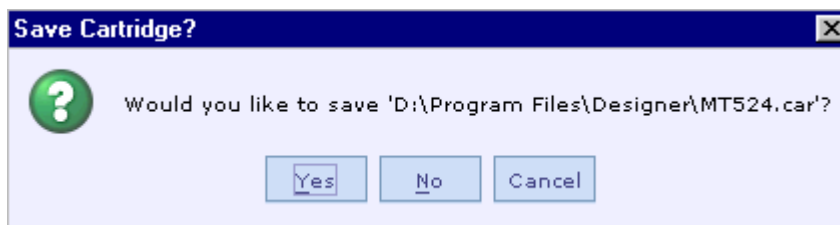
4. Press 'Yes' to save the cartridge and exit the designer. If you do not want to save the cartridge press 'No'. The cartridge will not be saved and designer will be exited. Press 'Cancel' if you do not want to exit designer.

See Also:

[Designer User Interface](#)

Cartridge

A cartridge is a model that captures,
    Format definitions of internal and external messages
    Transformations between internal and external messages
    The workflow that defines the behavior of the runtime application on receiving a
    new message.

Cartridge is saved as a file with car extension.

See Also:

[Creating a Cartridge](#)
[Opening a Cartridge](#)
[Saving a Cartridge](#)
[Validating a Cartridge](#)
[Closing a Cartridge](#)
[Cartridge References](#)
[Designer User Interface](#)
[Message](#)
[Message Mapping](#)
[Formula](#)
[Function Definition](#)
[Code Generation](#)
[Simulator](#)
[Working With Cartridge Designer](#)

Creating a Cartridge

Follow the steps given below to create a new cartridge.

1. Click on the **File** menu.

2. Select the **New Cartridge** menu item.
    The **New Cartridge** dialog box appears.
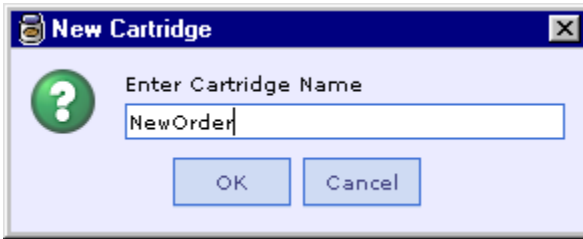
3. Enter the name of the cartridge and click the **OK** button.

4. The cartridge design element node appears at the root of the explorer pane. The cartridge design element UI is shown in the following picture.
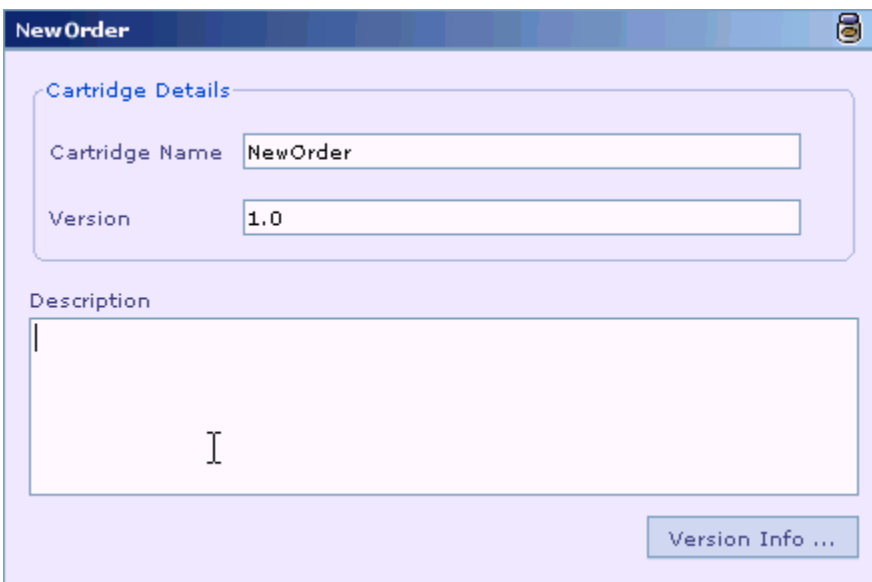
5. Change the version number of the cartridge in the Version text box, if required.

6. Enter the description of the cartridge in the Description text area. This along with the version number helps in identifying a cartridge.

See Also:

Cartridge
Opening a Cartridge
Saving a Cartridge
Validating a Cartridge
Cartridge References

Opening a Cartridge

Follow the steps given below to open an existing cartridge.

1. Click on the **File** menu.

2. Select the Open Cartridge menu item.

3. The **Open Cartridge** dialog box appears.
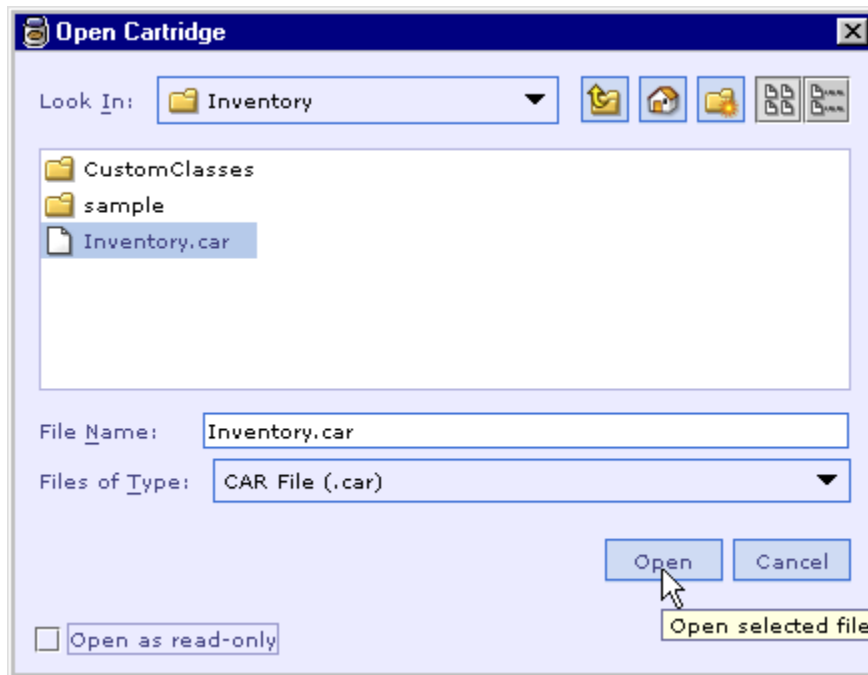
4. Navigate to the directory that contains the cartridge file and select it.

5. Select the **Open as read-only** check box if the cartridge has to be opened as read-only.

6. Click the **Open** button.

7. If the cartridge contains cartridge references and if there are broken references, for each broken reference, it will display the 'Cartridge Reference Error' message box and asks you whether you wish to fix it.
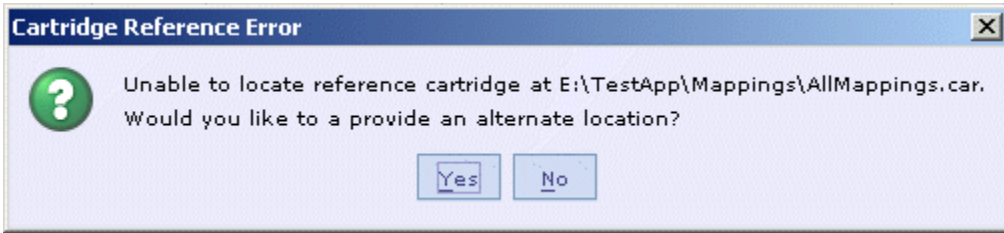
8. If you select the **Yes** button, it will show you the 'Resolve Cartridge Reference' dialog.
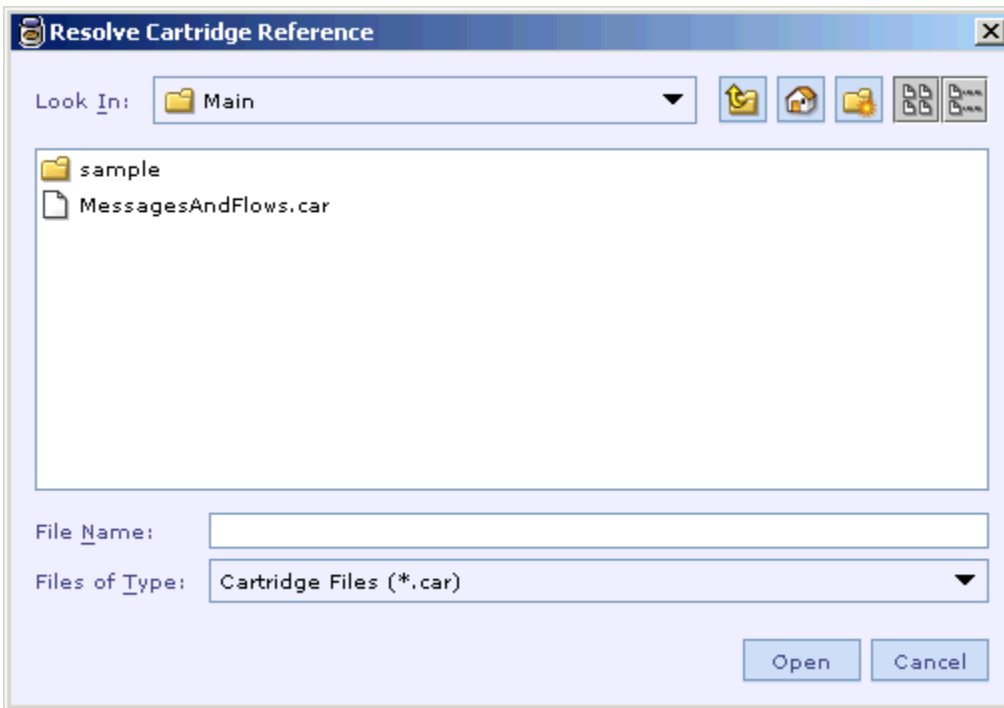
9. Navigate to the directory that contains the cartridge corresponding to the broken link and select it.

10. Click the **Open** button. When all the broken references are fixed, the main cartridge will be opened in the Designer window.

The cartridge is opened in the same frame if it does not have a cartridge already open. If a cartridge is already opened in the current frame, the new cartridge is opened is a new frame.

See the section Cartridge Read-Only Mode for more information on opening a cartridge in read-only mode.

When you save the cartridge, Designer will save the new locations selected for the broken references.

Saving a Cartridge

Follow the steps given below to save a newly created cartridge or to save changes made to a cartridge.

1. Select the **File** menu.

2. Select the **Save** menu item.

3. If the cartridge is new, the **Save Cartridge As** dialog box appears.
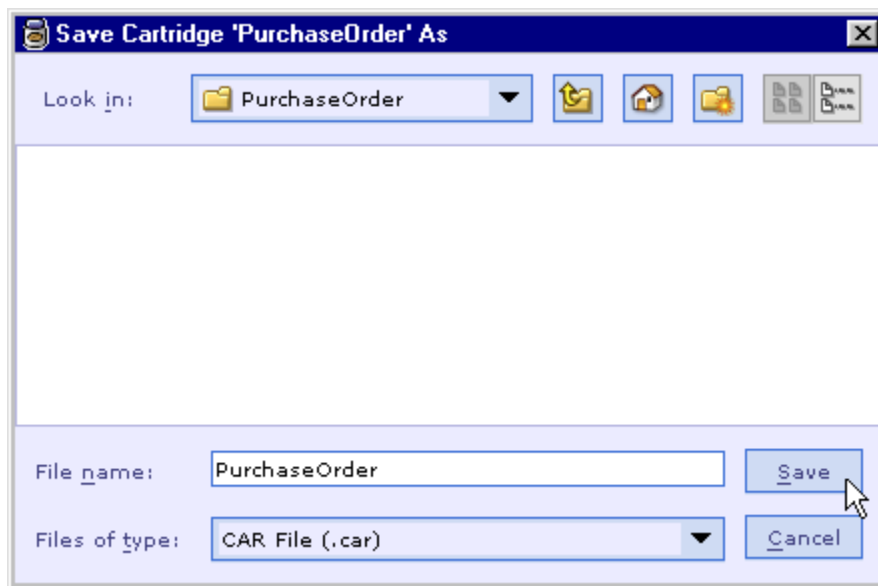
4. Change to the directory under which you want to save the cartridge file.

5. Enter a name for the cartridge file in the **File name** text box and click the **Save** button.

Note:

Designer prompts the user to save the current cartridge when the user tries to create a new cartridge or tries to open another cartridge. The user is also prompted to save the current cartridge, when the user exits Designer.

If the cartridge is already saved in a file, selecting the **File > Save** menu item simply updates that cartridge file.

Use the **File > Save As** menu item to save the current cartridge in a different file from that of the original cartridge file.

See Also:

[Cartridge](#)
[Creating a Cartridge](#)
[Opening a Cartridge](#)
[Validating a Cartridge](#)
[Cartridge References](#)

Validating a Cartridge

A cartridge is a model that captures message definitions, transformations and message flows. This model is later converted to platform specific executable code. For this to be possible, the cartridge model should adhere to some basic rules. These rules enforce certain constraints in the model so that processing of this model does not result in errors. For instance, the name of a field cannot be empty. If this rule is violated it would be difficult to convert the cartridge model to platform specific code.

Select the Validate All menu item from the Build menu to validate the cartridge design element and all of its child elements.

Validating the cartridge ensures that there are no errors that would make the code generation fail.

You can also validate individual elements by selecting the 'Validate' menu item from the short-cut menu that appears when right-clicking that element in the Explorer window.

Note

The cartridge is automatically validated before code generation.

See Also:

[Validating Design Elements](#)
[Cartridge](#)
[Creating a Cartridge](#)
[Opening a Cartridge](#)
[Saving a Cartridge](#)
[Cartridge References](#)

Closing a Cartridge

1. Select the **File** menu.

2. Select the **Close** menu item.

3. Save Cartridge dialog appears as shown below.
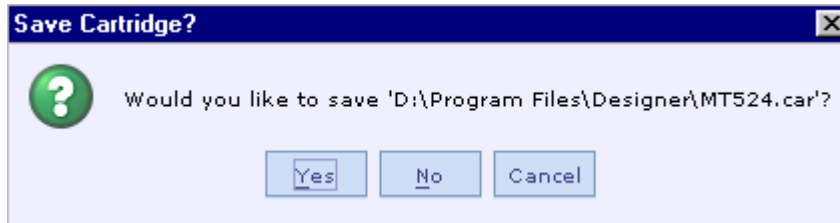
4. Press 'Yes' to save and the cartridge and exit. If you don't want to save the cartridge then press 'No'. The cartridge will be closed without being saved. If you don't want to close the cartridge press 'Cancel'.

See Also:

Creating a Cartridge
Opening a Cartridge
Saving a Cartridge
Validating a Cartridge

Adding Items to Cartridge

The items that can be added to a cartridge are

Message
Message Mapping
Function Definition
Resources
Cartridge References
Folders

To add an item to Cartridge select the cartridge node, right click and select the 'Add Item' menu item
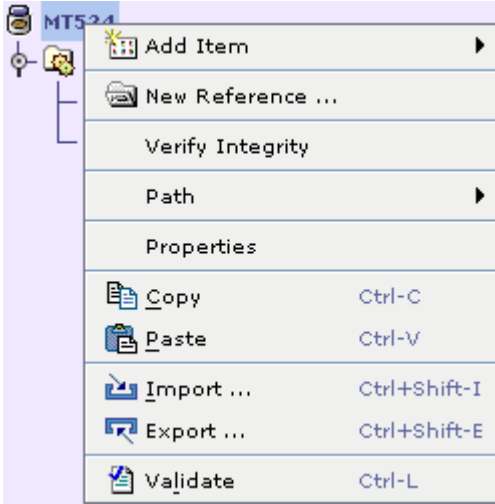
Select the item that you want to add.

You can also select the cartridge and click on the 'New Item' button in the cartridge explorer toolbar to add items to the cartridge.
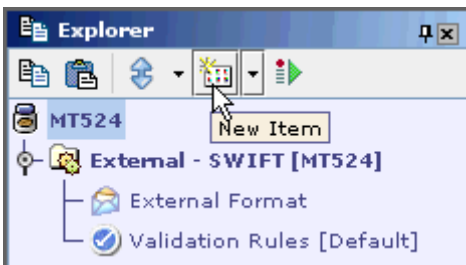
To add a 'Reference' to cartridge, right click the cartridge node and select 'New Reference' menu item to add a new reference to the cartridge.
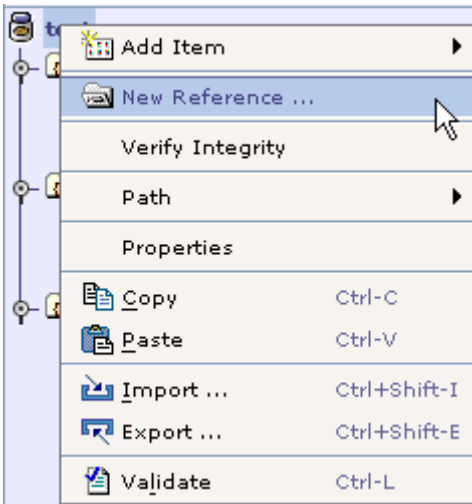
See Also:

Creating a Cartridge
Opening a Cartridge
Saving a Cartridge
Validating a Cartridge
Closing a Cartridge
Cartridge Properties
Cartridge Location

Cartridge Properties

The general properties of a cartridge like 'Author', 'Subject', comments about the cartridge etc. can be viewed and edited in the cartridge properties dialog. Right click the cartridge node and select 'Properties' from the context menu**.**

See Also:

Adding Items In Cartridge

Cartridge Location

One of the common use cases is to go to the Cartridge directory. The user may want to use the deployed Jars, run the command line version, back up the cartridge etc. It may be irritating to the user to locate the cartridge directory by using the explorer, particularly when the cartridge is deep inside some nested directory. The 'Path' menu provides two sub menus that let you open cartridge in explorer and to copy the cartridge path to clipboard.

1. Right click the cartridge node and select 'Path' menu.

2. Select 'Open Containing Folder'. The folder in which the cartridge is located is opened in explorer.

3. To locate the cartridge in 'File Explorer, select 'Locate in File Explorer' menu item under 'Path' menu. The 'File Explorer' window will be displayed with the cartridge selected.

4. To copy the path of the cartridge select 'Copy Path' menu item under 'Path' menu. The path of the cartridge will be copied to clipboard. You can then paste the path in Explorer to locate the folder of the cartridge.

See Also:

[Explorer](#)
[File Explorer](#)

Folders

Folders are just a convenient way of grouping related design elements. Folders do not have any other significance and has no runtime representation.

See Also:

[Adding a Folder](#)
[Deleting a Folder](#)

Adding a Folder

A folder can be added as a child of the cartridge node or another folder.

Follow the steps given below to create a new folder node.

1. In the Explorer window, select a cartridge node or a folder node.

2. Select the **New Folder** menu item from the short-cut menu that appears when you right-click the node or from the **New Item** drop down menu present in the Explorer toolbar.

3. The **Folder** dialog box appears.

4. Enter a name for the new folder.

5. Click the OK button to create the new folder.

See Also:

Deleting a Folder

Deleting a Folder

1. Right click on the 'Folder' element and select 'Delete' menu item.

2. Click 'OK' in the delete dialog that appears to delete the folder.

Note:

When a folder is deleted, the design elements that are added under it will also be deleted.

See Also:
Adding a Folder

Message

Designer supports two types of messages:

1. Internal Message
2. External Message

While an internal message is defined by the enterprise for a particular processing, an external message is defined outside the enterprise. Once the transformation between

an internal message and an external message is defined, the enterprise is able to process that external message.

See Also:

Designer User Interface
Cartridge
Message Mapping
Formula
Function Definition
Code Generation
Simulator
Working With Cartridge Designer

Internal Message

An internal message is a message standardized for use by the enterprise for a particular processing. In Designer, an internal message with its associated processing is represented by an internal message design element.

The internal message design element consists of two child elements:
1. Internal Format
2. Validation Rules

   Here, the **Internal Format** design element is used for defining the internal message structure consisting of data fields and data sections.

   The **Validation Rules** design element is used to check the conformance of data to the business rules. The validation rules node appears with name 'Default'.

During cartridge generation, a component is generated for an internal message. This component is later deployed under Runtime. Once an internal message component is deployed, a client can look it up by using its name for processing the internal message represented by it.

See Also:

Creating an Internal Message
Defining an Internal Message Format
Internal Message Processing
Internal Message Validation Rules
Deleting an Internal Message
Adding Persistence Designer
External Message
Validation Rules
Alias

Creating an Internal Message

Follow the steps given below to add an internal message element to the current cartridge.

1. Select the cartridge design element.

2. Select the **New Internal Message** from the 'Add Item' drop down list present in the Explorer toolbar.

3. The **New Internal Message** dialog box appears.
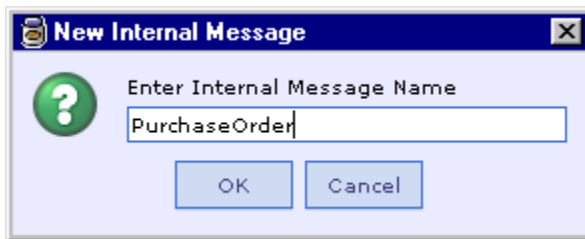
4. Enter a name for the internal message and click the **OK** button.

5. In the explorer pane, a new internal message node with the specified name is created under the cartridge node with its primary child nodes – Internal Format and Validation Rules.

6. Enter the internal message details in the internal message design element UI shown below. Apart from the version and usage information, which are entered in the Version and Description text fields respectively, the user can also enter information in the Standard Details section that is used in tracing the internal message.

See the Tracing Messages in a Cartridge to a Standard section for more information.

See Also:

Defining an Internal Message Format

The Internal Format UI helps in defining the internal message structure by providing facility for adding its constituent fields and sections.

See Also:

Adding a Field

A data field or simply a field represents a single item of information.

Follow the steps given below to add a new field to the internal message format.

1. Select the **Internal Format** design element in the explorer pane.

2. The Internal Format UI is shown in the Design Element UI pane.

3. Click on the **Add New Field** button in the toolbar at the top of the Internal Format UI.

4. A new row appears in the internal format table and its Enterprise Element column is selected by default. If you have selected a row before clicking on the Add New Field button, the new row is inserted immediately after the selected row.

5. Type in a name for the new field.

6. Type in the alias name of the new field in the **Alias** column, if required.

7. The Alias column is hidden by default. Select the Alias menu item from the short-cut menu that appears when you right-click on the header row of the internal format table.

8. See the section Alias for more information on using the alias names.

9. Select the data type of the new field from the list of supported data types (BigDecimal, BigInteger, Binary, Boolean, Character, DateOnly, DateTime, Double, Float, ISODate, ISODateTime, ISOTime, Integer, Long, String and TimeOnly) that appears in the **Type** column.

10. Type in the description of the new field in the **Description** column, if required.

11. Specify the field properties. See the section Specifying the Field Properties for more information.

12. Specify the database related properties of that field by clicking on the **Platform Specific Attributes** button in the **Properties** panel. See the section Specifying the Platform Specific Attributes for more information.

See Also:

Field Properties
Adding a Section

Field Properties

Hidden Property

Set the hidden property of a field by selecting or deselecting the check box that appears in the **Hidden** column of the internal format table. A hidden field cannot be used in mapping. This means that an input mapping cannot be specified for a hidden field and it cannot be used in an output mapping. So a hidden field is not shown in the Mapping Rules UI. But, a hidden field can be used in processing of an internal message.

See the Mappings section for more information on the types of mappings. See the Internal Message Processing and Internal Message validation section for more information on processing/validating an internal message.

Required Property

Set the required property of a field by selecting/deselecting the **Not Null** check box in the **Properties** panel that appears at the bottom of the Internal Format UI. If a required field is not assigned a value even after internal message processing is over, an error is generated indicating the failure of the not null check. Designer indicates an optional field by displaying the ? indicator symbol besides the field icon. No indicator symbol is displayed in case of mandatory fields. In the following picture all fields except the **lastname** field are mandatory.

Enterprise Element
◆ Security_ID
◆ firstname
◆? lastname
◆ age
◆ gender
◆ joindate
◆ designation
◆ salary

Default Value Property

A default value can be specified for a field in the **Default Value** text box of the **Properties** panel. The default value is assigned to that field only if it is not assigned a value during input mapping. The default value is overwritten if a value is assigned to that field during the internal message processing, which is performed next.

See the Message Mapping section for more information on the types of mappings. See the Internal Message Processing section for more information on processing an internal message.

The following points should be noted while assigning a default value for a field.

The data type of the value should match the data type of the field.

The default value for a **DateOnly** type field should be in the format yyyyMMdd.

While specifying the default value for a **String** type field, the string value should not be enclosed in double quotes.

An expression cannot be used to specify a default value.

Length Property

This property is applicable only for a **String** type field. This is used to specify the maximum number of characters allowed for that field value. This property is used in XSD export, db schema generation and webforms generation. But validation is not done for the length of the internal message fields when using NOXML as input.

See Also:

Defining an Internal Message Format
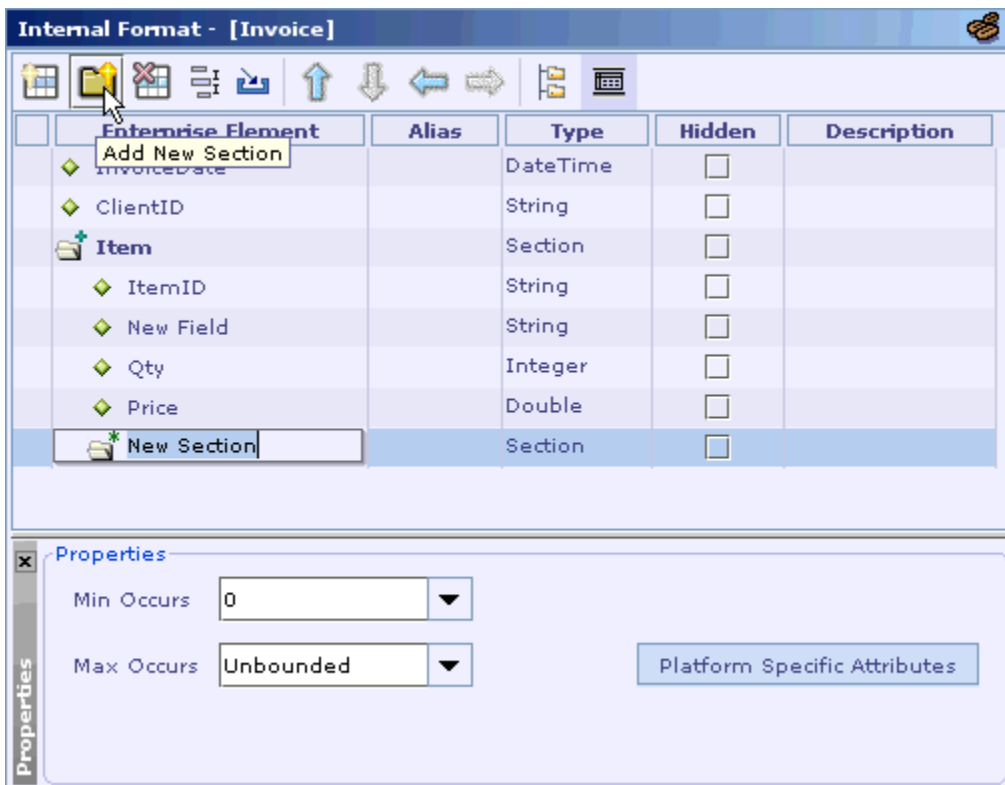
Adding a Section

A data section or simply a section is a set of related fields. A section also has the property of repeating. It means that the entire set of constituent fields can occur a number of times. For these reasons, a section can be thought of as an array of elements, where each element consists of all constituent fields.

For creating a section with its constituent fields, the user should first create a section and then add its constituent fields.

Follow the steps given below to add a new section to the internal message format.

1. Select the internal format design element in the explorer pane.

2. The Internal Format UI is shown in the Design Element UI pane.

3. Click on the **Add New Section** button in the Internal Format UI toolbar.

4. A new row appears in the internal format table and its Enterprise Element column is selected by default. If you have selected a row before clicking on the Add New Section button, the new row is inserted right after the selected row.

5. Type in a name for the new section.

6. Type in the alias name of the new section in the **Alias** column, if required.

7. The Alias column is hidden by default. Select the Alias menu item from the short-cut menu that appears when you right-click on the header row of the internal format table.
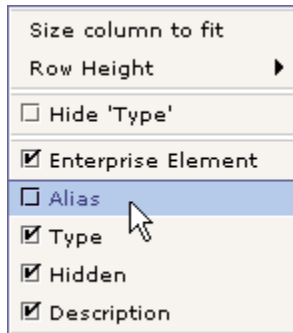
8. See the section Alias for more information on using the alias names.

9. Type in the description of the new section in the **Description** column.

10. Specify the section properties in the **Properties** panel that appears at the bottom of the Internal Format UI. See the section Section Properties for more information.

11. Specify the database related properties of that section by clicking on the **Platform Specific Attributes** button in the **Properties** panel. See the section Platform Specific Attributes for more information.

See Also:

Section Properties
Adding a Field

Section Properties

The section properties specify its repeating and optional attributes.

Min Occurs Property

This property specifies the minimum number of occurrences for a section. A 'Min Occurs' value of 0 means that this section is optional (there can be no occurrences for this section in the message).

Max Occurs Property

This property specifies the maximum number of occurrences for a section. A 'Max Occurs' value of 1 means that this section is non-repeating (there can be only one occurrence for this section in the message) and a value more than one means that this section is repeating. The special value for this property 'Unbounded' means that there can be any number of occurrences for this section.

The table given below describes the repeating and optional characteristics of a section indicated by its 'Min Occurs' and 'Max Occurs' values. It also shows the corresponding cardinal indicator displayed by the Designer for each type of section.

Section Properties

Min Occurs	Max Occurs	Cardinal Indicator	Description
0	Unbounded	*	The section is optional and repeating.
0	1	?	The section is optional and non-repeating.
1	Unbounded	+	The section is non-optional and repeating.
1	1	No indicator	The section is non-optional and non-repeating.

These cardinal indicators, which comply with XML convention, are shown beside the section icons as shown in the following picture.
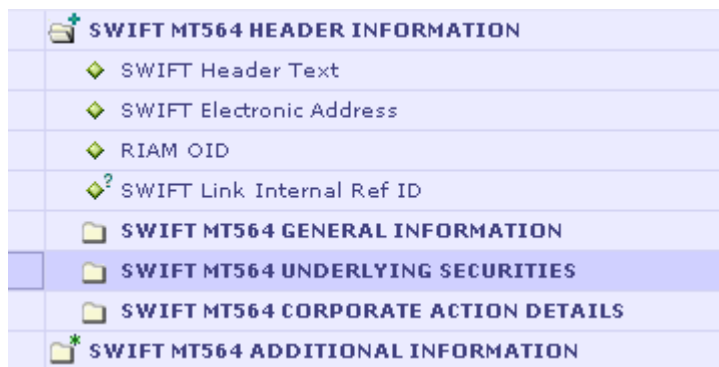
See Also:

Defining an Internal Message Format

Platform Specific Attributes

It is often required to persist an internal message in a database. In Designer, this can be done by adding a persistence designer to the internal message and adding a Persist activity (which refers to that internal message) in the corresponding message flow.

The internal format definition is used in creating the persistence designer. A section in the internal format is represented by a database table and a field is represented by a corresponding database field.

The platform specific attributes of an internal message field/section determine the database properties of the corresponding database field/table.

During cartridge generation, Designer generates a schema file based on these platform specific attributes. This schema file can later be used to generate database specific SQL command files, using the **SQL Generator** tool.

See Also:
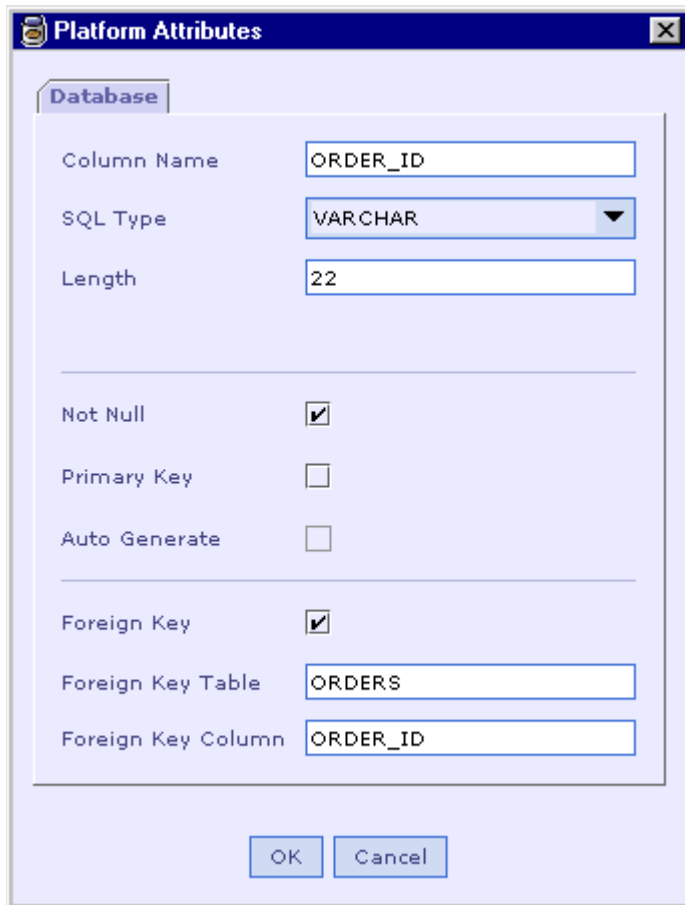
[Platform Specific Attributes of a Field](#)
[Platform Specific Attributes of a Section](#)

Platform Specific Attributes of a Field

The platform specific attributes specified for an internal message field determine the attributes of the corresponding database column.

The **Platform Attributes** dialog box for a field is shown below.

Column Name Attribute

The value of this attribute becomes the name of the corresponding database field. Designer generates an initial value for this field from the field name, based on general database field name conventions. For example, space characters in the original field name are replaced by an underscore character. But it can be changed based on database and application specific requirements, such as limiting the length of the name.

SQL Type Attribute

This determines the type of the corresponding database field. The SQL Type list box shows a set of SQL types based on the data type of the internal message field. The Designer data types and their corresponding SQL types are shown below. The user has to select one of the SQL types based on the underlying database platform and application requirements.

Designer Type	Corresponding SQL Types
BigDecimal	DECIMAL
BigInteger	BIGINT
Binary	VARBINARY
	LONGVARBINARY
	BINARY
	BLOB
Boolean	BIT
Character	CHAR
DateOnly	DATE
DateTime	DATE
	TIME
	TIMESTAMP
Double	DOUBLE
	DECIMAL
	NUMERIC
Float	REAL
	FLOAT
Integer	SMALLINT
	INTEGER

	TINYINT
Long	BIGINT
String	VARCHAR
	LONGVARCHAR
TimeOnly	TIME

Length Attribute

This attribute depends on the Designer type of the field.

For a **String** type field, it specifies the maximum number of characters allowed for the corresponding field value.

For **Integer**, **Long** and **BigInteger** type fields, it specifies the maximum number of digits allowed for the corresponding field value.

For **Float**, **Double** and **BigDecimal** type fields, the user needs to specify both the entire length of the corresponding field value and the length of its decimal part, as required in the **Number** type of Oracle.

For a **Binary** type field, it specifies the maximum number of bytes allowed for the corresponding field value.

This attribute is not applicable for **Character**, **Boolean** and all forms of date/time fields.

Not Null Attribute

This attribute determines the Not Null property of the corresponding data field. If set to true, a value should always be present for the corresponding data field.

Primary Key Attribute

Setting this attribute to true makes the corresponding data field the primary key column and automatically sets the Not Null attribute to true.

Auto Generate Attribute

This attribute is applicable only if the primary key attribute of the corresponding field is set to true. Runtime takes care of generating the primary key value if this attribute is set to true.

The persistence designer uses a unique key generator based on a unique key generation table.

See the Adding Persistence Designer section for information on using the persistence designer to persist internal messages.

Foreign Key Attribute

The **Foreign Key Table** and **Foreign Key Column** text boxes, shown when this attribute is set to true, are used to specify the table and its column referred to by the database column corresponding to this field.

Platform Specific Attributes of a Section

The platform specific attributes specified for an internal message section are used in defining the corresponding database table.

Table Name Attribute

The value of this attribute becomes the name of the database table corresponding to the currently selected section.

Schema Attribute

This attribute specifies the database schema under which the table corresponding to the currently selected section needs to be created.

See Also:

Defining an Internal Message Format

Arranging Fields of an Internal Message Format

The icons with arrow images in the Internal Format UI toolbar can be used to rearrange the position of the fields of an internal message format.

Once the required fields are selected, the user can click on these icons to arrange the position of these fields within the internal message format. Use the selection column (the empty column at the left) to select a field. To select a set of continuous fields, select the first field in the set and then SHIFT-click on the last field. Use CTRL-clicking to select a non-continuous field. CTRL-clicking on a selected field deselects it.

The following table describes the icons used to arrange the fields of an internal message format:

Icon	Purpose	Description
⬆	Move Selection Up	This moves the selected field(s) up by one position within the same level of internal message format (within the same section or within the top level). This icon is disabled when the top of that level is reached.
⬇	Move Selection Down	This moves the selected field(s) down by one position within the same level of internal message format (within the same section or within the top level). This icon is disabled when the bottom of that level is reached.
⬅	Move Selection Left	This promotes the selected field(s) by one level in the internal message format hierarchy. This means that the selected fields are moved to the same level of their previous parent.
➡	Move Selection Right	This demotes the selected field(s) by one level in the internal message format hierarchy. This icon is enabled only when the selected fields are positioned immediately after a section, which is at the same level of the selected field(s). Clicking on this icon makes the selected fields children of that section. The fields are positioned at the bottom of that section.

See Also:

Defining an Internal Message Format

Deleting Fields of an Internal Message Format

Follow the steps given below to remove a set of fields from the internal message format at once.

1. Select the fields/sections to be removed.

2. SHIFT-click in the selection column to select a set of continuous fields and CTRL-click in the selection column to select any non-continuous field without affecting the current selection.

3. Click on the Remove Selected Field(s) icon .

Note

If a section is removed, all of its constituent fields are also removed.

See Also:

Importing Internal Message Field Structure from External Sources

The **Import Field Structure** icon available in the Internal Format UI helps in defining the internal message format by importing field structure from database tables, XML schema files and DTD files.

See Also:

Steps to Import Field Structure From Database Tables

Follow the steps given below to import the internal format structure from tables of the specified data source.

1. Click on the **Import Field Structure** button in the toolbar at the top of the Internal Format UI.
   The **Import Options** dialog is shown.
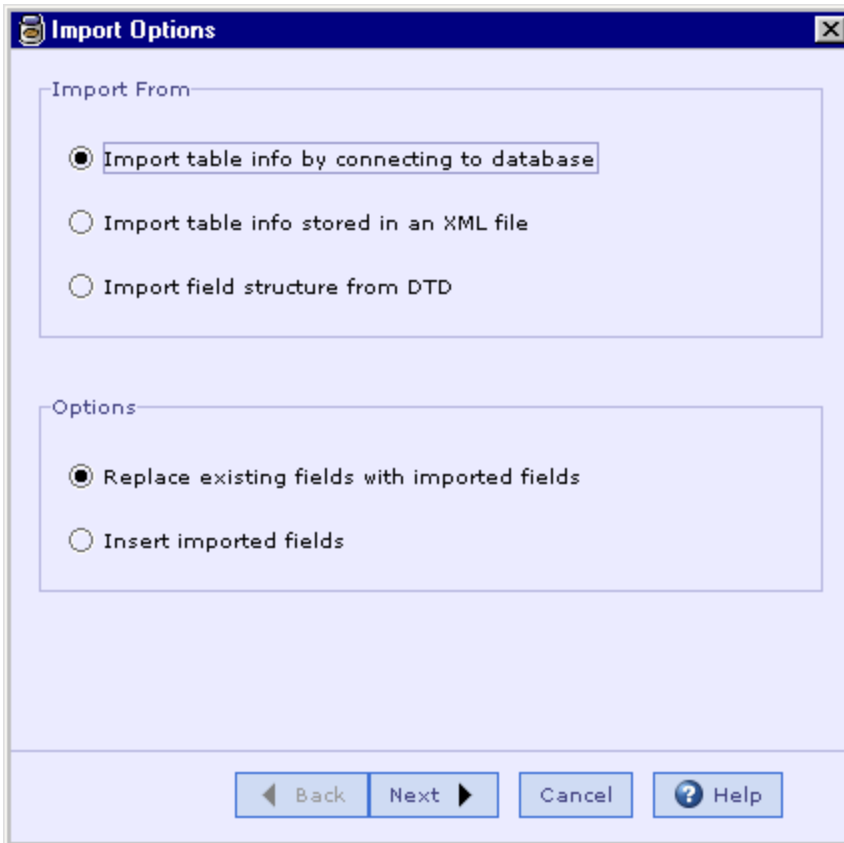
2. Select the Import table info by connecting to database option in the Import From section.

3. In the **Options** section, specify whether you want to keep or replace the existing fields and click on the **Next** button.
The Select Table(s) From Database dialog box is shown.
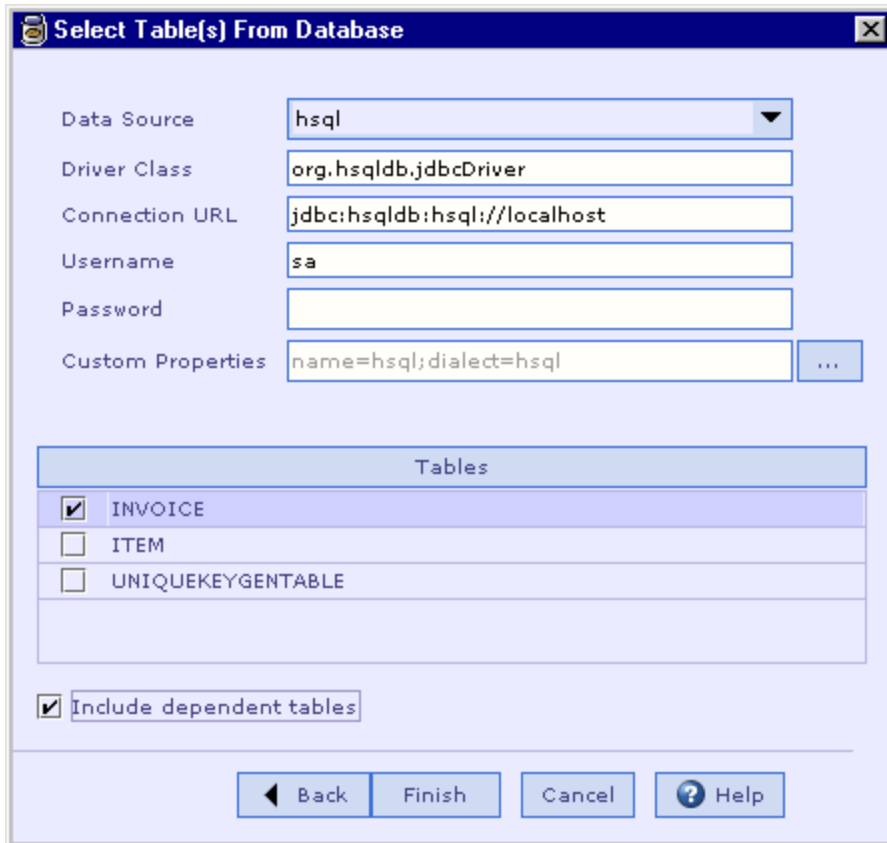
4. From the **Data Source** drop down list box choose the database platform to be connected.

5. In the **Driver Class**, **Connection URL**, **Username** and **Password** text boxes specify the values used in connecting to the database.

6. To specify database specific properties use the **Custom Properties** item.
Clicking on the ellipsis button brings up a **Properties** dialog box that can be used to specify the name and value of each custom property.
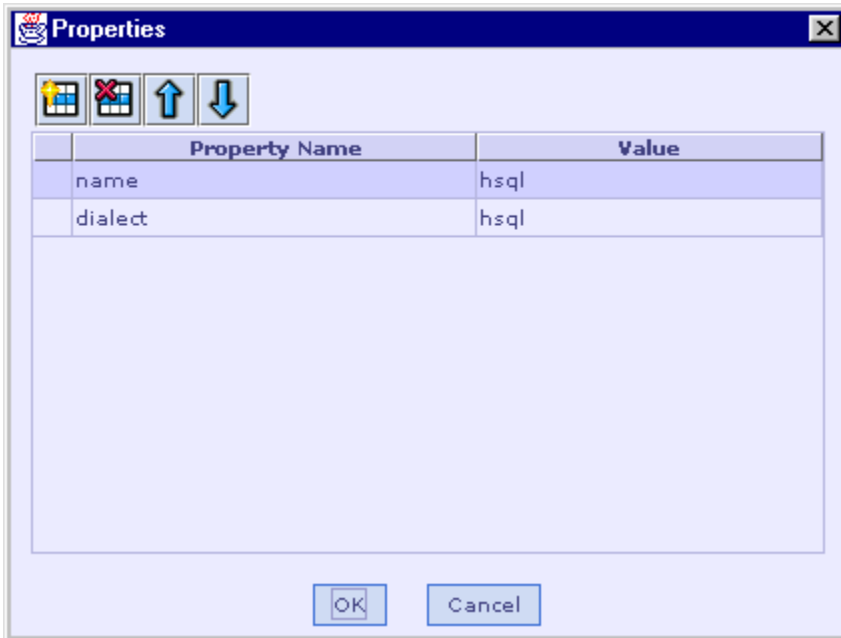
7. Now click on the **Tables** button. Designer connects to the specified data source and populates tables available in that data source.

8. Select the check boxes beside the tables to be imported.

9. Select the **Include dependent tables** check box to import the tables related to the selected tables.

10. Click on the **Finish** button to start importing the field structure.

See Also:

[Importing Internal Message Field Structure from External Sources](#)

Steps to Import Field Structure from an XML Schema File

Follow the steps given below to import the internal format structure from table information available in an XML schema file.

1. Click on the **Import Field Structure** button in the toolbar at the top of the Internal Format UI.
   The **Import Options** dialog is shown.

2. Select the Import table info stored in an XML file option in the Import From section.

3. In the **Options** section, specify whether you want to keep or replace the existing fields and click on the **Next** button.

4. The Select Tables Stored in XML dialog is shown.
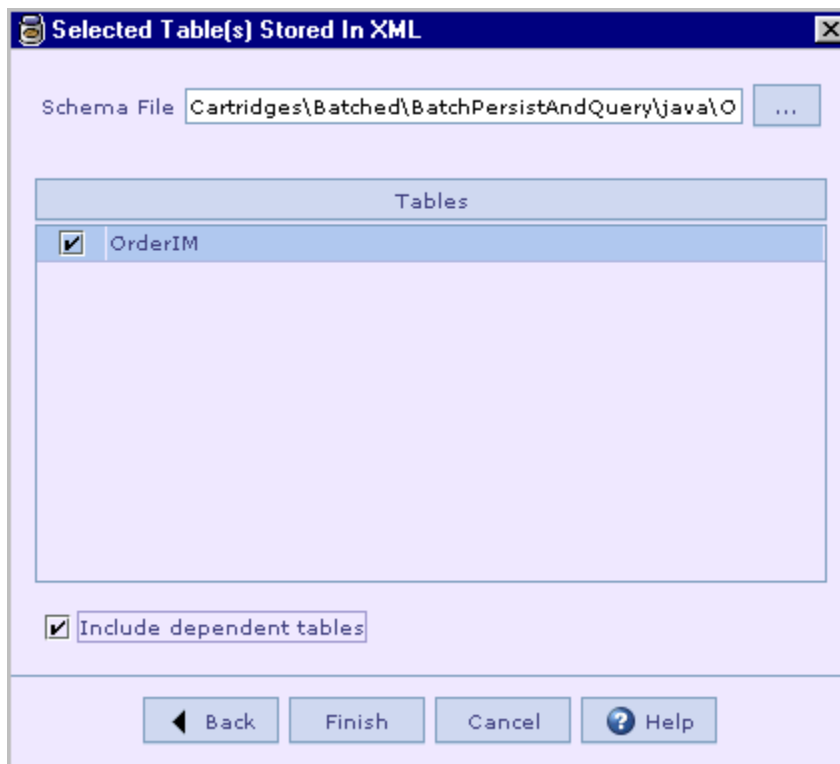
5. In the **Schema File** text box, specify the XML file from which the field structured needs to be imported. Clicking on the ellipsis button besides the **Schema File** text box brings up the **Open** dialog box that can be used to select the required XML schema file.

6. Now click on the **Tables** button. Designer populates table information available in the XML schema file.

7. Select the check boxes beside the tables to be imported.

8. Select the **Include dependent tables** check box to import the tables related to the selected tables.

9. Click on the **Finish** button to start importing the field structure.

See Also:

Importing Internal Message Field Structure from External Sources

Steps to Import Field Structure from a DTD File

Follow the steps given below to import the internal format structure from a DTD file.

1. Click on the **Import Field Structure** button in the toolbar at the top of the Internal Format UI.
   The **Import Options** dialog is shown.

2. Select the Import field structure from DTD option in the Import From section.

3. In the **Options** section, specify whether you want to keep or replace the existing fields and click on the **Next** button.

4. The **DTD Import** dialog box is shown.
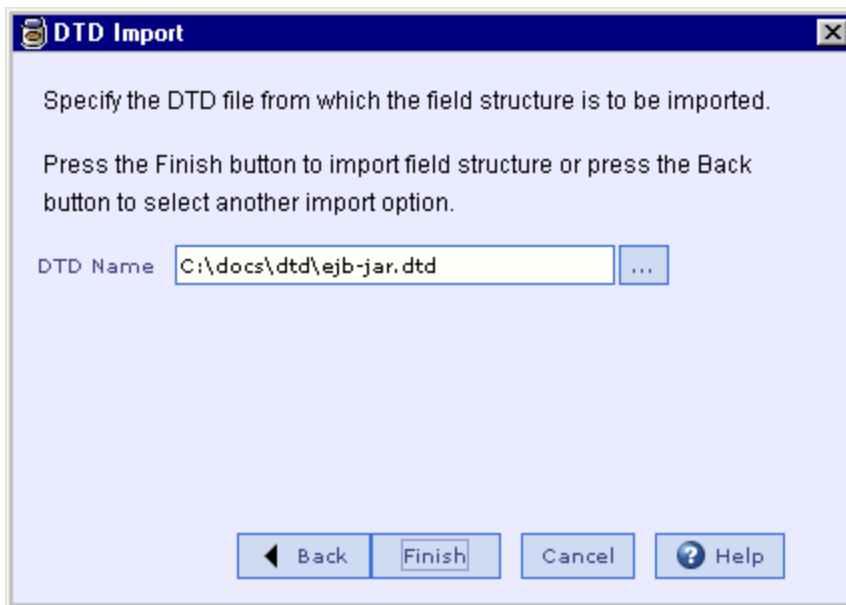
5. In the **DTD Name** text box, specify the name of the DTD file from which the field structure needs to be imported. Clicking the ellipsis button besides the **DTD Name** text box brings up the **Open** dialog box that can be used to select the required DTD file.

6. Click on the **Finish** button to start importing the field structure.

See Also:
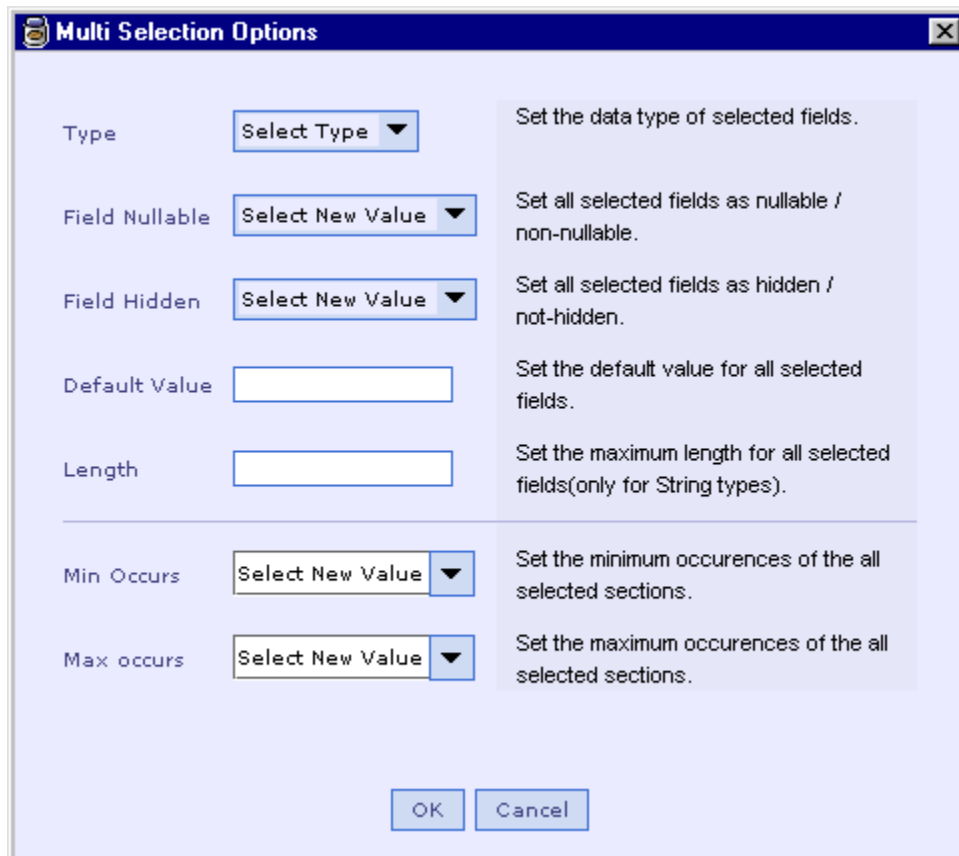
Importing Internal Message Field Structure from External Sources

Specifying Properties for Multiple Fields/Sections

If there is a large number of fields/sections in an internal format, specifying properties individually for each field/section might become time consuming. Designer makes this process easier by allowing the user to specify the common properties for a set of fields/sections at once.

Follow the steps given below to specify properties shared by a set of fields/sections.

1. Select the fields/sections, which share the same properties.

2. SHIFT-click in the selection column to select a set of continuous fields and CTRL-click in the selection column to select any non-continuous field without affecting the current selection.

3. Click on the Set properties for selected fields icon 🗃.
   The **Multi Selection Options** dialog box is shown.

4. Specify the field properties shared by the selected fields by using the items available in top section of the dialog box.

5. Specify the section properties shared by the selected sections by using the items available in the bottom section of the dialog box.

6. Click on the **OK** button to apply the specified properties to the selected fields.

See Also:

Importing Internal Message Field Structure from External Sources

Internal Message Processing

The Internal message Processing Rules is used for enriching the Internal message data. When an Internal message is added it appears with child nodes 'Internal Format' and 'Validation Rules'.

Processing rules node can be added by right clicking the Internal message node and selecting 'Processing' menu item under 'Add' menu.

The Processing Rules UI shown when you select the Processing Rules node of the Internal message node is used to specify the Processing Rules.

See Also:

Processing Rules UI
Adding Processing Rule
Renaming a Processing Rules Node
Creating an Internal Message
Defining an Internal Message Format
Internal Message Validation Rules
Deleting an Internal Message
Adding Persistence Designer
Internal Message

Adding Processing Rule

1. Right click the Internal Message node and select 'Add' menu.

2. Select 'Processing'. 'New Processing Rules' window appears. Enter the name of the 'Processing Rules' node to be added.
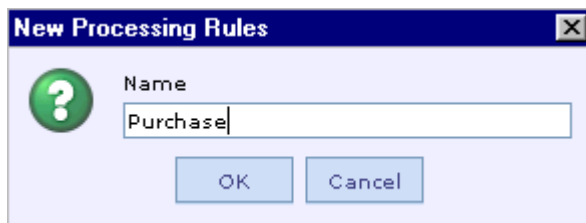
3. The Processing rules node with name "Purchase" gets added as the child node of the Internal message node as shown below.
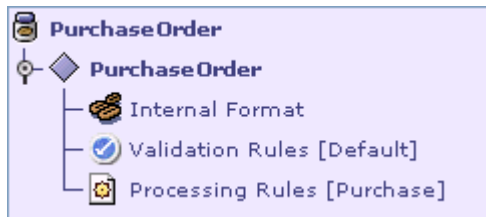
As you can see from the above diagram, the Processing Rules node 'Purchase' is added below the validation rules node.

Note:

The Processing rules node can be deleted by right clicking the Processing rules node and selecting the context menu item "Delete".

Multiple Processing Rules nodes can be added to an Internal message by right clicking the Internal message node and selecting the 'Processing' menu item in the 'Add' menu.

See Also:

Renaming Processing Rules Node
Processing Rules UI
Custom Message Processing

Renaming Processing Rules Node

1. Select the 'Processing Rules' node to be renamed.

2. Right click the Processing Rules node and select the menu Rename.

3. The 'Rename' window appears as follows.
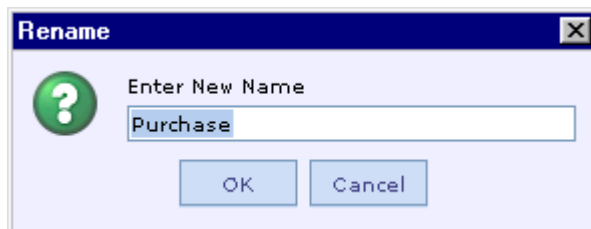
4. Enter the new name for the Processing Rules node.

5. The processing rules node gets renamed with the new name.

See Also:

[Adding Processing Rule](#)
[Processing Rules UI](#)
[Custom Message Processing](#)

Processing Rules UI

The Processing Rules UI is used to specify the rules for enriching the Internal message data.
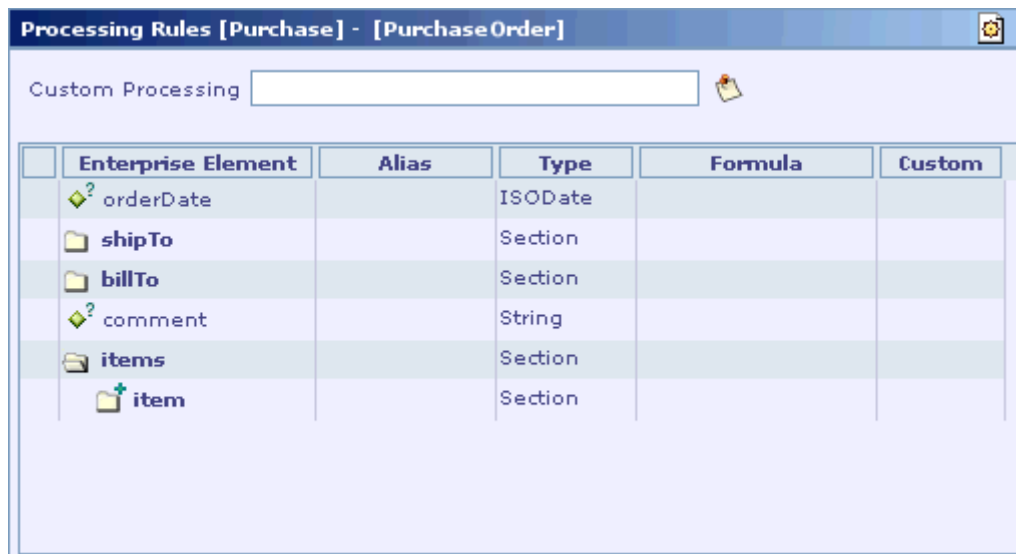
See the section [Moving from a Field to its Usage Items](#) for information on quickly moving from a field definition to its processing rule.

See the section [Moving Back to a Field Definition](#) for information on quickly moving from a field processing rule to the field definition.

See Also:

[Custom Message Processing](#)
[Processing an Internal Message Field](#)
[Internal Message Validation Rules](#)

Custom Message Processing

The processing of the entire internal message can be customized through the **IProcessing** interface. All the user needs to do is define a class that implements this interface and then specify its reference name in the **Custom Processing** text box in the Processing UI. See the section [Defining Processing Classes](#) in the [DefiningCustomClasses.doc](#) file for information about a processing class.

Please refer New File from Template for easily creating a custom processing class.

Follow the steps given below to specify a Custom Processing class for an internal message.

1. Specify the reference name of the custom processing class in the **Custom Processing** text box in the Processing Rules UI.

2. Add comments on the custom processing in the Comment dialog box, which is displayed when you click on the **Add Comment** button.
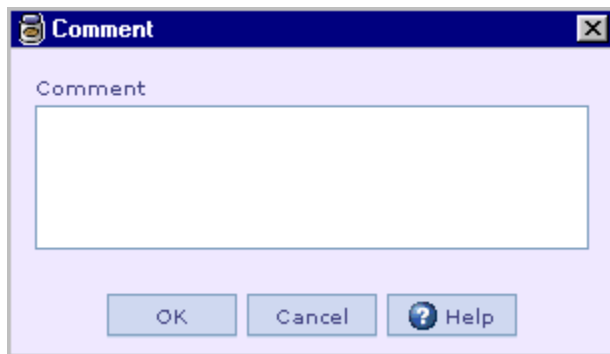
3. Bind the reference name to the actual class name using the **Language Bindings** tab of the corresponding **Code Generation Settings** dialog.

   See the Code Generation section for more information.

Processing an Internal Message Field

Processing of an internal message field can be specified either using a formula or using a custom class.

Custom Field Processing

The processing of an internal message field can be customized through the **IFieldProcessing** interface. All the user needs to do is to write a class that implements this interface and provide its reference name in the **Custom** column of the field to be processed. See the section Defining Processing Classes in the **DefiningCustomClasses.doc** file for information about a field processing class.

Please refer New File from Template for easily creating a custom field processing class.

Follow the steps given below to specify a Custom Processing class for an internal message field.

1. In the Processing Rules UI table, select the row corresponding to the internal message field for which processing needs to be specified.

2. Specify the reference name of the custom field processing class in the **Custom** column.

3. Optionally add your comments on the custom field processing.

    Comments can be added in the Comment pane, which is displayed when you click on the Insert Comment menu item of the short-cut menu that appears when you click inside the table. See the section Comment/Annotation Support in Tables for more information.

Field Processing Formula

Follow the steps given below to specify the processing formula for an internal message field.

1. In the Processing Rules UI table, select the row corresponding to the internal message field for which the processing formula needs to be specified.

2. Specify the formula to be processed in the **Formula** column.

3. Pressing the **F4** key or clicking the **Edit Formula** icon brings up the **Edit Formula** dialog box that can be used in defining the formula.

4. See the Formula section for more information on specifying a formula.

5. Optionally add your comments on the custom field processing.

6. Comments can be added in the Comment pane, which is displayed when you click on the Insert Comment menu item of the short-cut menu, that appears when you click inside the table. See the section Comment/Annotation Support in Tables for more information.

The following points should be noted while specifying a formula:

The type of the formula (type of its return value) should match the data type of the field for which it is specified.

The other fields that can be accessed in the formula specified for an internal message field depends on its position in the internal message structure.

See the section Fields Accessible in a Formula for more information on fields accessible from the field for which a formula is being specified. See the Formula section for more information on specifying a formula.

While specifying processing for an internal message field, you can easily move to the definition of that field in the Internal Format UI.

This can be done by clicking on the **Go To Field Definition** menu item from the short-cut menu that appears when you right-click on the corresponding row.

See the section Moving Back to a Field Definition for more information.

Note:

Specifying both a formula and a custom class for field processing generates an error during validation of the Processing Rules node.

Deleting an Internal Message

Follow the steps given below to remove an internal message node from a cartridge.

1. In the **Explorer** pane, right-click on the internal message node to be removed.

2. Select the **Delete** menu item from the popup menu that appears.
    A Confirm Delete message box is shown.

3. Click on the **Yes** button to remove the internal message from the cartridge.

4. Click on the **No** button to cancel the delete operation itself.

Note
The user needs to save the cartridge to permanently remove the internal message from that cartridge.

See Also:

Creating an Internal Message
Defining an Internal Message Format
Internal Message Processing
Internal Message Validation Rules
Adding Persistence Designer
Internal Message

Adding Persistence Designer

Designer provides an easy way for persisting internal messages. It requires the completion of the following operations.

1. Adding the persistence designer to the corresponding internal message element.

2. Executing the database schema file (which is generated from the persistence designer definition during cartridge generation) in the underlying database.

3. Adding a Persist activity (which refers to that internal message) in the corresponding message flow.

Follow the steps given below to add the persistence designer to an internal message element.

1. Select the internal message element to which the persistence designer needs to be added.

2. Select the **Add Persistence Designer** menu item from the popup menu that appears when you right-click the internal message element.
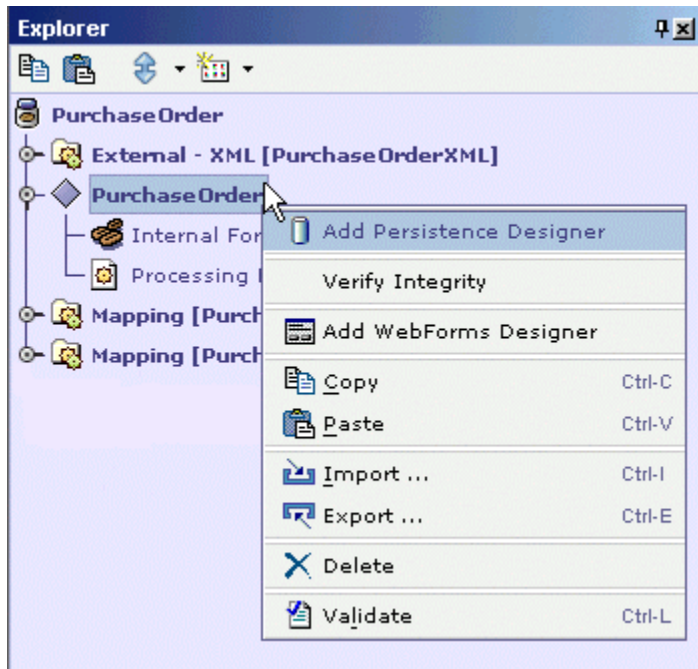
The **Persistence Designer** node is added to the internal message element. The platform specific properties defined for the internal message elements are used in creating the persistence designer. See the section Platform Specific Attributes for more information. See PersistenceDesigner.doc for more information on persistence designer.

Note

The popup menu of an internal message node displays the **Add Persistence Designer** menu item only when it does not have one already.

See Also:

Creating an Internal Message
Defining an Internal Message Format
Internal Message Processing
Deleting an Internal Message
Internal Message

External Message

An external message is a message defined outside the enterprise very often by a standards group such as SWIFT that follows into/out of the Runtime of the

enterprise. In Designer, an external message with its associated validations is represented by an external message design element.

When you define a new External Message, you are defining the following:

> the structure of the message
> parser, to parse the message
> writer, to serialize the message
> validator, to validate the message

The following points should be noted when using an external message element:

> The same external message format can be used to represent both input/output messages (unlike the input/output elements of the old model).

> An external message element does not have a built in flow, unlike the input element of the old model for instance. Each of its processes (parse, write and validate) has to be explicitly invoked from the message flow.

> Unlike the input/output elements, the external message element does not include the mapping node. The mapping from/to an external message has to be defined using the Mapping node and invoked explicitly from the message flow. See the Message Mapping section for information on the types of message mappings and how to define them.

An external message consists of two child elements:

1. External Format
2. Validation Rules

The **External Format** element is used for defining the external message structure. Usually external messages consist of header, data and trailer sections. While the header and trailer sections provide additional information about the message itself (such as the date of sending the message, the size of the message, and so on), the data/record section contains the actual message data. The message data might be a single record or a set of records (in case of a batch).

The **Validation Rules** element is used for specifying validation rules to be applied on the external message fields.

See Also:

Creating an External Message Node
Defining an External Message Format
Creating a Standard Message
Internal Message

Creating an External Message Node

Follow the steps given below to create a new external message design element:

1. Select the cartridge node or a folder node in the Explorer pane. Right click and select 'New External Message' menu item under the menu 'Add Item'.

2. The **New External Message** dialog will be displayed.

3. Specify the name of the external message in the **Transformation Name** text box.

4. This is the name, which is later used for looking up the component generated for this external message.

5. Select the external message format type from the **External Message** drop down list box.

6. Designer currently supports common message formats such as FIX, SWIFT and so on. Please note that your Designer installation may not contain support for all message formats supported by Designer.

7. Click on the **OK** button.

   For most of the message formats, another dialog box is presented to the user as shown below. In this dialog box, the standard messages are organized into different categories. The user can search for a standard message by typing text into the **Find Message** text box and clicking the **Find** button.
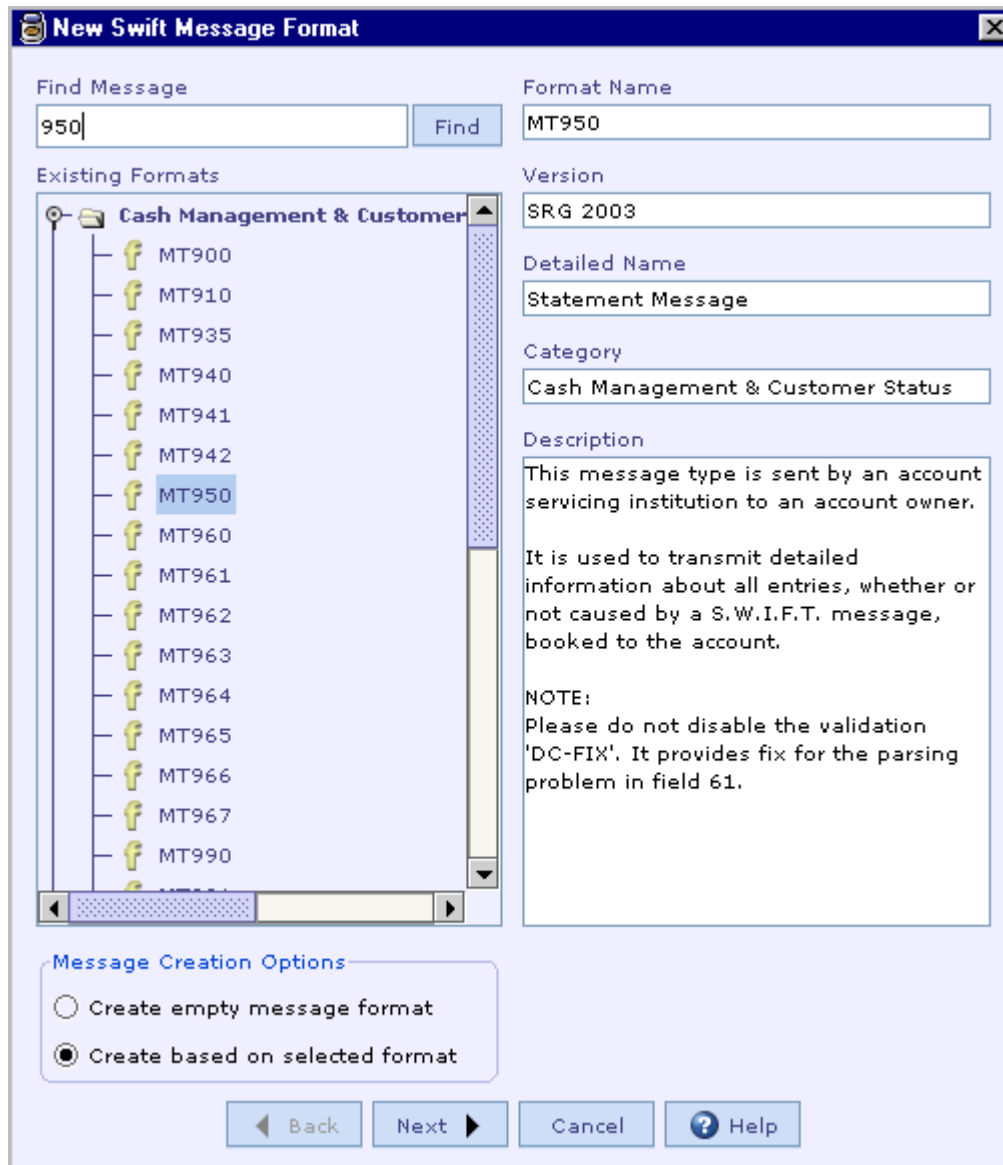
   Designer creates the external message element when the user selects one of the standard messages as shown in the following picture. Alternatively the user can choose to create an empty message format that can be used in defining a new message format.
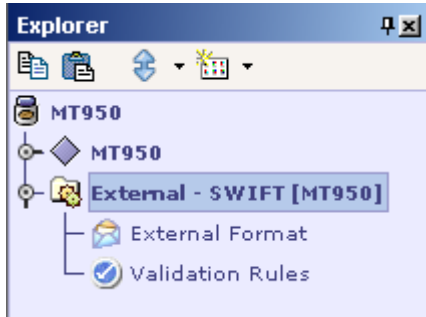
See Also:

[External Message](#)
[Defining an External Message Format](#)
[External Message Validation Rules](#)
[Creating a Standard Message](#)

Defining an External Message Format

Defining the field format of an external message depends on the selected message format. For some formats such as ASCII and XML, the user needs to create new fields and sections as done for defining a new internal message format. See the section [Defining an Internal Message Format](#) for more information.

The user can customize the external message format by enabling/disabling the optional fields by selecting/deselecting the **Enabled** check box from the External Format UI. This feature is supported in formats such as FIX and SWIFT that allow the user to select a standard message and in XML format that allows the user to define the message format by importing a Schema (DTD or W3C XSD). The disabled fields are not shown in the mapping/validation rules UI.
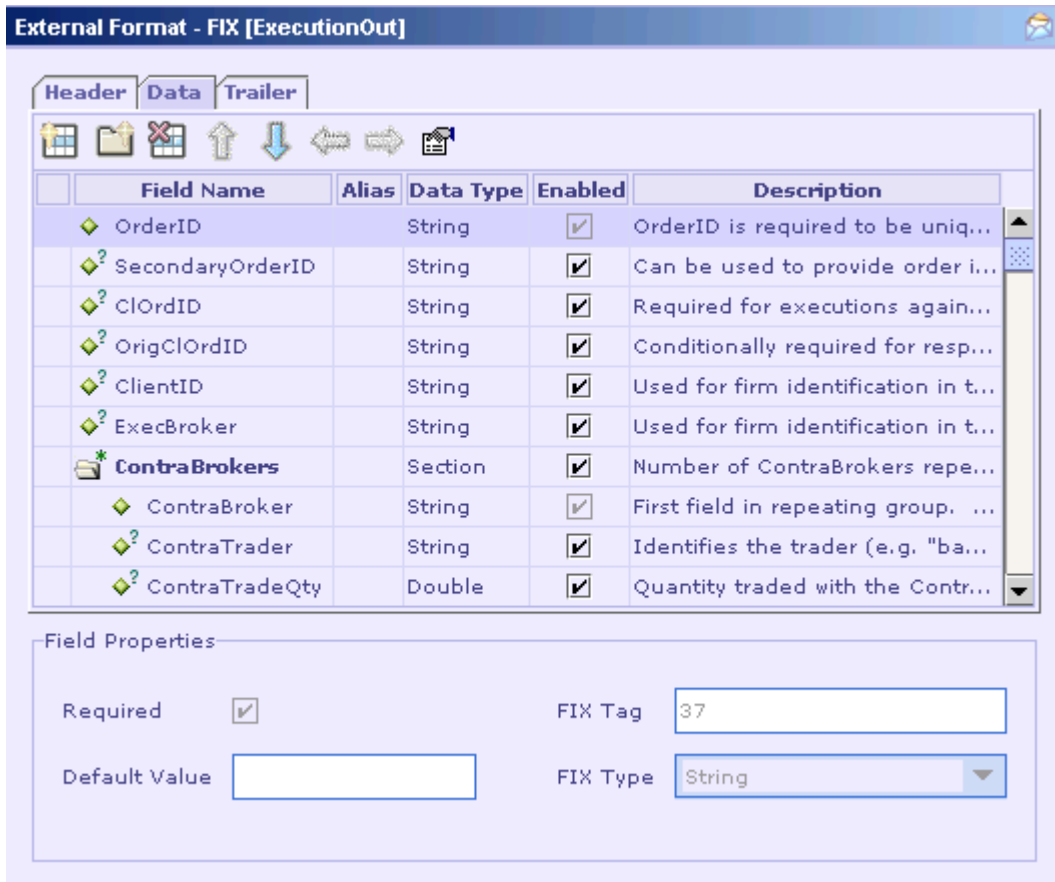
See Also:

[External Message](#)
[Creating an External Message Node](#)
[External Message Validation Rules](#)
[Creating a Standard Message](#)

Creating a Standard Message

Designer also allows for creating a new standard message format for most of the supported message formats.

Please note that your Designer installation may not contain support for all message formats supported by Designer.

For defining a new standard message, the user needs to do the following:

Create an empty external message format. See the [Creating an External Message Node](#) section for information on creating an empty message format.

Define the external message structure and associated validation rules. See the [Defining an External Message Format](#) section for information on defining the external message structure. See the [External Message Validation Rules](#) section for information on specifying validation rules for external messages.

Export the new message to the message repository of Designer.

In case of a SWIFT external message, it can be exported to the message repository by selecting the Save SWIFT Message Format menu item from the popup menu that appears when you right-click the message node.
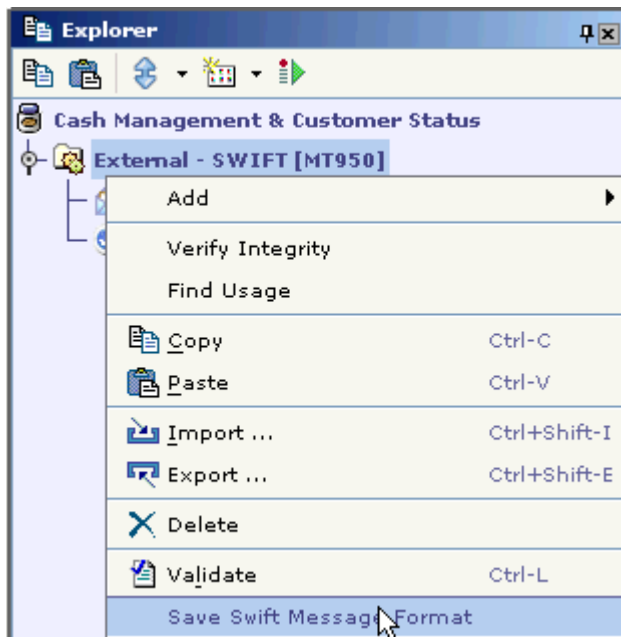
This brings up the **Save SWIFT Message Format** dialog shown below.

Here, you enter the message format name, its category and other details. Clicking on the OK button saves the message format.

See Also:

External Message
Creating an External Message Node
Defining an External Message Format
External Message Validation Rules

Validation Rules

A validation rule is used to check the conformance of data to the business rules. Like field level and record level validations (validations based on more than one field) that can be applied for an RDBMS table, Designer supports validations for a particular message element and also validations based on multiple message elements.

When an External/Internal message is added, it appears with corresponding 'External Format' or 'Internal Format' node and 'Validation Rules' node. When a message is added the Validation rules node appears with the name 'Default'.

The Validation Rules UI shown when you select the Validation Rules child node of the message is used to specify the validations.
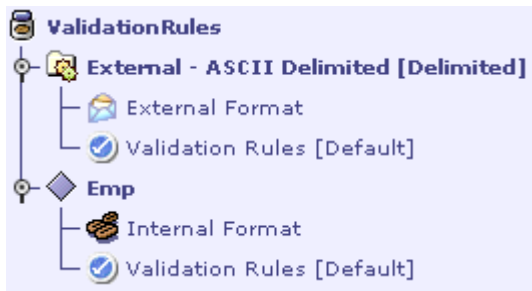
By right clicking the message node and selecting 'Validation' menu item from 'Add' menu, multiple validation rules can be added. The validations that are added can be invoked through the message flow.

Validation rules can be specified using the following:
1. Formula
2. Custom class

A formula validation rule should be a Boolean expression. While a return value of **true** indicates the conformance of the data to the business rules, a **false** value indicates the non-conformance of the data to the business rules.

While a formula can be used to specify a simple validation rule, a custom class can be used to incorporate a validation rule that is more involved.

Designer allows specification of validation rules for internal message and external message design elements. The components generated for these design elements include the validation rules specified for them, which are later invoked during runtime.

See Also:

Adding Validation
Renaming Validation Rules Node
Validation Rules UI
Formula Validations
Fields Accessible in a Formula
Invocation of Validation Rules
Internal Message
External Message
Alias
Tracing Messages in a Cartridge to a Standard
Message

Adding Validation

1. Right click the External/Internal Message node and select 'Add' menu.

2. Select 'Validation'. 'New validation' window appears. Enter the name of the 'Validation Rules' node to be added.
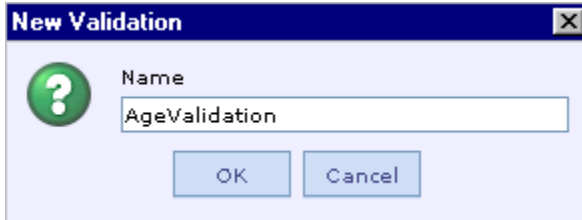
3. The 'Validation rules' node gets added as the last child node of the External Message node as shown below.
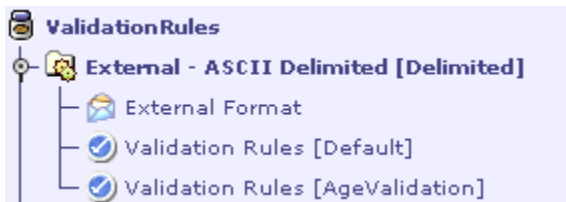
As you can see from the above diagram, the Validation rules 'AgeValidation' appears below the default Validation Rules node.

See Also:

Renaming Validation Rules Node
Validation Rules UI
External Message
Internal Message

Renaming Validation Rules Node

1. Select the 'Validation Rules' node to be renamed.

2. Right click the validation rules node and select the menu 'Rename'.

3. The 'Rename' window appears as follows.
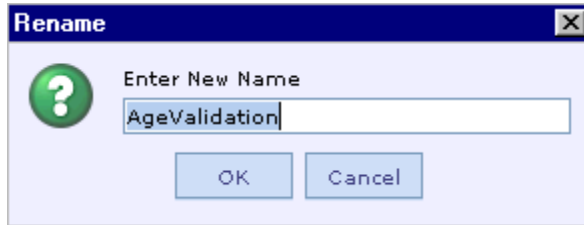
4. Enter the name as to be changed.

Note:

Please note that the 'Default' validation rules node cannot be renamed or deleted.

See Also:

Validation Rules UI
External Message
Internal Message
Renaming Processing Rules Node

Validation Rules UI

The Validation Rules UI of an External/Internal message supports the following views.

   View that shows field-wise validations.
   View that shows all validations.

The functionality of Validation Rules UI for an External message and an Internal message are the same except that, in an Internal message Validation Rules UI, there are no 'Header', 'Data' and 'Trailer' sections.

The view that shows field-wise validations for an External message is shown below:

As can be seen in the above diagram, the field-wise validations view is divided into two tables: while the top table (message format table) displays the constituent fields of the message format, the bottom table (field validations table) displays the validation rule(s) corresponding to the currently selected field in the top table.

The view that shows all validations for an External message is given below:

It shows only the field validations table. But it contains the **Applies To** column in addition to the columns found in the field validations table of the field-wise validations view.

You can easily switch between these two views by clicking on either the **Show All Validations** button or the **Show Validations Field-wise** button that is shown depending on the current view.

Validation rules can be entered using both views. While the field-wise validations view is convenient for entering field specific validation rules, the all field validations view is convenient for entering validation rules that involve multiple message elements.

See the section Moving from a Field to its Validation Rule for information on quickly moving from a field definition to its validation rule.

See the section Moving Back to a Field Definition for information on quickly moving from a field validation rule to the field definition.

See Also:

Adding a Validation Rule
Formula Validations
Fields Accessible in a Formula
Invocation of Validation Rules

Adding a Validation Rule

Usually a validation rule is specified (or applied) for a field/section. The field/section to which a validation rule is applied determines the invocation of that validation rule. It also determines the field(s)/section(s) that can be accessed in that validation rule.

See Also:

Adding a Validation Rule using Field-wise Validations View
Adding a Validation Rule using All Validations View
Validation Rules UI
Formula Validations
Fields Accessible in a Formula
Invocation of Validation Rules

Adding a Validation Rule using *Field-wise Validations View*

1. Select the desired field/section in the message format table.

2. Either click on the **Add New Formula Validation** button ![icon] or Add New Custom Validation button ![icon] in the field validations table based on the type of the validation rule to be specified.

    A new row is added to the field validations table.

3. Change the validation rule name generated by Designer, if required.

4. Based on the type of validation rule specified in the previous step, specify either a formula or a custom field validation class reference in the **Validation** column.

    See the section Defining Validation Classes in the **DefiningCustomClasses.doc** file for more information on defining validation classes.

    Please refer New File from Template for easily creating a custom validation class.

    Pressing the F4 shortcut key from within the **Validation** column displays the **Edit Formula** ![icon] dialog box. See the section Edit Formula Dialog for information on using the **Edit Formula** dialog in specifying a formula.

    See the Fields Accessible in a Formula section for information on the fields that can be used in a formula. See the Invocation of Validation Rules section for considerations involving invocation of validation rules.

    If you have specified a custom field validation class reference name, you have to bind it to the actual class name using the Language Bindings tab of the corresponding Code Generation Settings dialog. See the Code Generation Settings section for more information.

5. In the **Action Message** column, specify the message to be displayed when the validation rule fails.

    It can either be a simple string or an expression that evaluates to a string. The expression can access the fields of the data format to which it belongs just like a formula for a field. Pressing the F4 shortcut key from within the Action Message column displays the Edit Formula ![icon] dialog box. See the Edit Formula Dialog section for information on using the Edit Formula dialog in specifying a formula.

6. In the **Error Code** column, specify the value used in setting the ErrorCode field of TransformException that is generated when the validation fails.

In the validation rules table appearing in Validation Rules UI and Business Exceptions section of Processing Rules UI, the 'Error Code' column is hidden by default, which can be added to the table by right-clicking the table header and selecting the Error Code menu item from the context menu as shown in the following picture.

7. As shown in the following picture, in the **Severity** column, select a severity type from the drop down list displayed when selecting that column.

Validation Rules - SWIFT [MT950]

Header | Data | Trailer

	Name	Validation	Severity	Cascade Errors
🗒 ℹ	C1-1	StartsWith($value, Left(F60a[0].Currency, 2))	error	✔
🗒 ℹ	C1-2	StartsWith($value, Left(F60a[0].Currency, 2))	error	✔
🗒 ℹ	F61-A	$code = F61.Transaction_Type_Identification_Cc	error	✔
🗒 ℹ	F61-B	In(CharAt(F61.Transaction_Type_Identification_'	error	✔
🗒 ℹ	F61-C	If(StartsWith(F61.Transaction_Type_Identificatic	error	✔
🗒 ℹ	F61-D	IsDate(FormatDate(Today(), "yy") + F61.Entry_[	error	✔
🗒 ℹ	DC-FIX	$dc = F61.Debit_Credit_Mark;	error ▼	✔

fatal
error
warn

The severity type specified here would be set as the 'severity' of the exception that is generated in case of validation failure. This is purely for informative purposes and it is not interpreted anywhere else. It helps the user in handling exception in the message flow for taking corrective action based on the severity type.

The 'Severity' column is also hidden by default, which can be added to the table by right-clicking the table header and selecting the Severity menu item from the context menu.

8. If a validation is marked as cascadable (default) by selecting the check box in the **Cascade Errors** column, exceptions are accumulated and subsequent validations are also executed. If a validation rule is non-cascadable, then the validation aborts immediately and the remaining validation rules are not executed. In both cases the validation activity terminates with an exception; the difference is in when it stops.

The Cascade Errors column is also hidden by default, which can be added to the table by right-clicking the table header and selecting the corresponding menu item from the context menu.

9. If a validation rule is disabled, it will not be invoked during runtime. This can be done by deselecting the check box in the **Enabled** column of the corresponding validation rule as shown in the following picture.

The **Enabled** column is also hidden by default, which can be added to the table by right-clicking the table header and selecting the corresponding menu item from the context menu.

See Also:

Adding a Validation Rule using All Validations View
Adding a Validation Rule

Adding a Validation Rule using *All Validations View*

1. Either click on the **Add New Formula Validation** button or the

   Add New Custom Validation button in the field validations table based on the type of the validation rule to be specified.

2. Change the validation rule name generated by Designer, if required.

3. Based on the type of validation rule specified in the previous step, specify either a formula or a custom field validation class reference in the **Validation** column.

   Pressing the F4 shortcut key from within the Validation column displays the Edit Formula dialog box. See the Edit Formula Dialog section for information on using the Edit Formula dialog in specifying a formula.

   See the Fields Accessible in a Formula section for information on the fields that can be used in a formula. See the Invocation of Validation Rules section for considerations involving invocation of validation rules.

   If you have specified a custom field validation class reference name, you have to bind it to the actual class name using the Language Bindings tab of the

corresponding Code Generation Settings dialog. See the Code Generation Settings section for more information.

See the section Defining Validation Classes in the DefiningCustomClasses.doc file for more information.

Please refer New File from Template for easily creating a custom validation class.

4. Specify the field to which the validation rule needs to be applied by selecting a field from the **Applies To** combo box.
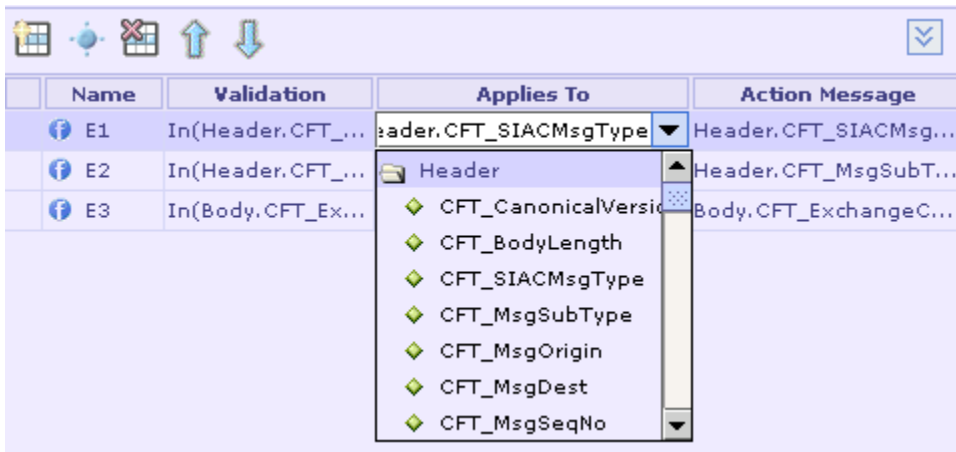
Please note that Designer automatically deduces the 'Applies To' field based on the field/section used in the formula. In case of a formula involving multiple fields/sections, the 'Applies To' field is the common ancestor of those fields/sections.

5. In the **Action Message** column, specify the message to be displayed when the validation rule fails.

It can either be a simple string or an expression that evaluates to a string. The expression can access the fields of the data format to which it belongs just like a formula for a field. Pressing the F4 shortcut key from within the Action Message column displays the **Edit Formula** 🖊 dialog box. See the Edit Formula Dialog section for information on using the Edit Formula dialog in specifying a formula.

6. In the **Error Code** column, specify the value used in setting the ErrorCode field of TransformException that is generated when the validation fails.

In the validation rules table appearing in Validation Rules UI and Business Exceptions section of Processing Rules UI, the 'Error Code' column is hidden by default, which can be added to the table by right-clicking the table header and

selecting the **Error Code** menu item from the context menu as shown in the following picture.

7. As shown in the following picture, in the **Severity** column, select a severity type from the drop down list displayed when selecting that column.

The severity type specified here would be set as the 'severity' of the exception that is generated in case of validation failure. This is purely for informative purposes and it is not interpreted anywhere else. It helps the user in handling exception in the message flow for taking corrective action based on the severity type.

The 'Severity' column is also hidden by default, which can be added to the table by right-clicking the table header and selecting the Severity menu item from the context menu.

8. If a validation is marked as cascadable (default) by selecting the check box in the **Cascade Errors** column, exceptions are accumulated and subsequent validations are also executed. If a validation rule is non-cascadable, then the validation aborts immediately and the remaining validation rules are not executed. In both cases the validation activity terminates with an exception; the difference is in when it stops.

The **Cascade Errors** column is also hidden by default, which can be added to the table by right-clicking the table header and selecting the corresponding menu item from the context menu.

9. If a validation rule is disabled, it will not be invoked during runtime. This can be done by deselecting the check box in the **Enabled** column of the corresponding validation rule as shown in the following picture.

The Enabled column is also hidden by default, which can be added to the table by right-clicking the table header and selecting the corresponding menu item from the context menu.

Formula Validations

The formula feature provided by Designer is quite powerful that most of the validations can be represented by a corresponding formula. It requires nothing less than programming effort to understand the nuances of writing formulae. But once understood, it makes specifying validations easier. The following sections illustrate some of the common validations that can be specified by using formula.

Length Validations

Length validations are used to check whether the number of characters in the value of a string field matches the required length.

Exact Length Validation

This validation is used to check whether the number of characters in the value of a string field is equal to a particular length.

Example

Suppose you want to validate the value of the 'clientId' field in the 'Trade' section to ensure that its length is exactly 10. This can be specified by the following formula:

```
Length(Trade.clientId) == 10
```

The following formula is same as the above if the 'Applies To' column of the row corresponding to the validation rule is 'Trade.clientId'.

```
Length($value) == 10
```

Here, the '$value' variable represents the value of the current field, i.e. the field specified in the 'Applies To' column of the row corresponding to this validation rule.

Minimum Length Validation

This validation is used to check whether the number of characters in the value of a string field is greater than or equals to a particular length.

Example

Suppose you want to validate the value of the 'clientId' field in the 'Trade' section to ensure that its length is at least 10 (equals to 10 or greater than 10). This can be specified by the following formula:

```
Length(Trade.clientId) >= 10
```

Maximum Length Validation

This validation is used to check whether the number of characters in the value of a string field is less than or equals to a particular length.

Example

Suppose you want to validate the value of the 'clientId' field in the 'Trade' section to ensure that its length is at most 10 (equals to 10 or less than 10). This can be specified by the following formula:

```
Length(Trade.clientId) <= 10
```

See Also:

Formula Validations

Range Validations

Range validations can be used to check whether the value of a numeric/date field satisfies the lower and upper bounds of a range.

Exclusive Range Validation

This validation is used to check whether the field value is greater than the lower bound and less than the upper bound of a range (excluding the lower and upper bounds of the range) of acceptable values.

Example 1

Suppose you want to validate the value of the 'SellerDaysCount' field in the 'AdditionalOrderInformation' section to ensure that it is above 0 and below 60. This can be specified by the following formula:

```
AdditionalOrderInformation.SellerDaysCount > 0 &&
AdditionalOrderInformation.SellerDaysCount < 60
```

Inclusive Range Validation

This validation is used to check, in a range of acceptable values, whether the field value is

greater than or equal to the lower bound of that range, and

less than or equal to the upper bound of that range.

Example

Suppose you want to validate the value of the 'SellerDaysCount' field in the 'AdditionalOrderInformation' section to ensure that it is greater than or equals to 0 and it is less than or equals to 60. This can be specified by the following formula:

```
AdditionalOrderInformation.SellerDaysCount >= 0 &&
AdditionalOrderInformation.SellerDaysCount <= 60
```

Note that the above formula can be simplified as given below if the 'Applies To' column of the row corresponding to the validation rule is specified as 'AdditionalOrderInformation.SellerDaysCount'.

```
Between($value, 0, 60)
```

Example 2

Suppose you want to validate the value of the 'InvoiceDate' field to ensure that it lies between May 1, 2003 and April 30, 2004 (including May 1, 2003 and April 30, 2004). This can be specified by the following formula:

```
Between(InvoiceDate, Date(2003, 05, 01), Date(2004, 04, 30))
```

Exclusive Lower Bounded Range Validation

This validation is used to check, in a range with a lower limit and unbounded upper limit, whether the value of a numeric/date field is greater than the lower limit.

Example

Suppose you want to validate the value of the 'SellerDaysCount' field in the 'AdditionalOrderInformation' section to ensure that it is anything above 0. This can be specified by the following formula:

```
AdditionalOrderInformation.SellerDaysCount > 0
```

Inclusive Lower Bounded Range Validation

This validation is used to check, in a range with a lower limit and unbounded upper limit, whether the value of a numeric/date field is greater than or equal to the lower limit.

Example

Suppose you want to validate the value of the 'SellerDaysCount' field in the 'AdditionalOrderInformation' section to ensure that either it is equals to 0 or it is anything above 0. This can be specified by the following formula:

```
AdditionalOrderInformation.SellerDaysCount >= 0
```

Exclusive Upper Bounded Range Validation

This validation is used to check, in a range with unbounded lower limit and an upper limit, whether the value of a numeric/date field is less than the upper limit.

Example

Suppose you want to validate the value of the 'SellerDaysCount' field in the 'AdditionalOrderInformation' section to ensure that it is anything below 60. This can be specified by the following formula:

```
AdditionalOrderInformation.SellerDaysCount < 60
```

Inclusive Upper Bounded Range Validation

This validation is used to check, in a range with unbounded lower limit and an upper limit, whether the value of a numeric/date field is less than or equal to the upper limit.

Example

Suppose you want to validate the value of the 'SellerDaysCount' field in the 'AdditionalOrderInformation' section to ensure that either it is equals to 60 or it is anything below 60. This can be specified by the following formula:

```
AdditionalOrderInformation.SellerDaysCount <= 60
```

See Also:

Formula Validations

Validation for Acceptable Values

This validation is used to check whether the field value is acceptable by comparing it against a list of acceptable values.

Example 1

Suppose you want to validate the value of the 'BranchCode' field to ensure that it is one of the acceptable values "YYY", "ZZZ", "QQQ" and "ZYX". This can be specified by the following formula:

```
In(BranchCode, "YYY", "ZZZ", "QQQ", "ZYX")
```

Example 2

Suppose you want to validate the value of the 'Year' field to ensure that it is one of the acceptable values - 1990, 2000, and 2010. This can be specified by the following formula:

```
In(Year, 1990, 2000, 2010)
```

Example 3

Suppose you want to validate the value of the 'InvoiceDate' field to ensure that it is one of the acceptable dates – May 1, 2004 and May 8, 2004. This can be specified by the following formula:

```
In(InvoiceDate, Date(2004, 5, 1), Date(2004, 5, 8))
```

Pattern Validations

This validation is used to check whether the value of a string field conforms to the exact sequence of characters that are acceptable.

Example

Suppose you want to validate the value of the 'SecurityID' field to ensure the following pattern:

Its first three characters are digits,

followed by a hyphen,

followed by three uppercase/lowercase letters,

followed by a hyphen and

its last three characters are uppercase/lowercase letters or digits.

This can be specified by the following formula:

```
Matches("^\\d{3}-[a-zA-Z]{3}-(\\d|[a-zA-Z]){3}$", SecurityID)
```

Here, the predefined class **\d** is used to match a digit character. Thus the sub-expression **\d{3}** (it is prefixed with the escape character \) matches exactly three digits.

The hyphen character matches the same in the given value.

The **[a-zA-Z]** character class with ranges matches any lowercase/uppercase letter. Thus the sub-expression **[a-zA-Z]{3}** matches exactly three lowercase/uppercase letters.

The character **|** is the logical OR operator. Thus the sub-expression **\d|[a-zA-Z]** matches either a digit or a letter.

The characters **^** and **$** take care of matching at the beginning and end of the value, respectively.

SWIFT Formats Validation

This validation is used to check whether the field value conforms to the exact sequence of characters specified by the SWIFT format.

Example

The format of the 'Identification of Security' field is given as 'ISIN1!e12!c'. This means that the value should consist of

The character sequence 'ISIN' at the beginning

Followed by a space (as denoted by the SWIFT format character 'e') and

Exactly twelve (denoted by the SWIFT format character '!') uppercase letters/digits (as denoted by the SWIFT format character 'c') at the end.

The VerifyFormat() function can be used to validate values against formats based on SWIFT format characters. Thus the 'Identification of Security' field can be validated using the following formula:

```
$secId = B.F35B.Identification_Of_Security;
(Equal(Left($secId, 5), "ISIN ") &&
VerifyFormat(RightStr($secId, "ISIN "), "12!c"))
```

Here, the assignment statement at the beginning of the formula assigns the value of the 'Identification of Security' field (in the 'B.F35B' section) to the variable $secId. This variable is then used in the formula to access the value of the 'Identification of Security' field.

The first five characters at the start of the value is extracted using the Left() function and compared against the string "ISIN " for equality.

The substring that appears after the string "ISIN " is extracted using the RightStr() function and it is verified against the SWIFT format 12!c.

See Also:

Formula Validations

Validation for Missing Field Value

The functions described in the following table help in specifying validations based on missing field values.

Function	Description
IsNotNull	Returns 'true' if the value of the specified field is not missing.
IsNull	Returns 'true' if the value of the specified field is indeed missing.

Note

The IsNull() and IsNotNull() functions should be used only with fields. If you want to check whether a section is empty, (no elements) use "Sec.$size == 0" or one of the **SecExists** functions.

The IsNull() function applied on a section would always return true since the section itself is never null but it can be without elements.

Example

Consider the following message structure:

Field Name	Data Type
StreetSide_Information	Section
StreetSide_Info	String
Additional_StreetSide_Information_Indicator	Character
Additional_StreetSide_Information	Section
Additional_StreetSide_Info	String

It consists of two non-repeating and optional sections – 'StreetSide_Information' and 'Additional_StreetSide_Information'. While the 'StreetSide_Info' field in the 'StreetSide_Information' section is mandatory, the 'Additional_StreetSide_Information_Indicator' in the same section is optional. The 'Additional_StreetSide_Info' field in the 'Additional_StreetSide_Information' section is mandatory.

Now consider the validation requirement:

> If the 'Additional_StreetSide_Information_Indicator' field in the 'StreetSide_Information' section is present, the 'Additional_StreetSide_Information' section should be present in the message.

Otherwise, the 'Additional_StreetSide_Information' section should not be present in the message.

The presence of the 'Additional_StreetSide_Information_Indicator' field in the 'StreetSide_Information' section can be determined by the following formula:

```
IsNotNull(StreetSide_Information.Additional_StreetSide_Information_Indicator
)
```

Here, the IsNotNull() function will return 'true', if the 'Additional_StreetSide_Information_Indicator' field in the only element of the 'StreetSide_Information' section (as it is a non-repeating section) is not missing (contains a value).

The presence of the 'Additional_StreetSide_Information' section can be determined by the following formula:

```
Additional_StreetSide_Information.$size != 0
```

The above formula will return 'true' if the number of elements in the 'Additional_StreetSide_Information' section is not zero, i.e. if the 'Additional_StreetSide_Information' section exists (present) in the message.

Thus, the formula representing the entire validation rule can be written as given below and it can be applied to the 'Additional_StreetSide_Information_Indicator' field.

```
If(IsNotNull(StreetSide_Information.Additional_StreetSide_Information_Indicat
or),
    Additional_StreetSide_Information.$size != 0,
    Additional_StreetSide_Information.$size == 0
)
```

This will validate the presence of the 'Additional_StreetSide_Information' section if the 'Additional_StreetSide_Information_Indicator' field is present in the 'StreetSide_Information' section. But it will not validate the presence of the 'Additional_StreetSide_Information' section if the 'Additional_StreetSide_Information_Indicator' field is missing in the 'StreetSide_Information' section. It is because the validation is not at all invoked when the 'Additional_StreetSide_Information_Indicator' field is missing (is null).

If the above validation is entered at the field level (as in the screen capture given below), the explanation that appears at the top of the validation rules panel makes it clear that this validation is not applied if the 'Additional_StreetSide_Information_Indicator' field is null.

Field Name	Data Type	Description
📁 Data	Section	
📁 StreetSide_Information	Section	
◆ StreetSide_Info	String	
◆ Additional_StreetSide_Information_Indicator	Character	
📁 Additional_StreetSide_Information	Section	
◆ Additional_StreetSide_Info	String	

These validations are applied for every non-null occurence of the field 'Additional_StreetSide_Information_Indicator' in the section 'StreetSide_Information'.

Na...	Type	Validation	Applies To
E1	For...	Additional_StreetSide_Information.$size == 0	StreetSide_Information.Addi...

This validation can be promoted to the 'StreetSide_Information' section (i.e. apply this validation to that section) to ensure that it gets invoked even when the 'Additional_StreetSide_Information_Indicator' field is null.

Even then the validation will not work properly, as it will not be invoked if the 'StreetSide_Information' section itself is missing. Thus, this validation needs to be specified at the parent level of the 'StreetSide_Information' section, i.e. at the message format level. This implies that the 'Applies To' column corresponding to the row containing this validation should be empty.

The validation formula also needs to be changed as given below:

```
If(StreetSide_Information.$size != 0 &&
IsNotNull(StreetSide_Information[0].Additional_StreetSide_Information_Indicato
r),
    Additional_StreetSide_Information.$size != 0,
    Additional_StreetSide_Information.$size == 0
)
```

As the 'StreetSide_Information' section is an optional section, its presence need to be checked by using the following formula:

```
StreetSide_Information.$size != 0
```

As the validation rule is specified at the message format level, only the top-level fields ('StreetSide_Information' and 'Additional_StreetSide_Information' sections) can be directly accessed in the formula. Thus, to access the 'Additional_StreetSide_Information_Indicator' field in the only element (if the previous condition is true, there must be an element in the 'StreetSide_Information' section and as this section is non-repeating it can have only one element) of the 'StreetSide_Information' section, it is specified along with the index 0.

Validations Based on Aggregate Functions

Checking for the Occurrence of a Section

The SecExistsXXX() functions help in checking whether there is the required number of parent section elements with an occurrence of the specified nested section, as described in the following table:

Function	Description
SecExistsInAtLeastOne	Helps in checking whether the specified nested section occurs in at least one element (one or more elements) of the specified parent section.
SecExistsInAtMostOne	Helps in checking whether the specified nested section occurs in at most one element (zero or one element) of the specified parent section.
SecExistsInOne	Helps in checking whether the specified nested section occurs in exactly one element of the specified parent section.
SecExistsInAll	Helps in checking whether the specified nested section occurs in all elements of the specified parent section.

Example

Consider the following network validation rule specified for the SWIFT message MT 564.

If Exchange Rate is present (Field :92B::EXCH), the corresponding Resulting Amount (Field :19A::RESU) must be present in the same (sub)sequence. If the Exchange Rate is not present, the Resulting Amount is not allowed (Error code(s): E62).

This check applies to subsequence E2.

Subsequence E2 if field: 92B::EXCH is…	Subsequence E2 then field :19A::RESU is…
Present	Mandatory
Not present	Not allowed

The phrase "*the same (sub)sequence*" conveys that this validation should be carried out for each instance of subsequence E2 as it occurs. So the 'Applies To' column must be 'E.E2'.

As per the SWIFT specification, the SWIFT field '92B' has a single format option – option B. The format option B consists of the SWIFT qualifier 'EXCH' and the SWIFT subfields – 'First Currency Code', 'Second Currency Code' and 'Rate'.

In Designer, a section represents the SWIFT field '92B' and its sub-section represents the qualifier 'EXCH'. The qualifier section consists of fields representing the SWIFT subfields.

Whether 'Exchange Rate' (Section 92B::EXCH) is present in the current instance of subsequence E.E2 can be determined by the following formula:

```
SecExistsInAtLeastOne(E.E2.F92B, "EXCH")
```

This returns 'true' if an EXCH element occurs (exists) in at least one element of the F92B section (representing SWIFT field 92B) contained in the current E2 element. Anyhow there cannot be more than one F92B element, as the F92B section is non-repeating.

Likewise the presence of 'Resulting Amount' (Section :19A::RESU) in the current instance of subsequence E.E2 can be determined the following formula:

```
SecExistsInAtLeastOne(E.E2.F19A, "RESU")
```

As the presence of the 'Exchange Rate' should be matched by the presence of the 'Resulting Amount', the complete formula is as follows:

```
SecExistsInAtLeastOne(E.E2.F92B, "EXCH") ==
SecExistsInAtLeastOne(E.E2.F19A, "RESU")
```

The 'equals to' operator ensures that the outcome of these two conditions is same (either true or false).

Checking for the Occurrence of a Field

The FieldExistsXXX() functions help in checking whether there is the required number of section elements with an occurrence of the specified field, as described in the following table:

Function	Description
FieldExistsInAtLeastOne	Helps in checking whether the specified field occurs in at least one element (one or more elements) of the specified section.
FieldExistsInAtMostOne	Helps in checking whether the specified field occurs in at most one element (zero or one element) of the specified

section.

FieldExistsInOne	Helps in checking whether the specified field occurs in exactly one element of the specified section.
FieldExistsInAll	Helps in checking whether the specified field occurs in all elements of the specified section.

Example

Consider the following network validation rule specified for the SWIFT message MT 564.

In each occurrence of sequence E, if field :22F::CAOP//OTHR is present, then in the same occurrence of sequence E field :70E::ADTX is mandatory (Error code(s): E79).

In each occurrence of Sequence E if field :22F::CAOP//OTHR is...(*)	In the same occurrence of Sequence E then field :70E::ADTX is...
Present	Mandatory
Not present	Optional

(*) if the Data Source Scheme is present in field :22F::CAOP//OTHR then the conditional rule does not apply.

The phrase "each occurrence of sequence E" conveys that this validation should be invoked for each element of the section E (representing sequence E) as it occurs. So the 'Applies To' column must be 'E'.

As per the SWIFT specification, the SWIFT field '22F' has a single format option – option F. The format option F consists of a SWIFT qualifier and two SWIFT subfields – 'Data Source Scheme' and 'Indicator'. As per the specification, the SWIFT qualifier can be one of the following - 'CAOP', 'DISF', 'OFFE' and 'OPTE'.

In Designer, a section represents the SWIFT field '22F' and each qualifier is represented by its own sub-section. Each qualifier section consists of the 'Data Source Scheme' and 'Indicator' fields representing the SWIFT subfields.

The SWIFT specification also specifies 'OTHR' as one of the codes (among others) to be contained by the SWIFT subfield 'Indicator' when the SWIFT qualifier is 'CAOP' and the 'Data Source Scheme' field is not present. Thus, to check for the presence of the field ':22F::CAOP//OTHR', we need to check whether the value of the field 'Indicator' in the qualifier section 'CAOP' is indeed 'OTHR' and the field 'Data Source Scheme' does not exist in that section. The following formula can be used to check

whether the value of the field 'Indicator' in the qualifier section 'CAOP' (all elements of the section) is indeed 'OTHR':

```
FieldValueExistsInAll(E.F22F, "CAOP.Indicator", "OTHR")
```

This returns 'true' if the 'CAOP.Indicator' field with value 'OTHR' occurs (exists) in all elements of the F22F section contained in the current instance of the sequence E. See the section [Checking for the Occurrence of a Field Value](#) for more information.

Anyhow there cannot be more than one F22F element. Even though the F22F section is shown as repeating (to conform to SWIFT which does not treat Qualifiers as independent fields but as part of the field), internally Designer treats it as non-repeating. As per Designer's definition of sections, there will be only one instance of F22F, and that instance contains one or more of the qualifiers.

The following formula can be used to check the absence of the 'Data Source Scheme' field in the qualifier section 'CAOP':

```
!FieldExistsInAtleastOne(E.F22F, "CAOP.Data_Source_Scheme")
```

This returns 'true' if the 'CAOP.Data_Source_Scheme' field does not occur in any element of the F22F section contained in the current instance of the sequence E.

The presence of field :70E::ADTX (represented by the qualifier section 70E.ADTX) in the same occurrence of sequence E can be determined the following formula:

```
SecExistsInAtleastOne(E.F70E, "ADTX")
```

This returns 'true' if an ADTX element occurs (exists) in at least one element of the F70E section contained in the current instance of sequence E. See the section [Checking for the Occurrence of a Section](#) for more information.

As we need to check for the presence of the qualifier section E.70E.ADTX only when the value of the field 'Indicator' in the qualifier section 'CAOP' is 'OTHR' and the field 'Data Source Scheme' does not exist, the complete formula is as follows:

```
If(FieldValueExistsInAll(E.F22F, "CAOP.Indicator", "OTHR") &&
      !FieldExistsInAtleastOne(E.F22F, "CAOP.Data_Source_Scheme"),
   SecExistsInAtleastOne(E.F70E, "ADTX"),
   true)
```

Checking for the Occurrence of a Field Value

The FieldValueExistsXXX() functions help in checking whether there is the required number of section elements with an occurrence of the specified field with the specified value, as described in the following table:

Function	Description
FieldValueExistsInAtLeastOne	Helps in checking whether the specified field with the specified value occurs in at least one element (one or more elements) of the specified section.
FieldValueExistsInAll	Helps in checking whether the specified field with the specified value occurs in all elements of the specified section.

Example

Consider the following network validation rule specified for the SWIFT message MT 564.

If the safekeeping accounts are not provided, ie, if field :97C::SAFE//GENR is present in any occurrence of subsequence B2, then:

subsequence B2 Account Information must not be repeated in the message
the Balance of Securities, ie, field 93a, must not be present in subsequence B2 Account Information.
subsequence E1 Securities Movement must not be present
subsequence E2 Cash Movement must not be present (Error code(s): E94).

In subsequence B2 if field :97C::SAFE//GENR is...	then subsequence B2 is Mandatory and ...	and in subsequence B2 field 93a is ...	and subsequences E1 and E2 are ...
Present	Not Repetitive min-Max = 1-1	Not allowed	Not allowed
Not present	Repetitive min-Max = 1-n	Optional	Optional and Repetitive min-Max = 0-n

The phrase "*any occurrence of subsequence B2*" means that all subsequence B2 instances that occur in an instance of sequence B should be taken into account for

this validation. It also means that this validation should be carried out for each instance of the sequence B as it occurs. So the 'Applies To' column must be 'B'.

As per the SWIFT specification, the SWIFT field '97a' has two format options – A and C. While the format option A consists of the SWIFT qualifier 'SAFE' and the 'Account Number' subfield, the format option C consists of the SWIFT qualifier 'SAFE' and the 'Account Code' subfield.

In Designer, a section represents the SWIFT field '97a' and its subsection represents the qualifier 'SAFE'. The qualifier section, in turn, consists of 'Account Number' and 'Account Code' fields representing the subfields specified by the format options.

The SWIFT specification also specifies that 'GENR' is the mandatory code for the 'Account Code' subfield. Thus, to check the presence of the field :97C::SAFE//GENR, we need to check whether the value of the 'Account Code' field in the qualifier section 'SAFE' is indeed 'GENR'.

```
FieldValueExistsInAtleastOne(B.B2, "97a.SAFE.Account_Code", "GENR")
```

This returns 'true' if the '97a.SAFE.Account Code' field with value 'GENR' occurs (exists) in at least one element of the B2 section contained in the current instance of the sequence B.

Note that if the field (second argument) is specified in the nested format (as in this case), the FieldValueExistsInAtleastOne() function will iterate through each element of the outermost section for processing. While processing an element of the outermost section, it will also iterate through each element of its immediate nested section, if any, and continues in that order.

In our case both the sections – '97a' and 'SAFE', are non-repeating. So there can be only one element for both the outermost section '97a' and its nested section 'SAFE'.

The presence of the B2 subsequence with exactly one instance can be determined by the following formula:

```
B.B2.$size == 1
```

The $size variable when invoked on a section returns the number of elements in that section. As the above formula invokes it on the 'B.B2' section, it returns the number of elements in it.

The success of the previous condition (B.B2.$size == 1) means that there is exactly one instance for the B2 subsequence. So, to check the absence of the '93a' section (as it is not allowed as per the rule) in the subsequence B2, we need to check that there is no 93a element in the only element (with index 0) of the B2 subsequence. This can be done by the following formula:

```
    B.B2[0].F93a.$size == 0
```

The absence of the 'E1' subsequence can be determined by the following formula:

```
    !SecExistsInAtleastOne(E, "E1")
```

This returns 'true' if there is no instance for the 'E1' section taking into account all instances of the 'E' section.

Likewise, the absence of the 'E2' subsequence can be determined by the following formula:

```
    !SecExistsInAtleastOne(E, "E2")
```

The complete formula representing the given SWIFT network validation rule is as follows:

```
If(FieldValueExistsInAtleastOne(B.B2, "97a.SAFE.Account_Code", "GENR"),
    B.B2.$size == 1 && B.B2[0].F93a.$size == 0 &&
        !SecExistsInAtleastOne(E, "E1") && !SecExistsInAtleastOne(E, "E2"),
    true)
```

We need to check for all the above conditions only when the field :97C::SAFE//GENR is present. If it is not present, none of the conditions need to be checked except the mandatory requirement of subsequence B2. This is automatically taken care of by Designer as part of section validation.

See Also:

[Formula Validations](#)

Fields Accessible in a Formula

A formula validation rule is specified (applied) for a field to validate the value that occurs for that field in the given message.

A field/section accessed in a formula should be either at the same level of the 'Applies To' field or at the top level of the 'Applies To' field.

Consider the following message structure.

The following list explains accessible fields based on the above message structure.

If a validation rule is specified for the top-level field **Company**, it can access the other top-level fields **eMailAddress**, **Region** (not its nested fields) and **MailAddress**. This is allowed because these fields are at the same level of the **Company** field. This is also true for the validation rules specified for the other top-level fields.

If a validation rule is specified for the **RegionName** field, which is a nested field of the **Region** section, it can access the **Branch** section (which is at the same level of the **RegionName** field) and its top-level fields (**Company**, **eMailAddress**, **Region** and **MailAddress**).

See Also:

[Validation Rules UI](#)
[Adding a Validation Rule](#)
[Formula Validations](#)
[Invocation of Validation Rules](#)

Invocation of Validation Rules
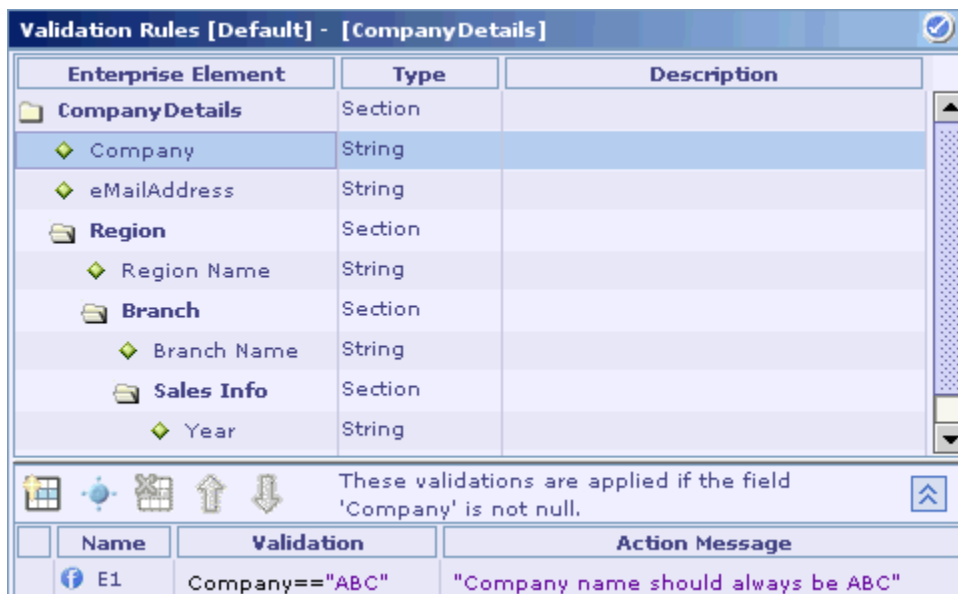
Invocation of a validation rule (either formula or a custom class) involves the following considerations:

The validation rules are executed in 'Applies To' field order. The validation rule(s) associated with fields are executed in the order in which fields are defined. In the diagram below, validations for 'Company' is executed first followed by 'emailAddress'. Then validations for section 'Region' and its children are executed. This is done for every instance of the Region section (and its subsections) before it moves on to next top-level field.

All validations that are not associated with a field (Applies To field is null) are executed before validations associated with a field is executed.

You can safely assume that, before executing the validation for a field, all validations associated with fields above it (in message definition order) have been executed.

Note that, this is not necessarily the order in which validation rules themselves are defined. Because of this, it is important not to have dependencies in the validation rules.

A validation rule specified for a field is invoked for every non-null occurrence of that field in the given message. The position of the field in the data format/message structure (whether it is a top-level field or a nested field) also affects the invocation of a field validation rule.

Consider the following example in which a validation rule is specified for a top-level field.

Enterprise Element	Type	Description
📁 CompanyDetails	Section	
◆ Company	String	
◆ eMailAddress	String	
📂 Region	Section	
◆ Region Name	String	
📂 Branch	Section	
◆ Branch Name	String	
📂 Sales Info	Section	
◆ Year	String	

These validations are applied if the field 'Company' is not null.

	Name	Validation	Action Message
ⓘ	E1	Company=="ABC"	"Company name should always be ABC"

Here, the validation rule is specified for the top-level field **Company**. As a top-level field occurs only once in the actual message, this validation rule is evaluated at most once for a particular message or not even invoked. This is because the validation rule is invoked only if the field value is not null. If the field value is null, the validation rule is not even invoked for that particular message. The explanation given at the top of the validation rule table makes this behavior clear.

Consider the following example in which a validation rule is specified for a nested field.

Validation Rules [Default] - [Internal Format]*		
Enterprise Element	**Type**	**Description**
📁 **Internal Format**	Section	
◆ Company	String	
◆ eMailAddress	String	
📂 **Region**	Section	
◆ RegionName	String	
📂 **Branch**	Section	
◆ BranchName	String	
📂 **Sales Info**	Section	
◆ Year	Integer	
📂 Monthly Sales	Section	

These validations are applied for every non-null occurence of the field 'RegionName' in the section 'Region'.

	Name	Type	Validation
❶	E3	Formula	In(Region.RegionName,"East","West","North","South")

Here, a validation rule is specified for **RegionName**, which is a nested field of the repeating section **Region**. The explanation given at the top of the Validation Rule panel makes it clear that this validation rule is invoked for every non-null occurrence of the **RegionName** field in the **Region** section. This means that if there are four **Region** elements and the **RegionName** field occurs in all of them, then this validation rule is invoked four times (once for each of the four elements).

A validation rule specified for a section is invoked for every occurrence of that section. Similar to field validation rules, the position of the section in the data format/message structure (whether it is a top-level section or a nested section) also affects the invocation of a section validation rule.

Consider the following example in which a validation rule is specified for a top-level section.

Here, the validation rule is specified for the top-level repeating section **Region**. The explanation given at the top of the Validation Rule panel makes it clear that this validation rule is invoked for every element of the **Region** section. As a top-level section, the **Region** section itself can occur only once in the entire message (but it could contain many elements, as it is a repeating section). So if there are four **Region** elements, then this validation rule is invoked four times (once for each of the four **Region** elements).

Consider the following example in which a validation rule is specified for a nested section.
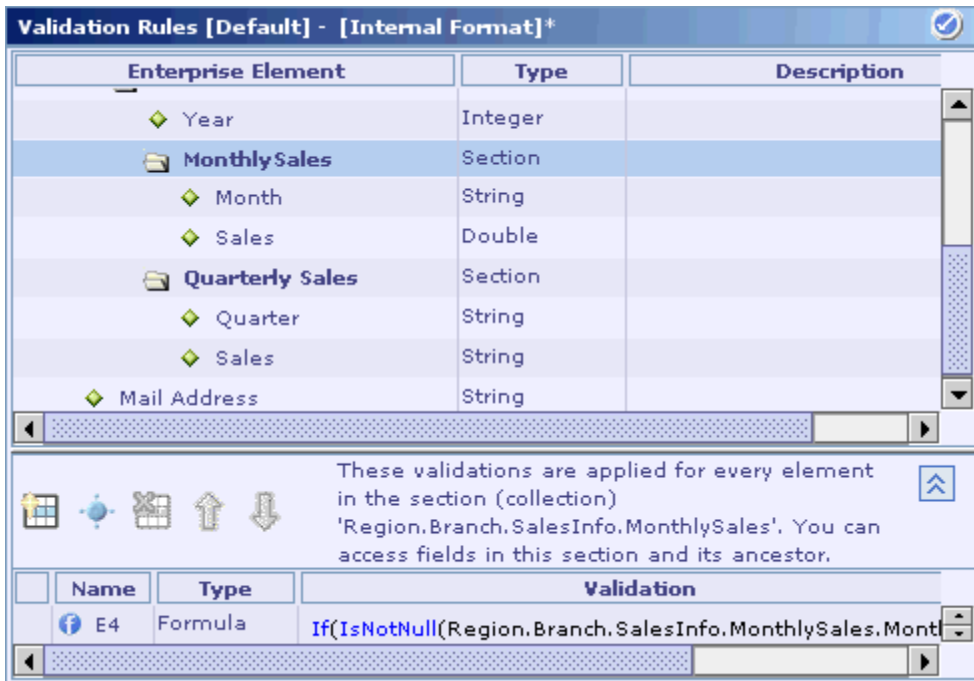
Here, a validation rule is specified for the repeating section **MonthlySales**, which is a nested section of the repeating section **SalesInfo**. The explanation given at the top of the Validation Rule panel makes it clear that this validation rule is invoked for every element in the **Region.Branch.SalesInfo.MonthlySales** section. This means that if there are 12 **MonthlySales** elements in a particular element of the **SalesInfo** section (which in turn depends on the occurrence of the **Region.Branch** section), this validation rule is invoked 12 times during the processing of that particular element of the **SalesInfo** section.

When a validation rule involves multiple fields/sections, its invocation is based on the common ancestor of the fields/sections referred.

Consider the following example in which a validation rule is specified based on two sections.
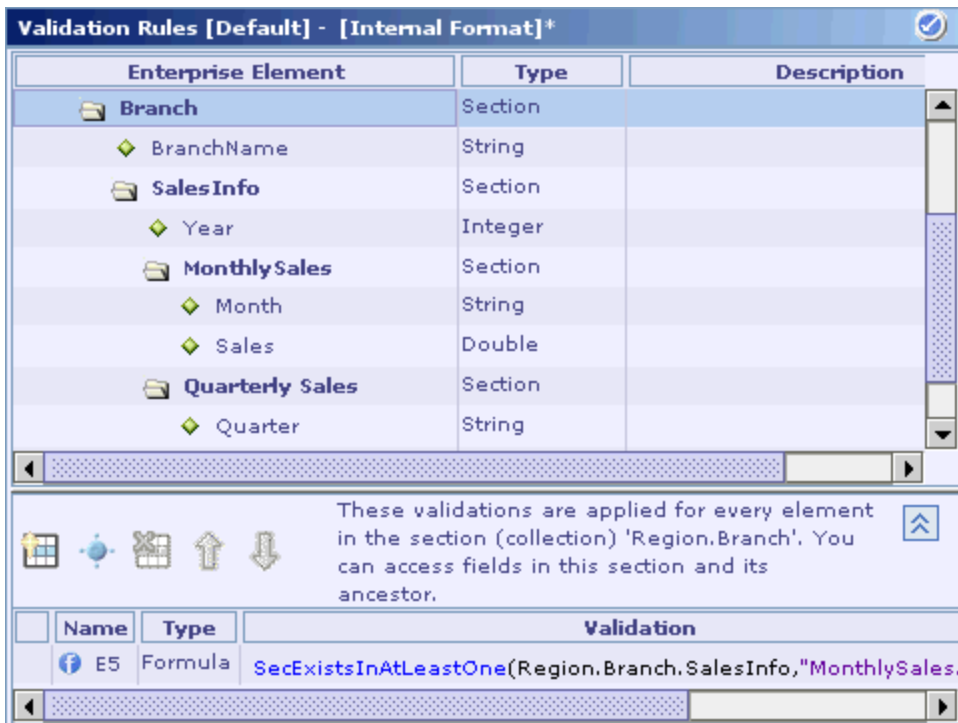
The above picture shows only a part of the validation. The entire validation is given below:

```
SecExistsInAtleastOne(Region.Branch.SalesInfo, "MonthlySales") ==
SecExistsInAtleastOne(Region.Branch.SalesInfo, "QuarterlySales")
```

Here, **Region.Branch** is the applies-to field as it is the common ancestor of the sections referred. So this rule is invoked for each **Region.Branch** instance.

See Also:

Validation Rules UI
Adding a Validation Rule
Formula Validations
Fields Accessible in a Formula

Working with Message - Overview

Once a message has been created, field/section can be added to it in the Message Format UI. The Message Format UI is composed of 'Toolbar', 'FieldsList' and 'PropertiesPanel'.

Toolbar

The tool bar contains the following icons.

Add field/section , icons.	You can use these icons to add a field/section to the message,
Remove field/section icon	You can use this icon to remove a field/section from the message.
Arrange field/section , , , icons.	You can use the arrange icons to move field/section up/down/left/right in a message.
Toggle Tree view icon	You can use the Tree View icon to view the Table contents of a message in a tree structure.
Properties icon	The 'Properties' icon can be used to show or hide the 'Properties' window.
Format Option icon	This icon is used to display the format options that are specific to each format. This button will not be always present in a tool bar. Only messages for which format options can be specified will have this button.

Note:

These are the most common icons that are present in toolbar of all messages. The toolbar may also contain other buttons, which are format specific.

Fields List

	Field Name	Alias	Type	Description
	Records		Section	
	firstname		String	
	lastname		String	Last Name Of Employee
	salary		Double	
	designation		String	

The fields list table displays the fields/section that have been added to the message. The columns in the table are

Name	Name of the field/section. The name specified should be unique (i.e.) no other field/section in the same level should have the same name. This
Alias	Alias name for the field/section.
Type	The data type of the field.
Description	Description for the field/section.

Note:

Depending on the format the table may have additional columns.

Properties Panel

In addition to name/type etc. specified in the fields list table other properties can be specified in the field/section properties panel.

The cardinality for the field and its default value can be specified in the field properties panel. In case of 'Date' fields the date format can be specified in the field properties panel. In section properties panel the cardinality can be specified.

Note:

Depending on the message format additional properties will be displayed in the properties panel.

See Also:

Adding a Field/Section
Field/Section Properties
Removing a Field/Section
Arranging Fields In A Message
Alias
Tracing Messages in a Cartridge to a Standard

Adding a Field/Section

Follow the steps given below to add a new field/section to the internal message format.

1. Select the **Message** design element in the explorer pane.

   The Message UI is shown in the Design Element UI pane.

2. Click on the **Add New Field** button in the toolbar to add a field or the **Add New Section** button to add a section in the Message UI toolbar.

3. A new row appears in the message format table and its 'Name' column is selected by default. If you have selected a row before clicking on the Add New Field button, the new row is inserted immediately after the selected row.
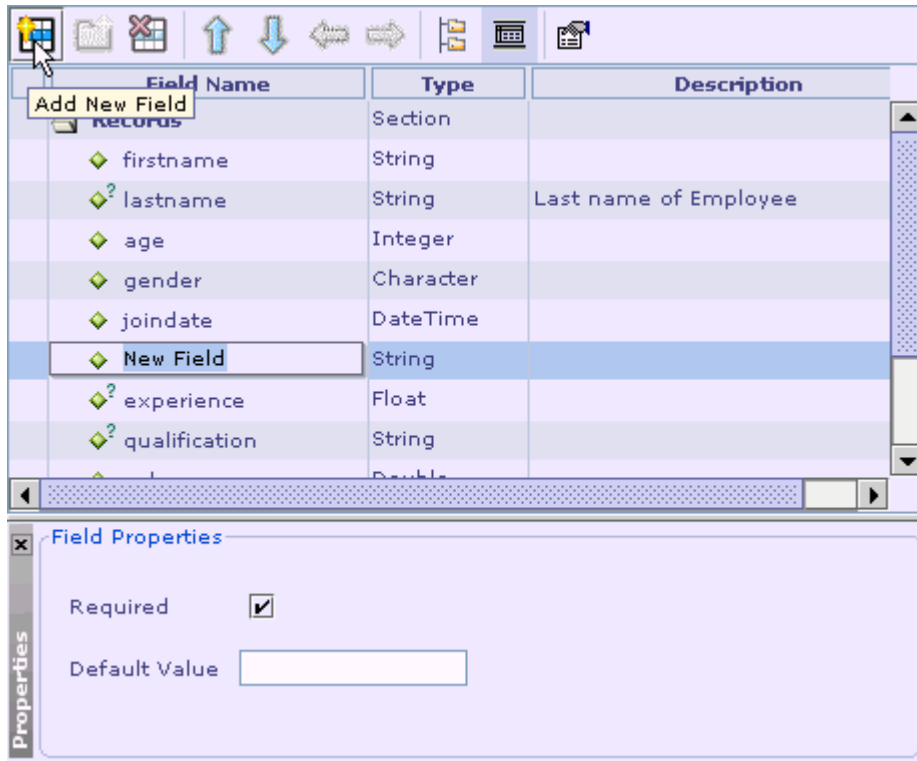
4. Type in a name for the new field/section.

5. Type in the alias name of the new field in the **Alias** column, if required.

6. The Alias column is hidden by default. Select the Alias menu item from the short-cut menu that appears when you right-click on the header row of the message format table.
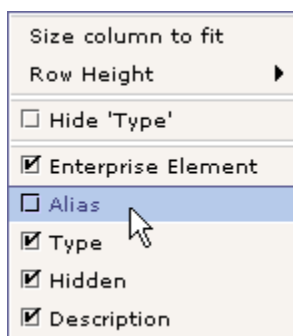
7. See the section Alias for more information on using the alias names.

8. Select the data type of the new field from the list of supported data types that appears in the **Type** column. The supported data types list is format specific and is not the same for all formats.

9. Type in the description of the new field in the **Description** column, if required.

10. Specify the field/section properties. See the section <u>Specifying the Field/Section Properties</u> for more information.

See Also:

<u>Field/Section Properties</u>
<u>Removing a Field/Section</u>
<u>Arranging Fields In A Message</u>

Field/Section Properties

After a field/section has been added properties for it can be specified in the properties panel that is displayed at the bottom of the message when the field/section is selected.

Cardinality

In the field/section **Properties** panel that appears at the bottom of the Message Format UI, cardinality properties like 'required', 'min occurs', 'max occurs' can be specified. Designer indicates an optional field by displaying the ? indicator symbol besides the field icon. No indicator symbol is displayed in case of mandatory fields. In case of sections the cardinality is indicated as shown by the following table

Section Properties

Min Occurs	Max Occurs	Cardinal Indicator	Description
0	Unbounded	*	The section is optional and repeating.
0	1	?	The section is optional and non-repeating.
1	Unbounded	+	The section is non-optional and repeating.
1	1	No indicator	The section is non-optional and non-repeating.

Default Value **Property**

A default value can be specified for a field in the **Default Value** text box of the **Properties** panel. The default value is assigned to that field only if it is not assigned

a value during input mapping. The default value is overwritten if a value is assigned to that field during the internal message processing, which is performed next.

The following points should be noted while assigning a default value for a field.

The data type of the value should match the data type of the field.

While specifying the default value for a **String** type field, the string value should not be enclosed in double quotes.

An expression cannot be used to specify a default value.

Note:

There may also be other properties in the field/section properties panel that are specific to a particular format.

Format **Property**

In case of a field of type 'Date' the format for the field can be specified in the properties panel. A list of pre defined date formats will be listed from which the user can select a format. The user can also specify a new date format for the field.

See Also:

Adding a Field/Section
Removing a Field/Section
Arranging Fields In A Message

Removing a Field/Section

Follow the steps given below to remove a set of fields/sections from the message format.

1. Select the fields/sections to be removed.
    SHIFT-click in the selection column to select a set of continuous fields and CTRL-click in the selection column to select any non-continuous field without affecting the current selection.

2. Click on the Remove Selected Field(s) icon .

Note:

If a section is removed, all of its constituent fields are also removed.

See Also:

Adding a Field/Section
Field/Section Properties

Arranging Fields in a Message

The icons with arrow images in the Message Format UI toolbar can be used to rearrange the position of the fields of a message format.
Once the required fields are selected, the user can click on these icons to arrange the position of these fields within the internal message format. Use the selection column (the empty column at the left) to select a field. To select a set of continuous fields, select the first field in the set and then SHIFT-click on the last field. Use CTRL-clicking to select a non-continuous field. CTRL-clicking on a selected field deselects it.

The following table describes the icons used to arrange the fields of an internal message format:

Icon	Purpose	Description
⬆	Move Selection Up	This moves the selected field(s) up by one position within the same level of message format (within the same section or within the top level). This icon is disabled when the top of that level is reached.
⬇	Move Selection Down	This moves the selected field(s) down by one position within the same level of message format (within the same section or within the top level). This icon is disabled when the bottom of that level is reached.
⬅	Move Selection Left	This promotes the selected field(s) by one level in the message format hierarchy. This means that the selected fields are moved to the same level of their previous parent.
➡	Move Selection Right	This demotes the selected field(s) by one level in the message format hierarchy. This icon is enabled only when the selected fields are positioned immediately after a section, which is at the same level of the selected field(s). Clicking on this icon makes the selected fields children of that section. The fields are positioned at the bottom of that section.

See Also:

Adding a Field/Section
Field/Section Properties

Alias

An alias name is a substitute for the fully qualified name of a field. Designer supports alias names for the following:

1. Fields/Sections of Internal Message
2. Fields/Sections of External Message

Once an internal/external field is assigned an alias name, it can be used wherever you would have used its qualified name, for example, in places like validation/mapping formula. If an internal/external field is assigned an alias name, the qualified name of that field is replaced with its newly assigned alias name in all the places it is used. Even changing/removing the alias name of an internal/external field reflects in all the places of its usage.

See Also:

[Specifying an Alias Name](#)
[Alias Name Substitution](#)
[Alias Name Rules](#)
[Items Supporting Alias Name Substitution](#)
[Internal Message](#)
[External Message](#)
[Validation Rules](#)
[Tracing Messages in a Cartridge to a Standard Message](#)
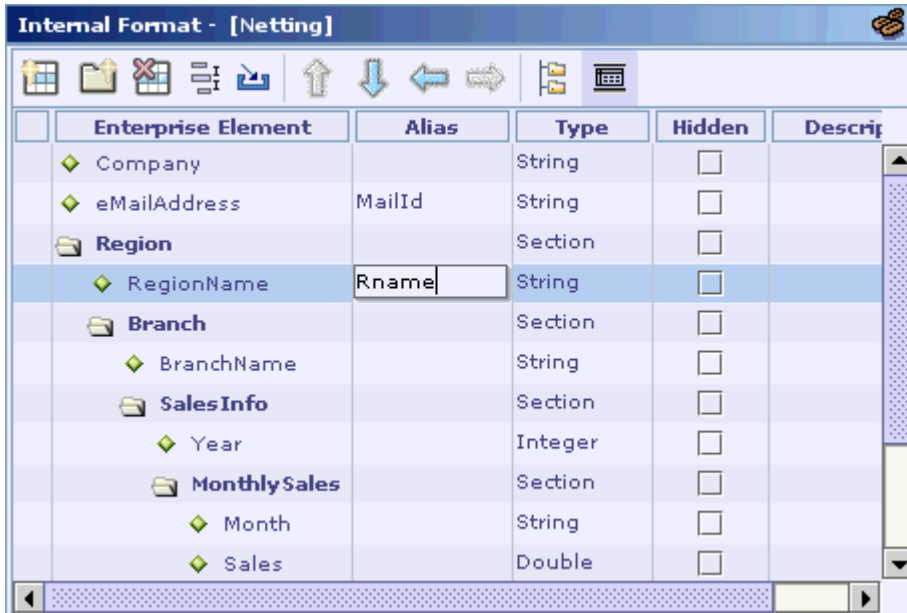
Specifying an Alias Name

An internal field/section can be assigned an alias name using the 'Alias Name' column of the 'Internal Format' table, as shown in the following figure.

Likewise an external field/section can be assigned an alias name using the 'Alias Name' column of the table in the corresponding External Format UI.

See Also:

Alias Name Substitution
Alias Name Rules
Items Supporting Alias Name Substitution
Alias

Alias Name Substitution

Alias name is a substitute for the fully qualified name of a field. Hence, wherever you had earlier used a fully qualified name you can now replace it with its alias name.

Consider the following message structure:

To validate whether the 'RequestedQty' is greater than zero, the validation formula would be something like:

```
GoodsIssueNote.Product.RequestedQty > 0
```

Now, if the 'GoodsIssueNote' field is assigned an alias name 'GIN', the above formula would become:

```
GIN.Product.RequestedQty > 0
```

Now, if the 'Product' field is assigned an alias name 'Item' in addition to the alias name assigned for 'GoodsIssueNote', the above formula would become:

```
Item.RequestedQty > 0
```

Note that once the 'Product' field is assigned an alias name, the alias name of the 'GoodsIssueNote' field becomes insignificant as the alias name stands for 'GoodsIssueNote.Product'.

Now, if the alias name of the 'Product' field is removed, the formula would again become:

```
GIN.Product.RequestedQty > 0
```

The following table lists both valid and invalid substitution of alias names for the qualified names of fields.

No	Qualified Name	Substitute
1	A.B.C	Calias

2	A.B.C	Balias.C
3	A.B.C	Aalias.B.C
4	A.B	A.Balias (Illegal)
5	A.B.C	Aalias.B.Calias (illegal)
6	A.B.C	Balias.Calias (Illegal)

Valid Alias Name Substitutions

1. If the 'GoodsIssueNote.Product.RequestedQty' field is assigned an alias name 'ReqQty', then the formula

   ```
   GoodsIssueNote.Product.RequestedQty > 0
   ```

   could be written as given below:

   ```
   ReqQty > 0
   ```

2. If the 'GoodsIssueNote.Product' section is assigned an alias name 'Item', the formula

   ```
   GoodsIssueNote.Product.RequestedQty > 0
   ```

   Could be written as given below:

   ```
   Item.RequestedQty > 0
   ```

3. If the 'GoodsIssueNote' section is assigned an alias name 'GIN', the formula

   ```
   GoodsIssueNote.Product.RequestedQty > 0
   ```

   Could be written as given below:
   ```
   GIN.Product.RequestedQty > 0
   ```

Invalid Alias Name Substitutions

If the 'GoodsIssueNote.Product' section is assigned an alias name 'Item', then the qualified name 'GoodsIssueNote.Product' cannot be replaced as 'GoodsIssueNote.Item'. This is because the alias name 'Item' stand for 'GoodsIssueNote.Product' that includes the 'GoodsIssueNote' field. Thus the 'GoodsIssueNote' field should not be used again. The same is true for the other invalid alias name substitutions listed in the alias name substitution table.

Alias Name Rules

1. The alias name for all the fields in a format should be unique.

   This means that if one of the fields in a format is assigned an alias name, say 'Broker', then another field of that format cannot be assigned the same name.

2. Alias name for any field in format should not be same as the name of a top-level field.

   This means that if one of the top-level fields in a format is named, say 'Broker', then it cannot be used as an alias name for any other field in that format. It is OK for an alias name to clash with the name of a nested field, since this does not result in any conflict.

3. Alias name should be an identifier with the pattern **[a-z_A-Z][a-z_A-Z0-9]***.

   The following is a list of valid alias names:

   requestedQty
   GIN
   _Product_ID
   Address1

   The following is a list of invalid alias names:

   Product-ID
   1Address
   #dollar

Items Supporting Alias Name Substitution

Alias names can be used in the following places instead of the qualified name of the corresponding field:

1. Internal message processing formula (i.e. formula specified in "Processing Rules -> Formula" column)

2. Internal message validation formula (i.e. formula specified in the "Validation Rules" UI of the corresponding internal message)

3. External message validation formula (i.e. formula specified in the "Validation Rules" UI of the corresponding external message)

4. One-to-one/formula-mapping rules of external messages specified in the "Mapping Rules" UI of the corresponding external message.

5. Validation rules of the internal message mapping node.

6. One-to-one/formula-mapping rules of the internal message mapping node.

Note that when you use the 'Edit Formula' dialog in defining a formula, if you select a field/section that is already assigned an alias name, only its alias name, not the qualified field name, will appear in the formula editor, as shown in the following figure:
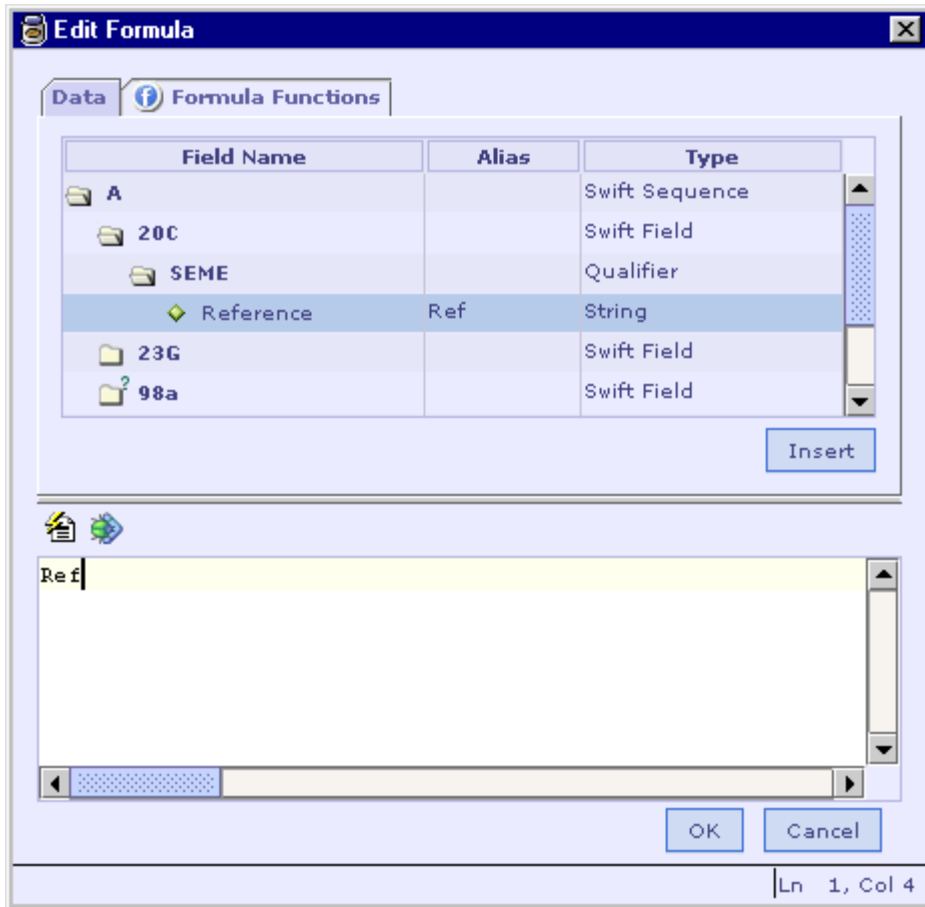
See Also:

[Specifying an Alias Name](#)
[Alias Name Substitution](#)
[Alias Name Rules](#)
[Alias](#)

Tracing Messages in a Cartridge to a Standard

When an internal/external message is added to the cartridge, the information about the standard on which it is based on can be maintained by providing information in the Standard Details section of the internal/external message UI.

This additional information can be used to trace back the message instance in the cartridge to the standard (for instance during integrity verification). For this reason these fields should be edited with utmost care. In fact editing for these fields is allowed so that users can fill up the standard details themselves for existing cartridges.

See Also:

[Internal Message](#)
[External Message](#)
[Validation Rules](#)
[Alias](#)
[Message](#)

Message Mapping

Message mapping is the definition of transforming a message (source message) to another message (target message). In Designer, message mapping can be defined using a custom mapping class and/or mapping rules. If a message mapping is defined using both mapping rules and the custom mapping class, the mapping rules are evaluated before the custom mapping class is invoked.

A mapping rule specifies how a source message field/section is transformed into a target message field/section.

While the section mapping rule (also called as 'section mapping') determines the number of elements to be created for the target section, the field mapping rule (also

called as 'field mapping') determines how a target field is evaluated from source field/fields.

Note:

A field cannot be mapped to a section and vice versa.

Based on the type of source and target messages, the message mappings supported by Designer can be categorized as follows:

1. Input Mapping
2. Output Mapping
3. Internal Message Mapping
4. External Message Mapping

   Here, input mapping is the mapping defined from an external message to an internal message.

   Output mapping is the mapping defined from an internal message to an external message.

   Internal message mapping is the mapping defined from an internal message to another internal message.

   External message mapping is the mapping defined from an external message to another external message.

See Also:
Creating a Message Mapping
Mapping Rules UI
Field Mapping
Section Mapping
Designer User Interface
Cartridge
Message
Formula
Function Definition
Code Generation
Simulator
Working With Cartridge Designer

Creating a Message Mapping

Follow the steps given below to create a new mapping node that can be used to define the transformation between two message formats.

1. In the **Explorer** pane, right-click on the cartridge node or a Folder node to which the new message mapping node needs to be added.

2. Select the New Mapping menu item from the popup menu that appears.

3. The **Message Mapping** dialog box is shown.

4. Select the destination message format from the Destination Message drop down list box.

5. Select the source message format from the Source Message drop down list box.

6. Click on the OK button.

7. The **Mapping** node is created as shown below.
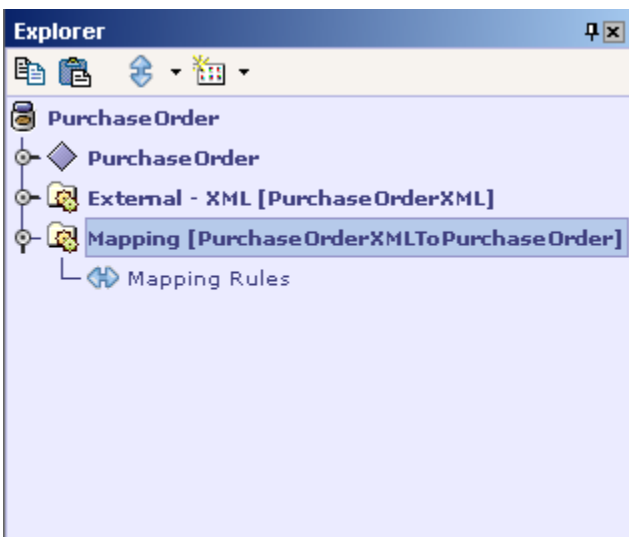
See the section for information on specifying the mapping between two messages.

See Also:

Mapping Rules UI

Defining the mapping from the source message format to the target message format can be done using the Mapping Rules UI shown below.

In the Mapping Rules UI shown above, the target message format is displayed in the table at the top of the UI and the source message format is displayed in the table at the bottom.

The source message format table at the bottom of the Mapping Rules UI can be hidden by clicking on the 'Close' button **x** as shown below.
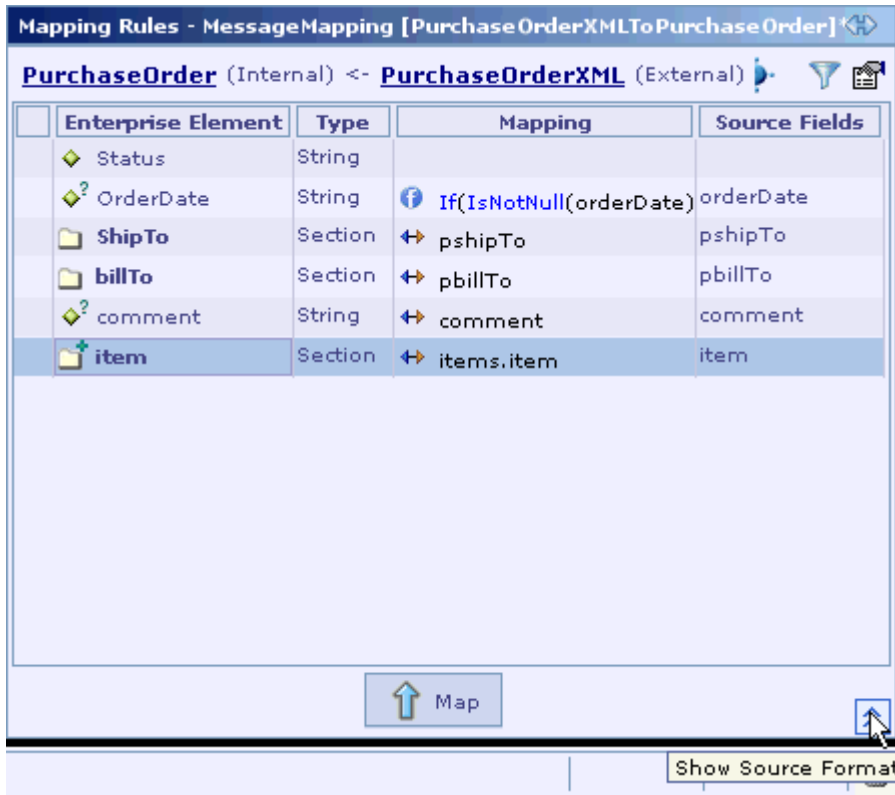
The hidden source message format table can be viewed again by clicking on the 'Show Source Format' button.

While the **Map** button is used in specifying one-to-one mapping rules between the source and target fields, the **Custom Mapping** button is used in specifying a custom mapping class.

The mapping filter at the top right of mapping screen helps user to hide fields that are not used in mapping.

The following table describes sections, which explain how to quickly move between items of Mapping Rules UI and other related items.

Section	Description
Moving from a Field to its Mapping Rule	This section shows you how to quickly move from a field definition to its mapping rule.
Moving from a Field to its Mapping Usage	This section explains how to quickly move from a field definition to its mapping usage, i.e. mapping rule in which this field is used as the source

field.

[Moving Back to a Field Definition](#)	This section describes how to quickly move from a field mapping rule to the field definition.
[Moving Between Source and Destination Fields in Mapping Rules UI](#)	This section describes how to quickly move between source and target fields of a mapping rule.

See Also:
[Adding a Mapping Rule](#)
[Custom Mapping](#)
[Source Field Mapping](#)
[Mapping Filter](#)
[Field Mapping](#)
[Section Mapping](#)
[Creating a Message Mapping](#)
[Message Mapping](#)

Adding a Mapping Rule

There are two types of mapping rules as given below:

1. One-to-One Mapping Rule
2. Formula Mapping Rule

In a one-to-one mapping rule, a target field/section is mapped to a source field/section.

In a formula mapping rule, a target field/section is assigned a formula involving one or more source fields/sections. The formula can also involve fields/sections from the target message format.

See Also:
[Adding a One-to-One Mapping Rule](#)
[Adding a Formula Mapping Rule](#)
[Custom Mapping](#)
[Source Field Mapping](#)
[Mapping Filter](#)
[Field Mapping](#)
[Section Mapping](#)
[Creating a Message Mapping](#)
[Mapping Rules UI](#)
[Message Mapping](#)

Adding a One-to-One Mapping Rule

Follow the steps given below to specify a one-to-one mapping rule:

1. Select the **Mapping Rules** child node of the **Mapping** node.

   The Mapping Rules UI appears in the Design Element UI pane. See the Mapping Rules UI section for more information. It consists of two tables: the top table representing the target message format and the bottom table representing the source message format.

2. In the target message format table at the top of the UI, select the target field/section to be mapped.

3. In the source message format table at the bottom of the UI, select the source message field/section.

4. Click on the Map button to finish mapping.

Please note that the one-to-one mappings are indicated by the ✛ icon. See the Field Mapping and Section Mapping sections for more information on field/section mapping rules.

Adding a Formula Mapping Rule

Follow the steps given below to specify a formula mapping rule:

1. Select the **Mapping Rules** child node of the **Mapping** node.

2. In the target message format table at the top of the UI, double click on the Mapping column of the target message field/section for which mapping needs to be specified.

3. Specify the mapping formula.

   Pressing the F4 key or clicking the Edit Formula icon ⬩ brings up the Edit Formula dialog box that can be used in defining the formula. See the Edit Formula Dialog section for more information on using the Edit Formula dialog for specifying a formula

   Please note that the formula mappings are indicated by the ⓕ icon. See the Field Mapping and Section Mapping sections for more information on field/section mapping rules.

Custom Mapping

The mapping process can be customized, by providing the mapping class that implements the mapping interface corresponding to the type of mapping.

See the section Defining Mapping Classes in the DefiningCustomClasses.doc for information about writing a mapping class and plugging it into an input/output format.

Please refer New File from Template for easily creating a custom mapping class.

Follow the steps given below to specify a custom message mapping class:

1. Select the **Mapping Rules** child node of the **Mapping** node.

2. Click on the **Custom Mapping** icon '' present in the destination format UI. The **Custom Mapping** dialog box is shown.
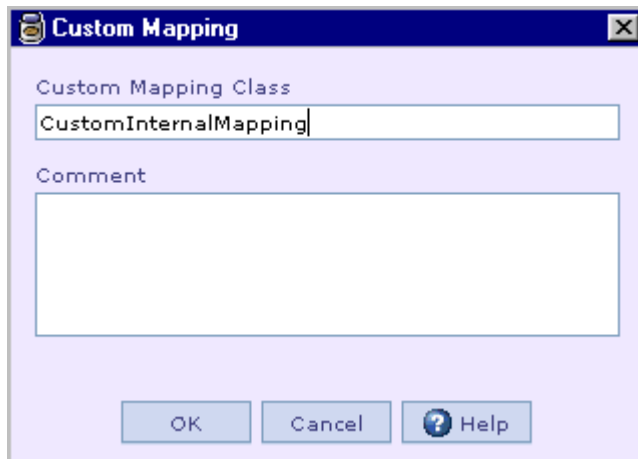
3. Specify the reference name of the custom class in the **Custom Mapping Class** text box.

4. Type in your comment in the **Comment** text area.

5. Click on the **OK** button to finish.

6. Bind the custom class reference name to the actual class name using the **Language Bindings** tab of the corresponding **Code Generation Settings** dialog.

   See the Code Generation section for more information.

See Also:
Adding a Mapping Rule
Source Field Mapping
Mapping Filter
Field Mapping
Section Mapping
Creating a Message Mapping
Mapping Rules UI
Message Mapping

Source Field Mapping

When mapping is done, Designer automatically sets the source fields for the mapped field in the **Source Fields** column. But the user can add/remove source fields for a mapping, using the Source Fields dialog. When error occurs in any one of the source fields specified, the mapped field would be highlighted.

Follow the steps given below to update the source fields list of a mapped field:

1. Select the mapping for a field. Click on the 'Source Fields' column. The column will appear as shown below.
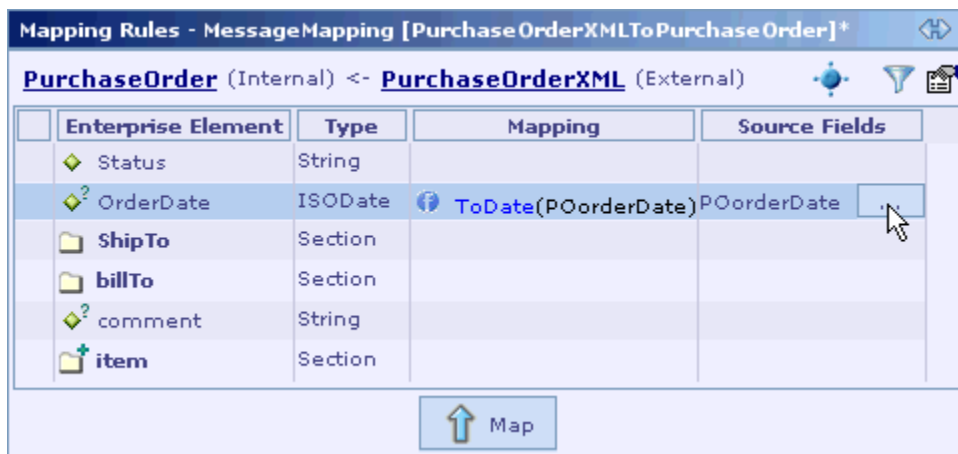
2. Click the ellipsis button displayed in the 'Source Fields' column. The **Source Fields** dialog for managing source fields will be displayed.
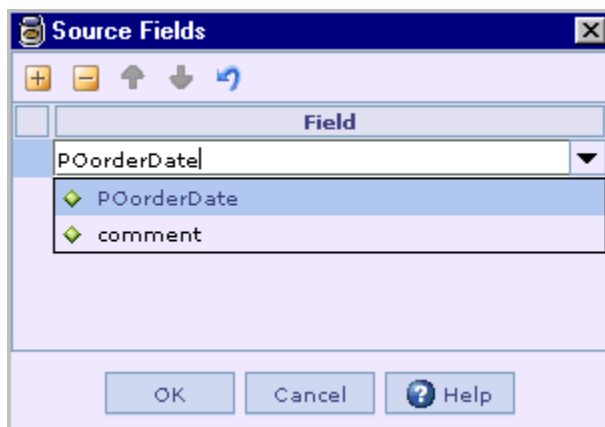
3. You can add source fields by clicking on the **Add** button and selecting the required field from the list of fields displayed or by typing the field name.

4. Likewise you can remove a source field specified for a mapped field by selecting the field to be removed and then clicking on the **Remove** button.

5. Clicking on the **Reset** button will clear the extra source fields added by the user from the list of source fields.

See Also:

Mapping Filter

The mapping filter helps user to hide fields that are not used in mapping. A small toolbar at the top right of mapping screen lets you control the filter. The tool bar is shown below

The filter is off by default, click to toggle it. Enabling it would hide fields that not relevant. What fields are relevant is controlled by the Filtering options you set.

You can specify the filter options by clicking the button. When the button is clicked the following dialog is displayed.
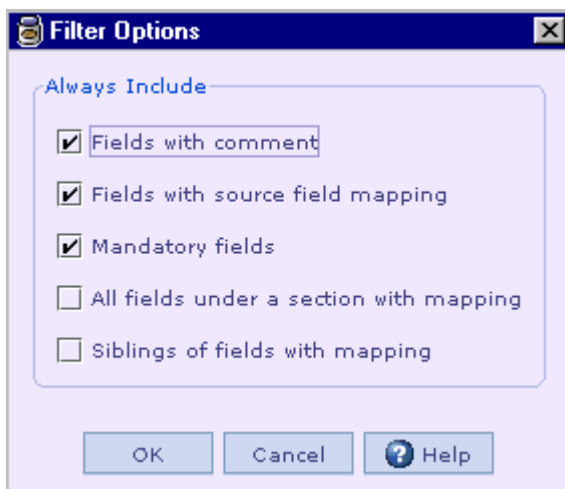
The options provided in the dialog help to display fields that are relevant to the user. One option that is not shown in the dialog is 'Show fields with mapping'. This option

is always true (hence it is not an option). This is because there will not be a case where a user would not be interested in fields with mappings.

Note:

The filter is not very effective when you are starting to enter the mappings. It may hide fields for which you want to specify a mapping. One useful idea would be to add a comment to such fields and use the options to display all fields with comments. Filtering is extremely useful for viewing completed or semi-completed mappings.

When you toggle the filter the current selection is not lost (if it is not filtered out), it is shown at exactly the same place in the screen. This is really useful, when you are entering mappings, since you can quickly toggle between these two modes without losing the current location.

When you are in filtered mode, you should be careful about fields that you see under a section. The section may actually have more fields than what you see. This can be bit confusing at times.

When a field has a mapping, all its ancestor sections are also made visible.

See Also:
Adding a Mapping Rule
Custom Mapping
Source Field Mapping
Field Mapping
Section Mapping
Creating a Message Mapping
Mapping Rules UI
Message Mapping

Field Mapping

Field mappings can be generally categorized as:

1. One-to-One Mapping Rule
2. Formula Mapping Rule

One-to-One Mapping Rule

In a one-to-one mapping rule, a target field is mapped to a source field.

When a target message field is mapped to a source message field, the value of the source field is assigned to the target field. But field mapping considerations such as the level of the source and target fields and the repeating/optional property of their corresponding sections make it more involved, as explained in the following sections.

Formula Mapping Rule

In a formula mapping rule, a target field is assigned a formula involving one or more source fields. The formula can also include fields from the same data format as that of the target field. The only constraint is that the field from the same target data format should have occurred before this field. In the formula, use the syntax **self.Field** to access the field from the same target data format. Pressing the **F4** key or clicking the **Edit Formula** icon 𝒇 from within the **Mapping** column of the Mapping Rules UI brings up the **Edit Formula** dialog box that can be used in defining the formula. See the [Edit Formula Dialog](#) section for more information on using the Edit Formula dialog for specifying a formula.

In a mapping formula, to test the occurrence of an optional field value, use the IsNull() or IsNotNull() functions. To set the null value for an optional field, use the SetNull() function. A sample mapping formula using these functions is given below.

```
If(IsNotNull(Additional_Order_Information.Do_not_Reduce_or_Increase),
set(Additional_Order_Information.Do_not_Reduce_or_Increase),
setNull()
)
```

Here, if the value of the **Additional_Order_Information.Do_not_Reduce_or_Increase** source field is not null, its value is set to the target field for which this formula is specified. Otherwise, the null value is assigned to the target field.

In a mapping formula, to test whether an optional section is empty (there are no occurrences of that section), use the **size** operator as shown in the following expression.

```
Additional_Order_Information.$size == 0
```

This expression will evaluate to true if there are no elements in the **Additional_Order_Information** optional section.

Note

Mapping from a section to a field or from a field to a section is not allowed.

See Also:

Top Level Field to Top Level Field

Source Data Format	Target Data Format	Mapping Allowed
Top-Level Field	Top-Level Field	Yes

Example

Consider the mapping 'EmployeeToEmployeeDetails' defined in TopLevelFieldToTopLevelField.car. The mapping rules specified for the target format fields 'EmployeeName' and 'EmployeeID' are examples of this case.

Target Data Format

Field Name	Type
{} EmployeeName	String
{} EmployeeID	String

Source Data Format

Field Name	Type
{} Name	String
{} ID	String

Mapping

Field Name	Type	Mapping
{} EmployeeName	String	↔ Name
{} EmployeeID	String	↔ ID

Pseudocode

```
destObj.EmployeeName =
srcObj.Name;
destObj.EmployeeID = srcObj.ID;
```

Explanation

The value retrieved from the 'Name' field of the source data object is set to the 'EmployeeName' field of the target data object. It works the same for the 'EmployeeID' field.

See Also:

[Nested Field to Top Level Field](#)
[Top-Level Field to Nested Field](#)
[Nested Field to Nested Field](#)
[Optional Field Mapping](#)
[Mapping involving Fields of Optional Sections](#)
[Merging Fields of Sibling Sections](#)
[Field Mapping](#)

Nested Field to Top Level Field

This can be further categorized into:
[Nested Field of a Repeating Section to Top Level Field](#)
[Nested Field of a Non-repeating Mandatory Section to Top-Level Field](#)
[Nested Field of a Non-repeating Optional Section to Top-Level Field](#)

Nested Field of a Repeating Section to Top Level Field

Source Data Format	Target Data Format	Mapping Allowed
Nested Field of a repeating section	Top-Level Field	No. But, a top-level field can be mapped to a field of a particular section element, using index.

Note

A top-level field can be directly mapped to a nested field of a non-repeating section. See the section [Nested Field of a Non-repeating Mandatory Section to Top-Level Field](#) for more details.

Example

Consider the mapping 'EmployeeToEmployeeDetails' defined in [RepeatingSectionFieldToTopLevelField.car](#). The mapping rules specified for the target data format fields 'TaskID' and 'TaskDescription' are examples of this case.

Target Data Format	
Field Name	**Type**
{} EmployeeName	String
{} TaskID	String
{} TaskDescription	String

Source Data Format	
Field Name	**Type**
{} Name	String
Task	Section
{} ID	String
{} Desc	String

Mapping

Field Name	Type	Mapping
{} EmployeeName	String	↔ Name
{} TaskID	String	Task[0].ID
{} TaskDescription	String	Task[0].Desc

Pseudocode

```
destObj.TaskID = srcObj.Task[0].ID;
destObj.TaskDescription =
srcObj.Task[0].Desc;
```

Explanation

Suppose there are two elements in the 'Task' section of the source data format. Then, there are two 'ID' values; one in element one (index 0) and another in element two (index 1). Using the index, we can assign either the value at index 0 or at index 1. The pseudocode assigns the value at index 0.

This approach is often useful when you know that the section is non-repeating and non-optional (size = 1).

Note

Mappings defined using the indexes are formula mappings.

See Also:

Nested Field to Top Level Field

Nested Field of a Non-repeating Mandatory Section to Top-Level Field

Source Data Format	Target Data Format	Mapping Allowed
Nested Field of a non-repeating	Top-Level Field	Yes

mandatory section		

Example

Consider the mapping 'EmployeeToEmployeeDetails' defined in NonReptMandatorySecFieldToTopLevelField.car. The mapping rules specified for the target data format fields 'DepartmentID' and 'DepartmentLocation' are examples of this case.

Target Data Format

Field Name	Type
() EmployeeName	String
() DepartmentID	String
() DepartmentLocation	String

Source Data Format

Field Name	Type
() Name	String
◰ Dept	Section
() ID	String
() Loc	String

Mapping

Field Name	Type	Mapping
() EmployeeName	String	↔ Name
() DepartmentID	String	↔ Dept.ID
() DepartmentLocation	String	↔ Dept.Loc

Pseudocode

```
destObj.DepartmentID = srcObj.Dept[0].ID;
destObj.DepartmentLocation =
srcObj.Dept[0].Loc;
```

Explanation

As the 'Dept' section in the source data object is a non-repeating mandatory section, it would contain exactly one element. So the generated code assumes index 0 to access the 'Dept' section. So the value of the 'ID' field in the first element of the 'Dept' section (index 0) is assigned to the 'DepartmentID' field of the target data object. Likewise, the value of the 'Loc' source field is assigned to the 'DepartmentLocation' target field.

See Also:

Nested Field to Top Level Field

Nested Field of a Non-repeating Optional Section to Top-Level Field

Source Data Format	Target Data Format	Mapping Allowed
Nested Field of a non-repeating optional section	Top-Level Field	Yes

Example

Consider the mapping 'EmployeeToEmployeeDetails' defined in NonReptOptionalSecFieldToTopLevelField.car. The mapping rules specified for the target data format fields 'Location' and 'CityName' are examples of this case.

Target Data Format

Field Name	Type
{} EmployeeName	String
{}² Location	String
{}² CityName	String

Source Data Format

Field Name	Type
{} Name	String
📁² Address	Section
{} Loc	String
{} City	String

Mapping

Field Name	Type	Mapping
{} EmployeeName	String	↔ Name
{}² Location	String	↔ Address.Loc
{}² CityName	String	↔ Address.City

Pseudocode

```
if (srcObj.Address.getElementCount() > 0)
{
    destObj.Location =
srcObj.Address[0].Loc;
}
if (srcObj.Address.getElementCount() > 0)
{
    destObj.CityName =
srcObj.Address[0].City;
}
```

Explanation

As the 'Address' section of the source data object is an optional non-repeating section, it would contain zero or one element. So the generated code first checks for the presence of the 'Address' element before assigning the value of 'Loc' field to the 'Location' target field. Index 0 is assumed to access the 'Address' section, as it is a non-repeating section. The mapping for the 'CityName' target field is handled in the similar way.

See Also:

Top Level Field to Top Level Field
Top-Level Field to Nested Field
Nested Field to Nested Field
Optional Field Mapping
Mapping involving Fields of Optional Sections
Merging Fields of Sibling Sections
Field Mapping

Top-Level Field to Nested Field

This can be further categorized into:

Top Level Field to Field of a Section without Mapping
Top Level Field to Field of a Section with Mapping

Top Level Field to Field of a Section without Mapping

Source Data Format	Target Data Format	Mapping Allowed

Top-Level Field	Nested Field	Yes

This case assumes that there is no section mapping from any source section to the target section containing the nested field.

Example

Consider the mapping 'EmployeeToEmployeeDetails' defined in TopLevelFieldToFieldofUnmappedSection.car. The mapping rules specified for the nested fields 'ID' and 'Loc' of the 'Dept' section of the target data format are examples of this case.

Target Data Format

Field Name	Type
() Name	String
📋 Dept	Section
() ID	String
() Loc	String

Source Data Format

Field Name	Type
() EmployeeName	String
() DepartmentID	String
() DepartmentLocation	String

Mapping

Field Name	Type	Mapping
() Name	String	↔ EmployeeName
📋 Dept	Section	
() ID	String	↔ DepartmentID
() Loc	String	↔ DepartmentLocation

Pseudocode

```
Dept dept = new Dept();
dept.ID = srcObj.DepartmentID;
dept.Loc =
srcObj.DepartmentLocation;
destObj.Dept.add(dept);
```

Explanation

Since there is only a single value to store, only a single element is created for the 'Dept' section of the target data object (note that any number of elements can be created for a repeating section). The value of the 'DepartmentID' source field is assigned to the 'ID' field of the newly created 'Dept' element. Likewise, the value of the 'DepartmentLocation' source field is assigned to the 'Loc' field of that 'Dept' element. At last, the newly created 'Dept' element is added to the 'Dept' section.

Top Level Field to Field of a Section with Mapping

Source Data Format	Target Data Format	Mapping Allowed
Top-Level Field	Nested Field	Yes

In this case, there is a section mapping from a source section to the target section containing the nested field.

Example

Consider the mapping 'EmployeeToEmployeeDetails' defined in TopLevelFieldToFieldofMappedSection.car. The mapping rules that were specified for the nested field 'ID' of the 'Dept' section are examples of this case.

Target Data Format

Field Name	Type
() Name	String
⌷ **Dept**	Section
() ID	String
() Loc	String

Source Data Format

Field Name	Type
() EmployeeName	String
() DepartmentID	String
⌷ **DepartmentLocation**	Section
() Street	String
() City	String

Mapping

Field Name	Type	Mapping
() Name	String	↔ EmployeeName
⌷ **Dept**	Section	
() ID	String	↔ DepartmentID
() Loc	String	↔ DepartmentLocation.City

Pseudocode

```
    for (int i=0; i < srcObj.DepartmentLocation.length;
++i) {
        Dept dept = new Dept();
        dept.ID = srcObj.DepartmentID;
        dept.Loc = srcObj.DepartmentLocation[i].City;
        destObj.Dept.add(dept);
    }
```

Explanation

Since there is an implicit mapping from the 'DepartmentLocation' section of the source data object to the 'Dept' section of the target data object (see Section Mapping for more information), for every element of the 'DepartmentLocation' section, the corresponding 'Dept' element is created.

The 'City' field value of a 'DepartmentLocation' element is assigned to the 'Loc' field of the corresponding 'Dept' element. The value of 'DepartmentID' source field is assigned to the 'ID' field of each newly created element of the 'Dept' section (*denormalized*).

See Also:

Top Level Field to Top Level Field
Nested Field to Top Level Field
Nested Field to Nested Field
Optional Field Mapping
Mapping involving Fields of Optional Sections
Merging Fields of Sibling Sections
Field Mapping

Nested Field to Nested Field

Source Data Format	Target Data Format	Mapping Allowed
Nested Field	Nested Field	Yes

Example

Consider the mapping 'EmployeeToEmployeeDetails' defined in NestedFieldToNestedField.car. The mapping rules specified for the 'TaskID' and 'TaskDescription' nested fields of the 'Responsibility' section of the target data format are examples of this type.

Target Data Format   **Source Data Format**

Field Name	Type
{} EmployeeName	String
◰⁺ **Responsibility**	Section
{} TaskID	String
{} TaskDescription	String

Field Name	Type
{} Name	String
◰⁺ **Task**	Section
{} ID	String
{} Desc	String

Mapping

Field Name	Type	Mapping
{} EmployeeName	String	↔ Name
◰⁺ **Responsibility**	Section	
{} TaskID	String	↔ Task.ID
{} TaskDescription	String	↔ Task.Desc

Pseudocode

```
for (int i=0; i< srcObj.Task.length; i++) {
    Responsibility responsibility = new
Responsibility();
    responsibility.TaskID = srcObj.Task[i].ID;
    responsibility.TaskDescription =
srcObj.Task[i].Desc;
    destObj.Responsibility.add(responsibility);
}
```

Explanation

For every element of the 'Task' section in the source data object, the corresponding element is created for the 'Responsibility' section of the target data object. Suppose there are two elements in the 'Task' section. The same numbers of 'Responsibility' elements (two) are created. Then, the 'ID' and 'Desc' field values of a 'Task' element are respectively assigned to the 'TaskID' and 'TaskDescription' fields of the corresponding 'Responsibility' element.

Note:

Even though there is no explicit mapping from the 'Task' section of the source data format to the 'Responsibility' section of the target data format, it is assumed. Since the mapping between the 'ID' and 'TaskID' fields is the deepest mapping between these two sections, the mapping between these sections is automatically assumed. See Section Mapping for more information.

Section level mapping between two sections results in the same number of elements in both sections.

Optional Field Mapping

Consider the mapping 'EmployeeToEmployeeDetails' defined in OptionalFieldMapping.car. The mapping rule specified for the 'EmployeeName' field of the target data format is an example of this type.

Target Data Format

Field Name	Type
() EmployeeName	String

Source Data Format

Field Name	Type
() FirstName	String
()² LastName	String

Mapping

Field Name	Type	Mapping
() EmployeeName	String	ⓘ FirstName + If(IsNotNull(LastName), " " + LastName, "")

Explanation

The value of 'EmployeeName' target field is the concatenation of the values of source fields 'FirstName' and 'LastName'. For this requirement, we tend to write the mapping formula as given below:

```
FirstName + " " + LastName
```

But, this would result in the following warning message during validation.

Optional field 'LastName' is accessed without an isNotNull() guard.

But, if you ignore this warning message and proceed with cartridge generation and deployment, at runtime submitting an input message without the optional 'LastName' field would result in 'TransformNullValueException' with the following error message:

Attempt to access field 'LastName' with null value

The reason is that the code generator will not add guard check automatically for the optional field, as it is used in formula mapping. Thus, when an optional field is used in a formula mapping, it is the user's responsibility to add guard check as shown below:

```
FirstName + If(IsNotNull(LastName), " " + LastName, "")
```

See Also:

Top Level Field to Top Level Field
Nested Field to Top Level Field
Top-Level Field to Nested Field
Nested Field to Nested Field
Mapping involving Fields of Optional Sections
Merging Fields of Sibling Sections
Field Mapping

Mapping involving Fields of Optional Sections

Consider the mapping 'OrderInfoToOrder' defined in FieldsofOptionalSectionsMapping.car. The mapping rule specified for the 'OrderQty' field of the target data format is an example of this type.

Target Data Format		**Source Data Format**	
Enterprise Element	**Type**	**Field Name**	**Type**
◆ OrderID	String	⟨⟩ OrderID	String
◆ OrderType	String	⟨⟩ OrderType	String
◆² OrderQty	Integer	Cancel_with_Leaves_Quantity	Section
		⟨⟩ LeaveOrWas	String
		⟨⟩ Leaves_Quantity	Integer
		Basic_Order_Data	Section
		⟨⟩ Quantity	Integer

Explanation

Here, the mapping requirement is as follows: if the 'LeaveOrWas' field of the optional section 'Cancel_with_Leaves_Quantity' is present and its value is "WAS", the value of 'Leaves_Quantity' field of that section should be set to the target field 'OrderQty'; otherwise, the value of the 'Quantity' field of the optional section 'Basic_Order_Data' should be set to that target field.

Consider the mapping formula given below for the above mapping requirement.

```
if(IsNotNull(Cancel_with_Leaves_Quantity.LeaveOrWas)) {
    if(Cancel_with_Leaves_Quantity.LeaveOrWas == "WAS") {
        Set(Cancel_with_Leaves_Quantity.Leaves_Quantity);
    }
}
else {
    Set(Basic_Order_Data.Quantity);
}
```

The mapping is expected to execute even if only one of the sections 'Cancel_with_Leaves_Quantity' and 'Basic_Order_Data' sections is present. But the above mapping will work only if both these sections are present in the message, as explained below.

This mapping accesses fields inside two sections directly; 'Cancel_with_Leaves_Quantity' and 'Basic_Order_Data'. Whenever fields of a non-repeating optional section are accessed directly without indices, then the Designer adds a guard. The Designer assumes that mapping is required only if at least an element exists in all the optional sections whose fields are accessed. For example, consider the simple mapping given below.

```
Set(Basic_Order_Data.Quantity + 10);
```

This mapping is executed only if 'Basic_Order_Data' section has an element. That is the above mapping gets translated to,

```
if(Basic_Order_Data.$size > 0) {
    Set(Basic_Order_Data[0].Quantity + 10);
}
```

Note that it has to access the 'Quantity' field from the first element; hence it ensures that such an element exists. Without the automatic guard, many of normal looking mappings will fail, with runtime exception.

This logic is applied to all the sections that are accessed directly. Hence the formula that is being considered gets converted to,

```
if(Cancel_with_Leaves_Quantity.$size > 0 && Basic_Order_Data.$size > 0)) {
    if(IsNotNull(Cancel_with_Leaves_Quantity[0].LeaveOrWas)) {
        if(Cancel_with_Leaves_Quantity[0].LeaveOrWas == "WAS") {
            Set(Cancel_with_Leaves_Quantity[0].Leaves_Quantity);
        }
    }
    else {
        Set(Basic_Order_Data[0].Quantity);
    }
}
```

This explains why the mapping is never executed when only one of the 'Cancel_with_Leaves_Quantity' and 'Basic_Order_Data' sections is present in the source message. The guard code (added for the entire mapping) prevents it from getting executed if the section elements are missing. So it would be a good idea to use explicit indices when accessing elements from multiple sections.

Mapping

Enterprise El...	Type	Mapping
◆ OrderID	String	↔ OrderID
◆ OrderType	String	↔ OrderType

◆² OrderQty	Integer	ⓘ if(Cancel_with_Leaves_Quantity.$size > 0) {
		        if(Cancel_with_Leaves_Quantity[0].LeaveOrWas == "WAS")
		            Set(Cancel_with_Leaves_Quantity[0].Leaves_Quantity);
		        }
		    }
		    else {
		        if(Basic_Order_Data.$size > 0) {
		            Set(Basic_Order_Data[0].Quantity);
		        }
		    }

The mapping formula given below uses explicit indices and it will work as expected.

```
if(Cancel_with_Leaves_Quantity.$size > 0) {
  if(Cancel_with_Leaves_Quantity[0].LeaveOrWas == "WAS") {
      Set(Cancel_with_Leaves_Quantity[0].Leaves_Quantity);
  }
}
else {
  if(Basic_Order_Data.$size > 0) {
      Set(Basic_Order_Data[0].Quantity);
  }
}
```

See Also:

Merging Fields of Sibling Sections

This can be further categorized into:

Merging Fields of Repeating Sibling Sections

Source Data Format	Target Data Format	Mapping Allowed
Fields of sibling sections (where the sibling sections are repeating sections)	Fields of a single section	No. But can be achieved using a special syntax of mapping explained below. *For this to work, the number of elements in the sibling sections should be same.*

Note:

Mapping from the fields of sibling sections to the fields of a target section is allowed, if only one of the sibling sections is a repeating section and others are non-repeating sections. See Merging Fields of Repeating and Non-Repeating Sibling Sections for more details.

Sibling sections are sections that are children of the same parent section. They can also be top-level sections that are unrelated with each other.

Example

Consider the mapping 'CourseToCourseDetails' defined in MergingRepeatingSiblingSecFields.car. The mapping rules specified for the fields of the 'SubjectInfo' section of the target data format illustrate this case.

Target Data Format   **Source Data Format**

Field Name	Type
⟨⟩ CourseName	String
▤ SubjectInfo	Section
⟨⟩ SubjectCode	String
⟨⟩ PrescribedBook	String

Field Name	Type
⟨⟩ Name	String
▤ Subject	Section
⟨⟩ Name	String
⟨⟩ Code	String
▤ Book	Section
⟨⟩ Name	String
⟨⟩ Author	String

In this case, an attempt is made to map the fields of 'Subject' and 'Book' sections (which are top-level sibling sections) of the source data format to the fields of the 'SubjectInfo' section of the target data format. The simple and usual way of mapping would be:

Target Fields	Source Fields
SubjectInfo.SubjectCode	Subject.Code
SubjectInfo.PrescribedBook	Book.Name

But the above mapping is not allowed.

Why the above mapping is not allowed?

The generated code for the above mapping would be similar to:

```
for(int i=0; i < srcObj.Subject.length; i++) {
    SubjectInfo subjectInfo = new SubjectInfo();
    subjectInfo.SubjectCode =
srcObj.Subject[i].Code;
    //cannot iterate srcObj.Book elements
}
```

The mapping of the field 'Code' of the first sibling section 'Subject' causes a loop to iterate the elements of that section to get the field values, in the generated code. There is no possible way to iterate the elements of 'Book' section within that loop (of 'Subject' section). Because of this constraint mapping of such unrelated sibling sections is not allowed.

How the mapping can be done?
The obvious solution to this problem is to use the same loop of 'Subject' section to iterate 'Book' section elements as well, i.e. use the same index of 'Subject' element

to get the corresponding 'Book' element field values. But this brings in another constraint that 'Subject' and 'Book' sections should have the *same number of elements*. The generated code then, will look similar to:

Pseudocode

```
for(int i=0; i < srcObj.Subject.length; i++) {
    SubjectInfo subjectInfo = new SubjectInfo();
    subjectInfo.SubjectCode =
srcObj.Subject[i].Code;
    subjectInfo.PrescribedBook =
srcObj.Book[i].Name;
    destObj.SubjectInfo.add(subjectInfo);
}
```

Note the same index 'i' is used to access the corresponding elements of 'Subject' and 'Book' elements.

What happens when the number of elements in the sibling sections is different?

Suppose there are 3 'Subject' elements and 2 'Book' elements in the source message. The above mapping would create two 'SubjectInfo' elements based on the first two elements (with index 0 and 1) of 'Subject' and 'Book' sections. As the third element is available only in 'Subject' section but not in 'Book' section ('Book' section has only two elements), it would result in runtime error. Hence during runtime, the sibling sections 'Subject' and 'Book' should have the same number of elements.

Mapping in Designer

The solution for this mapping is to use the same index to access the corresponding elements of the sibling sections 'Subject' and 'Book', as mentioned above. The mapping for the 'PrescribedBook' field of the 'SubjectInfo' section of the target data format shows how to use the $index variable to obtain the index of the element being processed (see figure below).

Mapping

Field Name	Type	Mapping
() CourseName	String	↔ Name
📑 SubjectInfo	Section	
() SubjectCode	String	↔ Subject.Code
() PrescribedBook	String	🔵 Book[Subject.$index].Name

As per the above mapping, for each element of the 'Subject' section in the source data object, a corresponding 'SubjectInfo' element is created in the target data object (note that section level mapping between two sections results in the same number of elements in both sections). Then the value of the 'Code' field of the 'Subject' element is assigned to the 'SubjectCode' field of the corresponding 'SubjectInfo' element.

The index of the current 'Subject' element can be obtained using $index as shown below:

```
Subject.$index
```

This $index variable is used to access the value of 'Name' field of the 'Book' element (whose index is same as that of the current 'Subject' element) and it is assigned to the 'PrescribedBook' field of the 'SubjectInfo' element.

See Also:

[Merging Fields of Sibling Sections](#)

Merging Fields of Repeating and Non-Repeating Sibling Sections

Source Data Format	Target Data Format	Mapping Allowed
Fields of sibling sections (where only one of the sibling sections is a repeating section and others are non-repeating sections)	Fields of a single section	Yes

Sibling sections are sections that are children of the same parent section. They can also be top-level sections that are unrelated with each other.

Example

Consider the mapping 'CourseToCourseDetails' defined in [JustOneofSiblingSectionsIsRepeating.car](#). The mapping rules specified for the fields of the 'SubjectInfo' section of the target message format illustrate this case.

In this case, an attempt is made to map the fields of 'Subject' and 'CoachingCenter' sections (which are top-level sibling sections) of the source data format to the fields of the 'SubjectInfo' section of the target data format. While the first sibling section 'Subject' is a repeating section, the second sibling section 'CoachingCenter' is a non-repeating section.

	Target Data Format	Source Data Format

Target Data Format

Field Name	Type
() CourseName	String
SubjectInfo	Section
() SubjectCode	String
() CoachingCenterCode	String

Source Data Format

Field Name	Type
() Name	String
Subject	Section
() Name	String
() Code	String
CoachingCenter	Section
() Code	String
() Address	String

Mapping

Field Name	Type	Mapping
() CourseName	String	↔ Name
SubjectInfo	Section	
() SubjectCode	String	↔ Subject.Code
() CoachingCenterCode	String	↔ CoachingCenter.Code

Pseudocode

```
for(int i=0; i < srcObj. Subject.length; i++) {
    SubjectInfo subjectInfo = new SubjectInfo();
    subjectInfo.SubjectCode = srcObj.Subject[i].Code;
    subjectInfo.CoachingCenterCode =
srcObj.CoachingCenter[0].Code;
    destObj.SubjectInfo.add(subjectInfo);
}
```

Explanation

For each element in the source section 'Subject', the corresponding 'SubjectInfo' element is created and the value of the 'Code' field of a 'Subject' element is assigned to the 'SubjectCode' field of the corresponding 'SubjectInfo' element. The value of 'Code' field in the only element of the 'CoachingCenter' section is assigned to the 'CoachingCenterCode' field of each element created for the target section 'SubjectInfo'.

See Also:

Top Level Field to Top Level Field
Nested Field to Top Level Field
Top-Level Field to Nested Field
Nested Field to Nested Field
Optional Field Mapping

Section Mapping

Normally, designer figures out section mapping automatically by looking at field mapping. It uses section mapping to create the correct number of section elements in the output. The user can map sections if a section has no fields, but has subsections (in this case designer cannot figure out how many elements need to be created). If the user does not provide a section mapping, it creates just one element.

There is no harm in mapping sections even if the fields have mapping. Designer ensures that the section mapping is consistent with the field mappings (extra validation).

It is important to remember that field mapping is necessary (if the field is needed at the output) even when section mapping is provided.

How is section mapping determined?

A section, SecOut, in the output is mapped to a section, SecIn, in the input if,

> the two sections are explicitly mapped by the user,
>
> a field in the output section is mapped to a field in the input section and no other field in *SecOut* is mapped to a field belonging to a descendent section of *SecIn*. That is the mapping for an output section is determined by the deepest mapping of its fields.
>
> The following restrictions apply.
>
> The section mapping determined by each of the fields of the output section must be in the same hierarchy (they should have parent–child-child-… relationship). The actual mapping is the deepest section in the hierarchy. This is why mapping to fields of sibling sections in the input is not allowed (Case 8A).
>
> The user specified mapping for a section couldn't be weaker than the mapping determined by its fields. This means that if the deepest mapping determined by the fields of the section SecOut is SecIn, then it is an error to explicitly map SecOut to a parent section of (or a section unrelated to) SecIn. It is acceptable to map it to a child of SecIn.

Top-Level Section to Nested Section

Source Data Format	Target Data Format	Mapping Allowed
Top-Level Section	Nested Section	Yes

Example

Consider the mapping 'SalesToSalesDetails' defined in [TopLevelSectionToNestedSection.car](#). The mapping specified for the 'ProductInfo' nested section of the 'YearlySales' target data format section is an example of this case.

Target Data Format

Field Name	Type
YearlySales	Section
SalesYear	Integer
ProductInfo	Section
BrandInfo	Section
BrandName	String
SalesQty	Integer

Source Data Format

Field Name	Type
Year	Integer
Product	Section
Brand	Section
Name	String
Qty	Integer

Mapping

Field Name	Type	Mapping
YearlySales	Section	
SalesYear	Integer	↔ Year
ProductInfo	Section	↔ Product
BrandInfo	Section	↔ Product.Brand
BrandName	String	↔ Product.Brand.Name
SalesQty	Integer	↔ Product.Brand.Qty

Pseudocode

```
    for(int j=0; j < srcObj.Product.length; ++j) {
        ProductInfo productInfo = new ProductInfo();
        for(int k=0; k < srcObj.Product.Brand.length; ++k) {
            BrandInfo brandInfo = new BrandInfo();
            brandInfo.BrandName =
    srcObj.Product[j].Brand[k].Name;
            brandInfo.SalesQty =
    srcObj.Product[j].Brand[k].Qty;
            productInfo.BrandInfo.add(brandInfo);
        }
        destObj.ProductInfo.add(productInfo);
    }
```

Explanation

Suppose there are two 'Product' elements in the input object and there are two 'Brand' elements in the first 'Product' element and one 'Brand' element in the second. Note that section level mapping between two sections results in the same number of elements in both sections. So, two 'ProductInfo' elements are created in the only element of the 'YearlySales' section (This is because of the mapping from the top-level field 'Year' in source data format to the nested field 'SalesYear' in target data format). For further details, refer to the section Top Level Field To Field of a Section Without Mapping. The mapping of the 'BrandInfo' section of the target data format to the 'Brand' section of the source data format results in the same number of 'BrandInfo' elements to be created in a 'ProductInfo' element as that of the corresponding 'Product' element. So, two 'BrandInfo' elements are created for the first 'ProductInfo' element of the target data object and one 'BrandInfo' element is created for the second 'ProductInfo' element (same as in the input).

See Also:

Nested Section to Top-Level Section
Nested Section to Nested Section
Optional Section Mapping
Section Formula Mapping
Section Mapping

Nested Section to Top-Level Section

Source Data Format	Target Data Format	Mapping Allowed
Nested Section	Top-Level Section	Yes

Example

Consider the mapping 'SalesToSalesDetails' defined in NestedSectionToTopLevelSection.car. The mapping specified for the 'ProductInfo' target section is an example of this case.

Target Data Format

Field Name	Type
YearlySales	Section
SalesYear	Integer
ProductInfo	Section
BrandInfo	Section
BrandName	String
SalesQty	Integer

Source Data Format

Field Name	Type
Year	Integer
Product	Section
Brand	Section
Name	String
Qty	Integer

Mapping

Field Name	Type	Mapping
SalesYear	Integer	YearlySales[0].Year
ProductInfo	Section	YearlySales.Product
BrandInfo	Section	YearlySales.Product.Brand
BrandName	String	YearlySales.Product.Brand.Name
SalesQty	Integer	YearlySales.Product.Brand.Qty

Pseudocode

```
for(int i=0; i < srcObj.YearlySales.length; ++i) {
  for(int j=0; j < srcObj.YearlySales.Product.length; ++j) {
    ProductInfo productInfo = new ProductInfo();
    for(int k=0; k < srcObj.YearlySales.Product.Brand.length; ++k) {
        BrandInfo brandInfo = new BrandInfo();
        brandInfo.BrandName =
srcObj.YearlySales[i].Product[j].Brand[k].Name;
        brandInfo.SalesQty = srcObj.YearlySales[i].Product[j].Brand[k].Qty;
        productInfo.BrandInfo.add(brandInfo);
    }
    destObj.ProductInfo.add(productInfo);
  }
}
```

Explanation

Suppose there is a single 'YearlySales' element in the source data object that contains two 'Product' elements with two 'Brand' elements in the first 'Product' element and one 'Brand' element in the second. Since the 'ProductInfo' section of the target data format is mapped to the 'Product' source section, for every 'Product'

element within every 'YearlySales' element, the corresponding 'ProductInfo' element is created in the target data object. In our case, two 'ProductInfo' elements are created in the target data object. The mapping of the 'BrandInfo' target section to the 'Brand' source section, results in the same number of 'BrandInfo' elements to be created in a 'ProductInfo' element as that of the corresponding 'Product' element. So there will be two 'BrandInfo' elements in the first 'ProductInfo' element of the target data object and one 'BrandInfo' element in the second 'ProductInfo' element.

Consider the following input:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Sales SYSTEM "Sales.dtd">
<Sales>
    <YearlySales>
        <Year>2003</Year>
        <Product>
            <Brand>
                <Name>HP</Name>
                <Qty>1000</Qty>
            </Brand>
            <Brand>
                <Name>Philips</Name>
                <Qty>1500</Qty>
            </Brand>
        </Product>
        <Product>
            <Brand>
                <Name>BPL</Name>
                <Qty>1200</Qty>
            </Brand>
        </Product>
    </YearlySales>
</Sales>
```

The output in this case (when the sections are mapped) is as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE SalesDetails SYSTEM "SalesDetails.dtd">
<SalesDetails>
    <SalesYear>2003</SalesYear>
    <ProductInfo>
        <BrandInfo>
            <BrandName>HP</BrandName>
            <SalesQty>1000</SalesQty>
        </BrandInfo>
        <BrandInfo>
            <BrandName>Philips</BrandName>
            <SalesQty>1500</SalesQty>
        </BrandInfo>
    </ProductInfo>
    <ProductInfo>
        <BrandInfo>
            <BrandName>BPL</BrandName>
            <SalesQty>1200</SalesQty>
        </BrandInfo>
    </ProductInfo>
</SalesDetails>
```

See Also:
Top-Level Section to Nested Section
Nested Section to Nested Section
Optional Section Mapping
Section Formula Mapping
Section Mapping

Nested Section to Nested Section

Consider the mapping 'SalesToSalesDetails' defined in
NestedSectionToNestedSection.car. This is same as same as Nested Section to
Top-Level Section example but with a difference that there is no mapping for the
'ProductInfo' output section as shown in the figure given below:

Mapping

Field Name	Type	Mapping
() SalesYear	Integer	🛈 YearlySales[0].Year
🗐 ProductInfo	Section	
🗐 BrandInfo	Section	↔ YearlySales.Product.Brand
() BrandName	String	↔ YearlySales.Product.Brand.Name
() SalesQty	Integer	↔ YearlySales.Product.Brand.Qty

Pseudocode

```
ProductInfo productInfo = new ProductInfo(); //Note the placement of this
stmt
for(int i=0; i < srcObj.YearlySales.length; ++i) {
  for(int j=0; j < srcObj.YearlySales.Product.length; ++j) {
    for(int k=0; k < srcObj.YearlySales.Product.Brand.length; ++k) {
        BrandInfo brandInfo = new BrandInfo();
        brandInfo.BrandName =
srcObj.YearlySales[i].Product[j].Brand[k].Name;
        brandInfo.SalesQty =
srcObj.YearlySales[i].Product[j].Brand[k].Qty;
        productInfo.BrandInfo.add(brandInfo);
    }
  }
}
destObj.ProductInfo.add(productInfo);
```

Explanation

Since there is no mapping for the 'ProductInfo' section of the target data format, only one 'ProductInfo' element is created in the target data object, for all 'Product' elements that occur within the 'YearlySales' section of the source data object. The mapping of 'BrandInfo' section of the target data object is handled as in Nested Section to Top-Level Section.

Source Data Object:

No. of 'YearlySales' elements	No. of 'Product' elements	No. of 'Brand' elements
2	2	2
		3
	1	4

Target Data Object:

No. of 'ProductInfo' elements	No. of 'BrandInfo' elements
1	9

The output in this case (for the input in <u>Nested Section to Top-Level Section</u>) is as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE SalesDetails SYSTEM "SalesDetails.dtd">
<SalesDetails>
    <SalesYear>2003</SalesYear>
    <ProductInfo>
        <BrandInfo>
            <BrandName>HP</BrandName>
            <SalesQty>1000</SalesQty>
        </BrandInfo>
        <BrandInfo>
            <BrandName>Philips</BrandName>
            <SalesQty>1500</SalesQty>
        </BrandInfo>
        <BrandInfo>
            <BrandName>BPL</BrandName>
            <SalesQty>1200</SalesQty>
        </BrandInfo>
    </ProductInfo>
</SalesDetails>
```

Though section mapping is not a must, it is done implicitly when the fields are mapped. But in certain cases like this, section mapping needs to be done to get the output in the required format.

See Also:

Optional Section Mapping

Consider the mapping 'PayrollXMLToPayroll' defined in OptionalSectionMapping.car. The mapping rule specified for the 'Deduction' field of the target data format is an example of this case.

Target Data Format		**Source Data Format**	

Enterprise Element	Type
📄 **PayDetails**	Section
◆ EmpID	String
◆ BasicPay	Double
◆² Allowance	Double
◆² Deduction	Double

Field Name	Type
📄 **Employee**	Section
() ID	String
() Basic	Double
()² HRA	Integer
📄 **Deductions**	Section
() PF	Integer
() IT	Integer

Mapping

Enterprise Ele...	Type	Mapping
📄 **PayDetails**	Section	
◆ EmpID	String	↔ Employee.ID
◆ BasicPay	Double	↔ Employee.Basic
◆² Allowance	Double	🔵 If(IsNotNull(Employee.HRA),
◆² Deduction	Double	🔵 If(Employee.Deductions.$size > 0, Set((Employee.Deductions[0].PF / 100.0) * Employee.Basic + (Employee.Deductions[0].IT / 100.0) * Employee.Basic))

Explanation

To calculate the value of the 'Deduction' target field based on the values (given in percentage of basic pay) of 'PF' and 'IT' fields of the optional source section 'Deductions', we tend to write the mapping formula as given below:

```
(Employee.Deductions[0].PF / 100.0) * Employee.Basic +
   (Employee.Deductions[0].IT / 100.0) * Employee.Basic
```

But, this would result in FieldNotFoundException with the following error message during runtime if the 'Deductions' section is missing in the input (which is valid as it is an optional section).

Error accessing element of section 'Employee.Deductions'. Section index out of bounds, size = '0' , index accessed = '0'.

When the user uses a field from an optional section in a formula, to avoid runtime exception, the code generator adds guard code around optional section as explained in [Nested Field of a Non-repeating Optional Section to Top-Level Field](). However, if the user specifies an index (in this case 0), then it is the user's responsibility to specify the guard and code generator will not add guard check automatically. Thus, the above mapping rule should be written as shown below. Here **Employee.Deductions.$size** returns the number of elements in the 'Employee.Deductions' section.

```
If(Employee.Deductions.$size > 0,
   Set((Employee.Deductions[0].PF / 100.0) * Employee.Basic +
       (Employee.Deductions[0].IT / 100.0) *
         Employee.Basic))
```

See Also:

[Section Mapping]()

Section Formula Mapping

Consider the [SectionFormulaMapping.car]() cartridge. Here the mapping requirement is that the input fields F1, F2 and F3 (see the pictures given below) need to be mapped to the field 'Fld' of the internal message section 'Sec' as shown below:

F1	->	Sec[0].Fld
F2	->	Sec[1].Fld
F3	->	Sec[2].Fld

Here, all the source fields are optional. In the destination we need to create as many elements of 'Sec' as required.

Target Data Format (NO Fields)

Enterprise Element	Type
Sec	Section
Fld	String

Source Data Format(Input Fields)

Field Name	Type
F1	String
F2	String
F3	String

This cannot be achieved by using normal mapping at it requires control over creation of the target section elements as well as assignment of values to the fields of the target section elements created.

This can be achieved by specifying a mapping formula for the section 'Sec' as shown in the following picture.

Input Mapping

Enterprise Element	Type	Mapping

Sec	Section	`if(IsNotNull(F1)) {`
		    `def elm = SecAddElement(self.Sec);`
		    `elm.Fld = F1;`
		`}`
		`if(IsNotNull(F2)) {`
		    `def elm = SecAddElement(self.Sec);`
		    `elm.Fld = F2;`
		`}`
		`if(IsNotNull(F3)) {`
		    `def elm = SecAddElement(self.Sec);`
		    `elm.Fld = F3;`
		`}`

Fld	String	

Note that this formula does not return anything. This is taken as a clue that user has taken charge of the section mapping. Also, note that there is no mapping for the field 'Fld'.

The formula creates section elements based on the occurrences of fields F1, F2, and F3 and sets the value of sub field 'Fld'. For instance, if field F2 is missing, then only two elements are created.

If the target format has other fields/sections, the user can provide a mapping for them. The only restriction is that the user cannot provide a mapping for fields/sections under this section (for which a void returning formula has been specified).

See Also:
Top-Level Section to Nested Section
Nested Section to Top-Level Section
Nested Section to Nested Section
Optional Section Mapping
Section Mapping

Formula

A formula is an expression that evaluates to a value. A formula expression can refer to one or more data items depending on the context in which they are used. The value returned by a formula expression is used for setting/validating fields. These formula expressions have Java like expression syntax and can access a number of predefined functions available in Designer.

See Also:

Entering a Formula
Editing a Formula
Reformat Formula
Edit Formula Dialog
Designer User Interface
Cartridge
Message
Message Mapping
Function Definition
Code Generation
Simulator
Working With Cartridge Designer

Entering a Formula

A formula can include one or more functions and has to be specified in the **Formula** text box. The users need not type the formula directly. Instead, they can choose the required function(s) from the **Function name** list box of the **Edit Formula** dialog box. See the section Edit Formula Dialog for more information. Let's consider an example to understand the usage of this dialog.

In this example, the condition that we will be assuming is that the first character of the input data for a particular field should not be a "/". The following steps have to be performed to validate data to suit the above condition.

a) The first character from the left of the input data has to be extracted.

b) This character should be checked to see that it is not equal to "/".

The above condition can be verified using the following formula:

```
NotEqual(Left(A.F20C.SEME.Reference, 1), "/")
```

In the above formula,

A.F2OC.SEME.Reference is the data field under consideration.

The function **Left(A.F2OC.SEME.Reference, 1)** extracts the first character from the left.

The **NotEqual** function checks if the extracted character is not equal to "/".

Let's now go through the steps involved in specifying the above formula in the **Formula** text box.

1. Double click inside the **Formula** text box. Now pressing the **F4** key or clicking the **Edit Formula** icon ✏ brings up the **Edit Formula** dialog box that can be used in defining the formula.

2. **NotEqual** is a logical function. Therefore, in the **Formula Functions** tab of the **Edit Formula** dialog, select **Logical** item from the **Function category** list box.
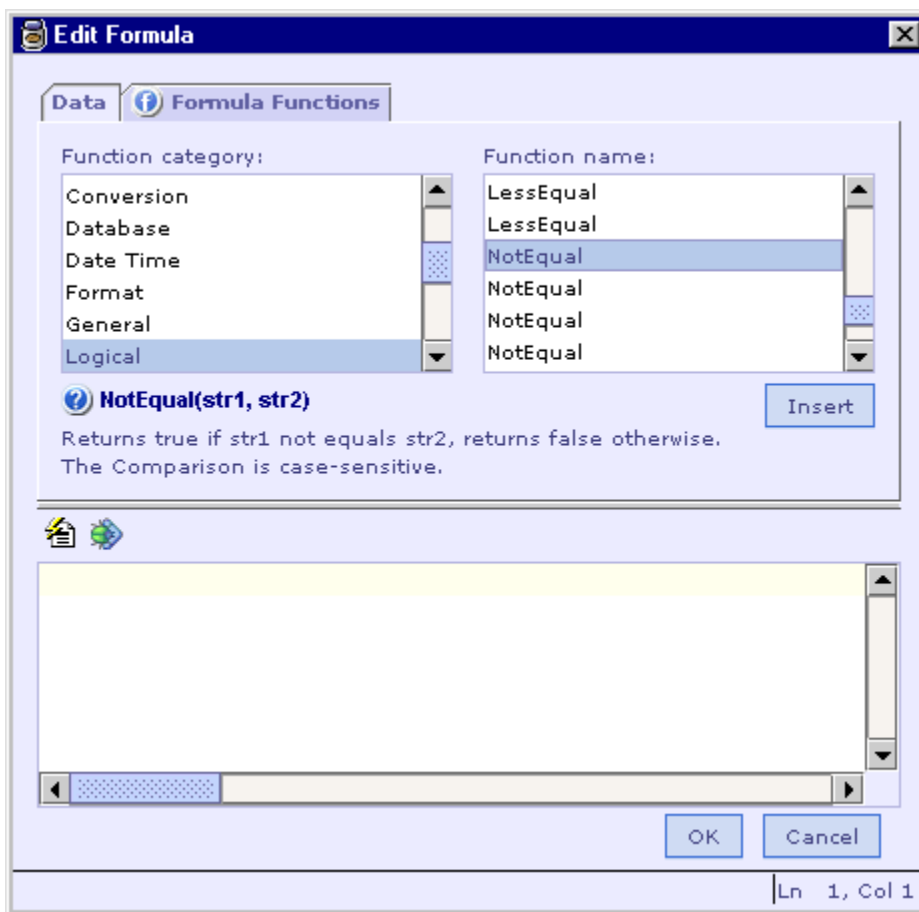
3. The functions under the **Logical** category will be displayed in the **Function name** list box. In the figure shown above only a few of the functions are

currently visible. Use the scroll bar in this box to view the remaining functions. You can double-click on a function name to select it. Otherwise, click on the function name and then click on the **Insert** button that is present below this box. Now select the **NotEqual** function for this example. The current state of the **Edit Formula** dialog box is shown below.
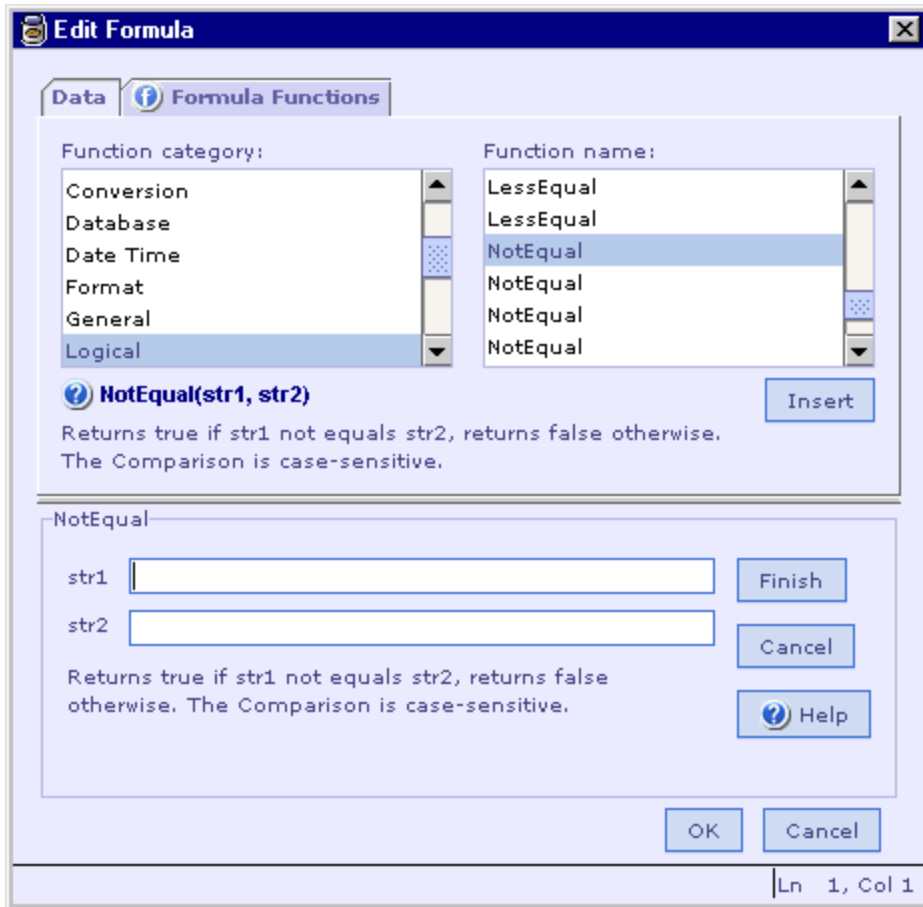
4. Now, the bottom panel of the **Edit Formula** dialog displays another section and the name of this section is the same as the selected function (in our case it is **NotEqual**). The fields in this section correspond to the inputs that need to be specified for the function. In this example, the first value for the **NotEqual** function is another function, which is the **Left** function.

5. Place the cursor in the **val1** field to indicate the insertion point. The function **Left** falls under the **Text** category. So, first select the **Text** category and then the **Left** function as indicated in steps 2 and 3 of this procedure. The bottom panel of the **Edit Formula** dialog will now be different as shown in the following figure. As you can see, the inputs that need to be provided for the **Left** function are displayed.

Edit Formula

Data | ❶ Formula Functions

Function category:
- General
- Logical
- Math
- RegEx
- Swift
- Text
- Transform
- Util

Function name:
- IsAlphaUpperNum
- IsNum
- Left
- LeftStr
- Length
- Lower
- Lpad
- Mid

❷ **Left(str, num_chars)**

Returns the leftmost, specified no.of characters from a text string

Insert

Left

str [ ]   Finish

num_chars [ ]   Cancel

Returns the leftmost, specified no.of characters from a text string

❷ Help

OK   Cancel

6. The first input of the **Left** function is the string from which the leftmost character has to be extracted. In our case it is the **A.F2OC.SEME.Reference** field. To select this field instead of typing it, first make sure that the text cursor is in the **str** text box and then click on the **Data** tab.

7. The current state of the **Edit Formula** dialog box is shown below.
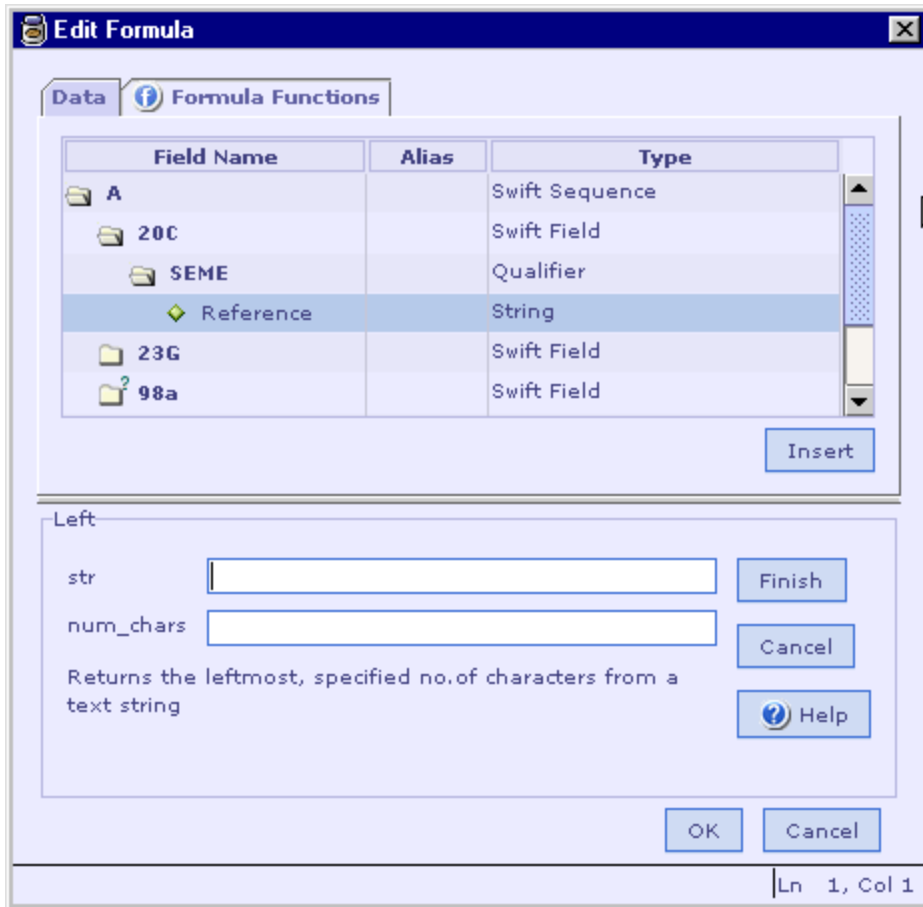
8. In the Fields list displayed in the top panel of the dialog, double-click on the **Reference** field or select it and click the **Insert** button. The field will appear in the **str** field in the bottom panel and the details of this field will be displayed besides the **Insert** button.

9. As only one character needs to be extracted from the left of the **Reference** field, specify 1 as the input for the **num_chars** field as in the following figure.
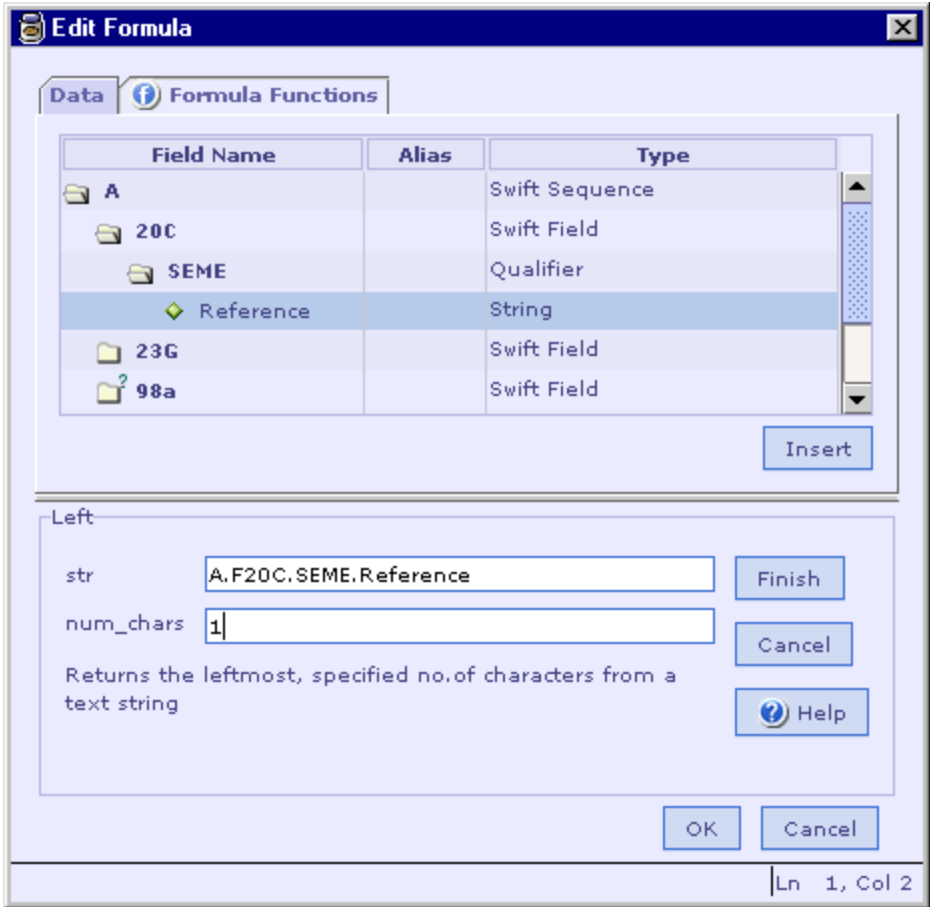
10. Having specified the inputs for the **Left** function, click the **Finish** button. Clicking the **Cancel** button takes the user to the previous screen that displays the input parameters of the **NotEqual** function. The following figure shows the current state of the **Edit Formula** dialog box.
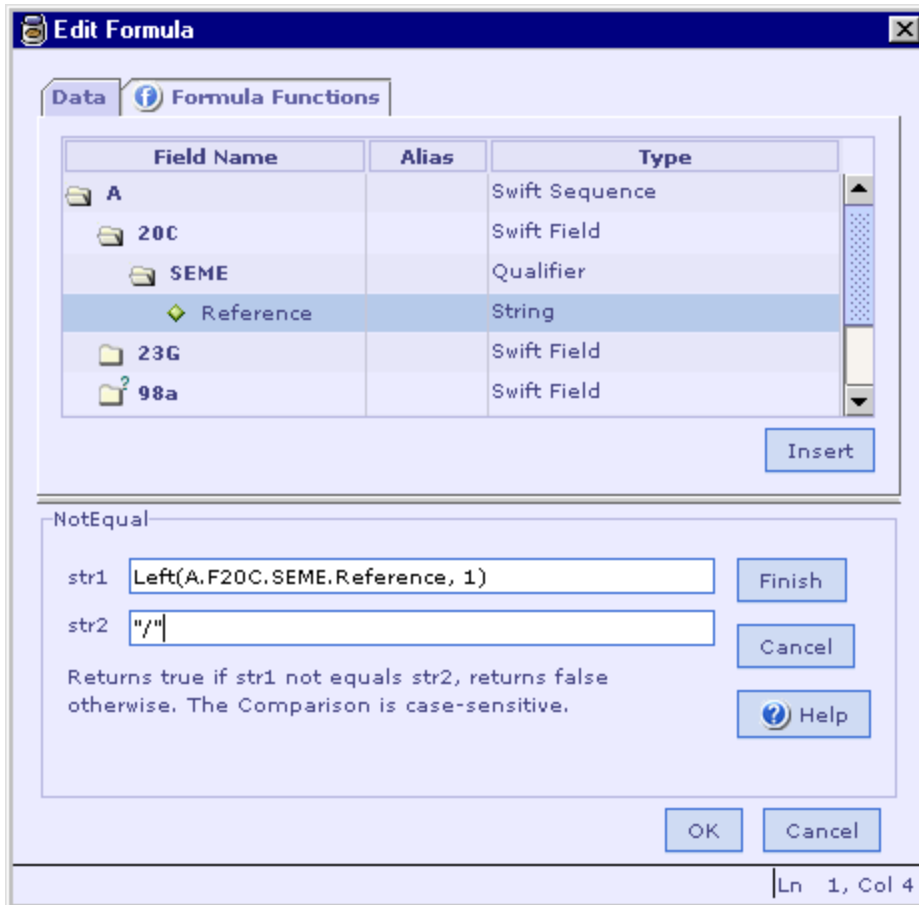
11. As the extracted character needs to be compared with a "/", specify "/" (with the quotation marks) in the **val2** text box. Having specified both the inputs for the **NotEqual** function, click the **Finish** button.

12. As in the following figure, the completed formula appears in the **Formula** text box. The **OK** button at the bottom of the **Edit Formula** dialog enables the user to quit this dialog after inserting the formula into the text field from where the dialog box is invoked. Alternatively, the **Cancel** button allows the user to quit the dialog without inserting the formula.
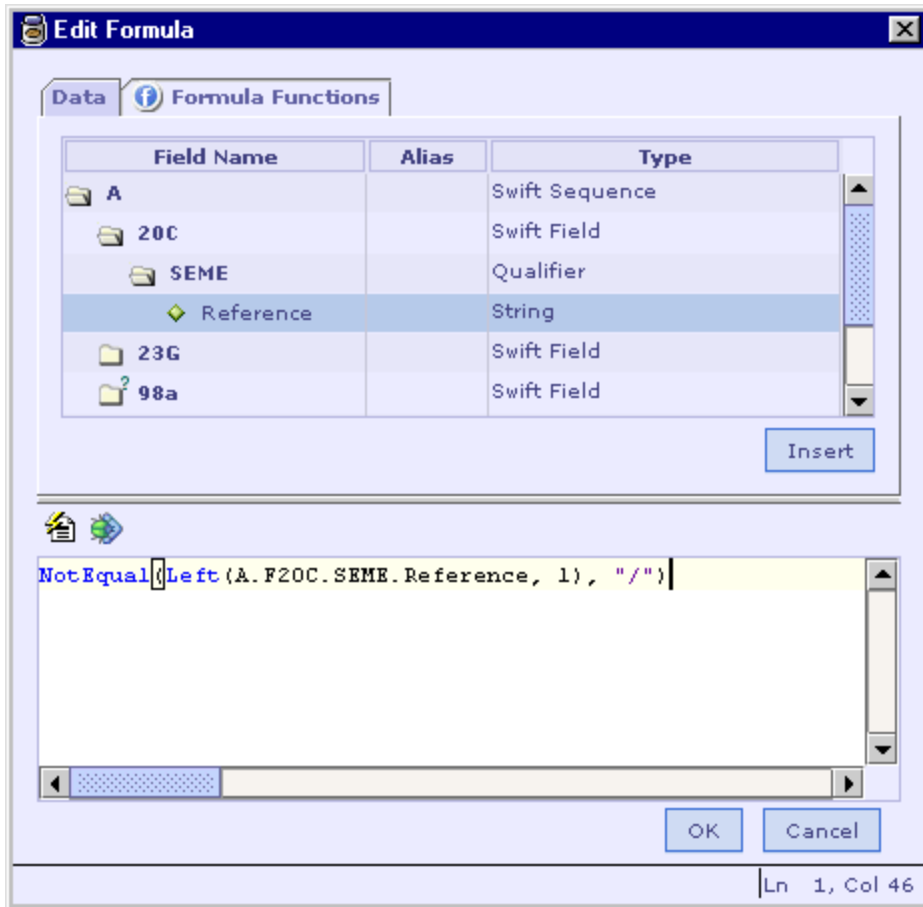
See Also:

[Editing a Formula](#)
[Edit Formula Dialog](#)
[Formula](#)

Editing a Formula

The users may find the necessity to edit a previously specified formula. The required changes can be made very easily. Let us consider the same example as given in the [Entering a Formula](#) section. Let us now see how the changes in the formula should be made if you want to extract the first two characters of the input string and check that they are not "--". To extract the first two characters, we need to edit the **Left** function. To check that these extracted characters are not "--", the **NotEqual** function should be edited.

1. Let us first edit the **Left** function definition. In the **Formula** text box wherein the complete formula is displayed, click on any of the parameters associated with the **Left** function. The signature of the **Left** function is displayed at the bottom of the

Edit Formula dialog box. See the section [Edit Formula Dialog](#) for more information.

2. Click on the hyperlink shown at the bottom of the dialog when the text cursor is moved into the **Left** function signature.
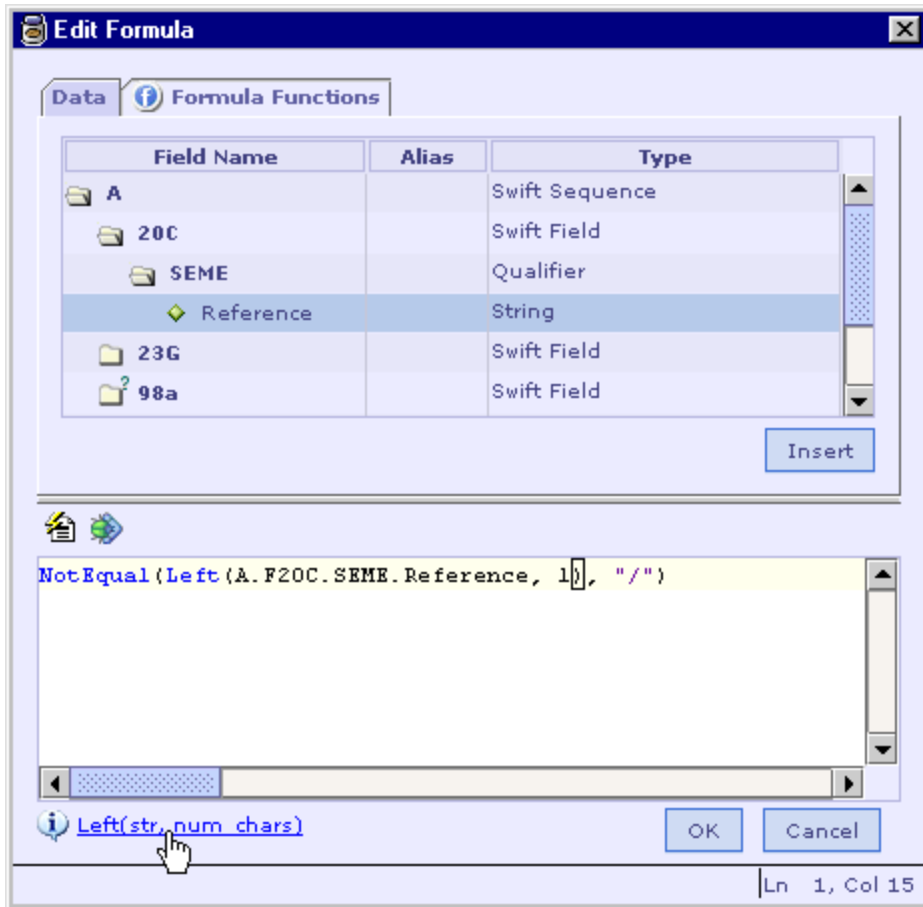
3. In the **Edit Formula** screen that displays the input parameters of the **Left** function, replace the value 1 in the **num_chars** field with 2.
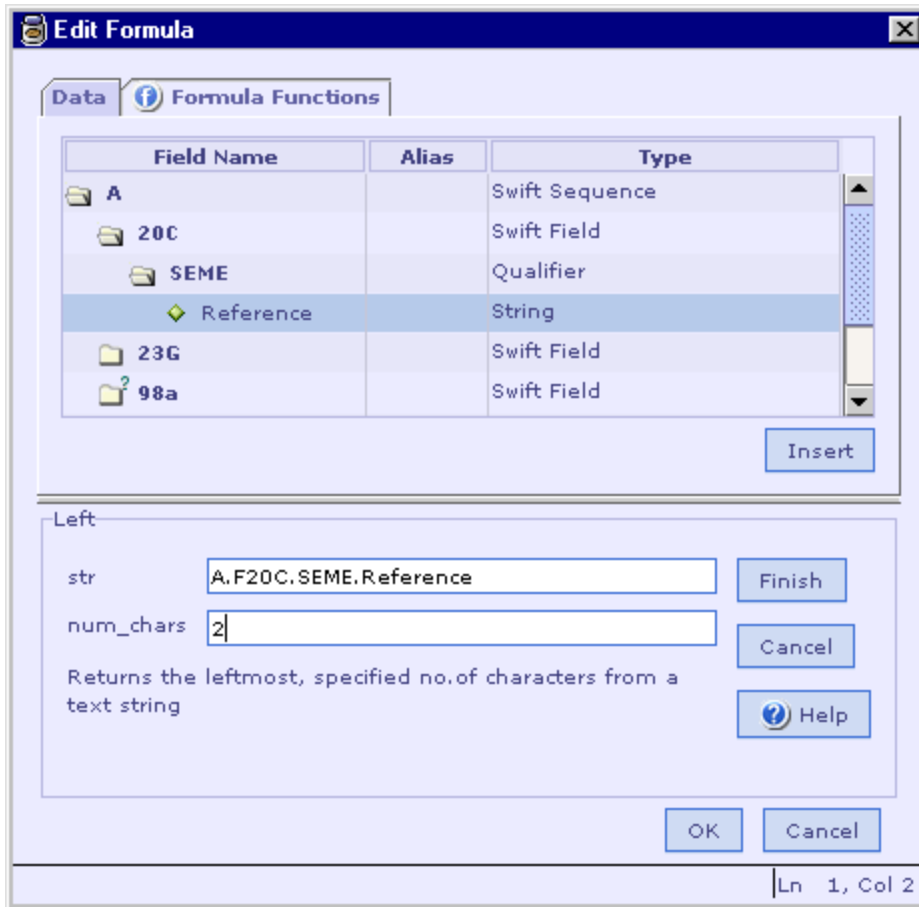
4. Clicking the **Finish** button will update the **Left** function part of the formula.

5. Similarly select any of the parameters associated with the **NotEqual** function and click on the hyperlink displayed.

6. In the **Edit Formula** screen that displays the input parameters of the **NotEqual** function, replace "/" with "--" in the **var2** text box.

7. Click on the **Finish** button to view the modified formula as in the following figure.
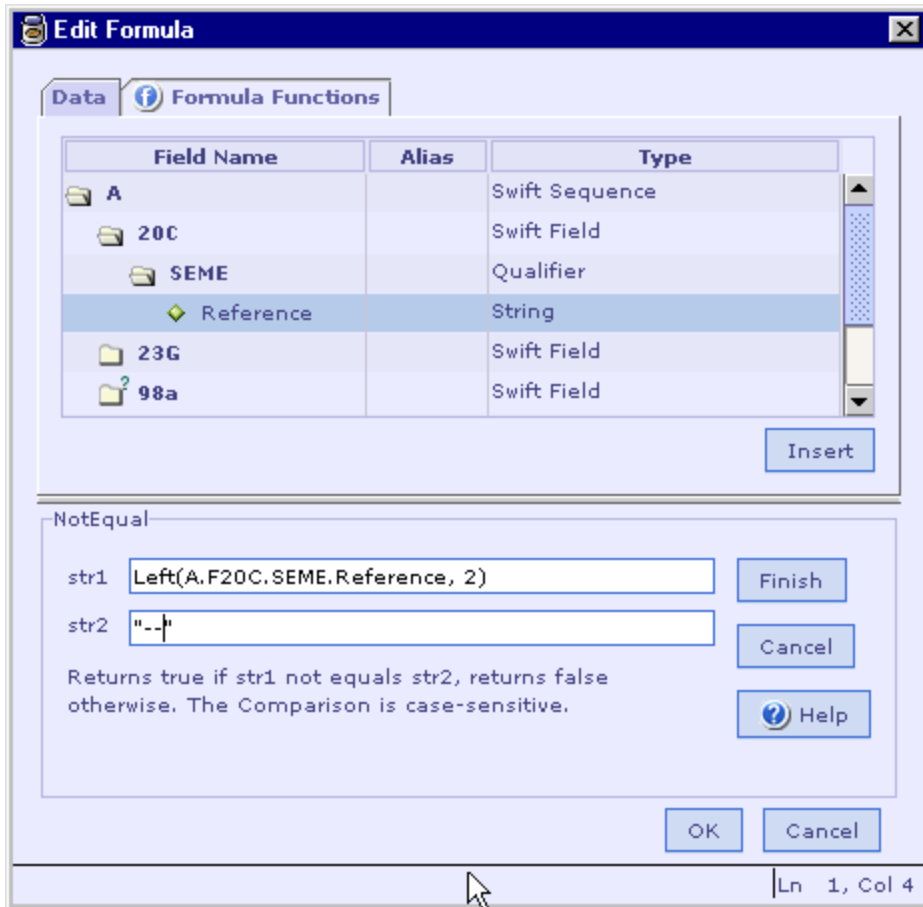
8. Click the **OK** button to accept the changes made to the formula. Clicking the **Cancel** button will retain the original formula without any changes.

Note:

The function help displayed in the status bar of the application window is also a hyperlink. Clicking on it will also launch this dialog.

See Also:

Entering a Formula
Edit Formula Dialog
Formula

Reformat Formula

The 'Reformat Formula' dialog can be used to auto-format the formula code. If a formula is typed with unwanted spaces or lines, it can be beautified using this feature.

1. Select 'Reformat Formula' menu item under 'Edit' menu. The Reformat Formula dialog is displayed.
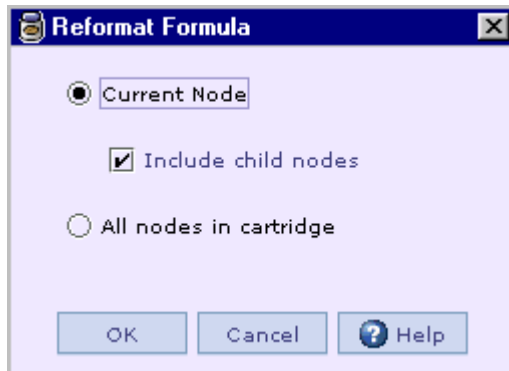
2. If you want to format formula present only in the current node select the 'Current Node' option button. If you want to format all formula present in the current node and its child nodes select the 'Include child nodes' check box.

3. If you want to reformat all formula entered in the cartridge, select 'All nodes in cartridge' option.

See Also:

Editing a Formula
Edit Formula Dialog

Edit Formula Dialog

The **Edit Formula** dialog helps in entering/editing formulas, checking the correctness of a formula and testing it. This dialog can be popped up using the **Edit Formula** 🖊 button in the main window toolbar or using the shortcut key **F4**, when the focus is in the formula text box. The **Edit Formula** dialog is shown below.
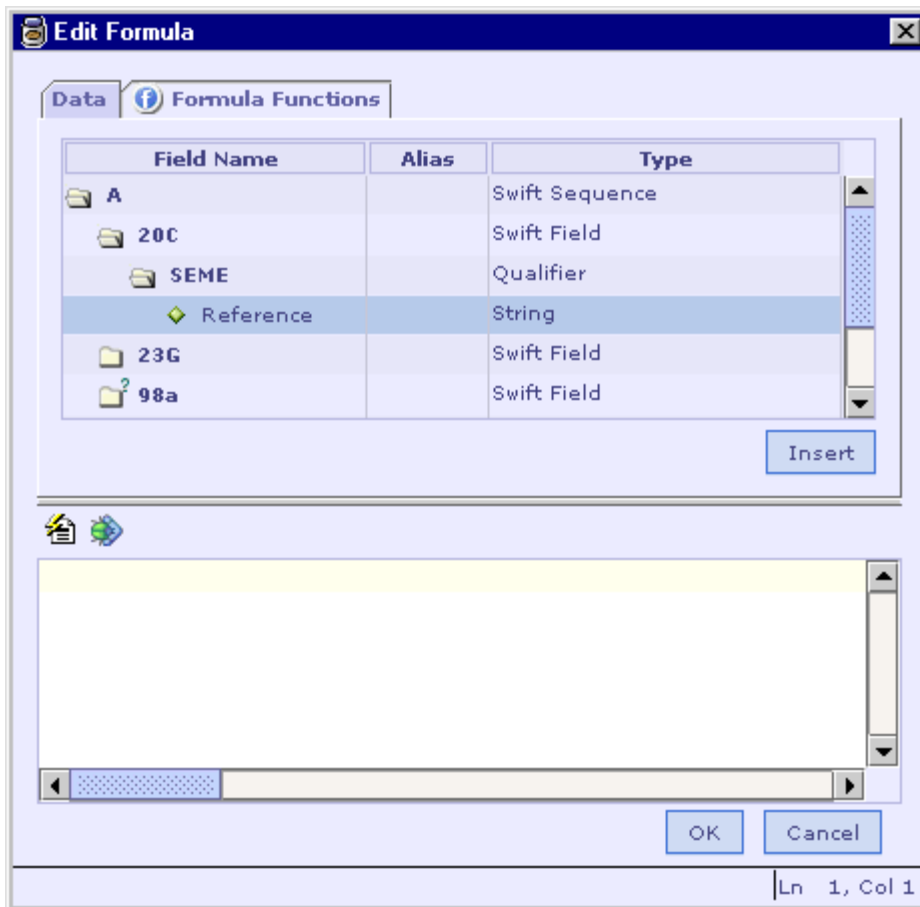
In the above picture, the **Data** tab is currently active. The top panel of the dialog displays the Fields list.

The bottom panel of the dialog displays the formula editor. In the formula editor, the user can directly type the required formula that might result in errors like syntax error, misspelled parameter name, etc. To avoid these common errors, it is recommended to insert data fields and functions used in the formula from the tabs of the **Edit Formula** dialog.

The **Formula Functions** tab lists the supported functions as shown in the following figure. These functions are categorized based on their functionality. The **Function category** list box displays the different function categories. The user can click on

any item in this box to select a specific category. The functions falling under the selected category will be displayed in the **Function name** list box. In the following figure, the functions in the **Arithmetic** category are displayed. On selecting any function, the syntax and usage of the function will be displayed below the list boxes.
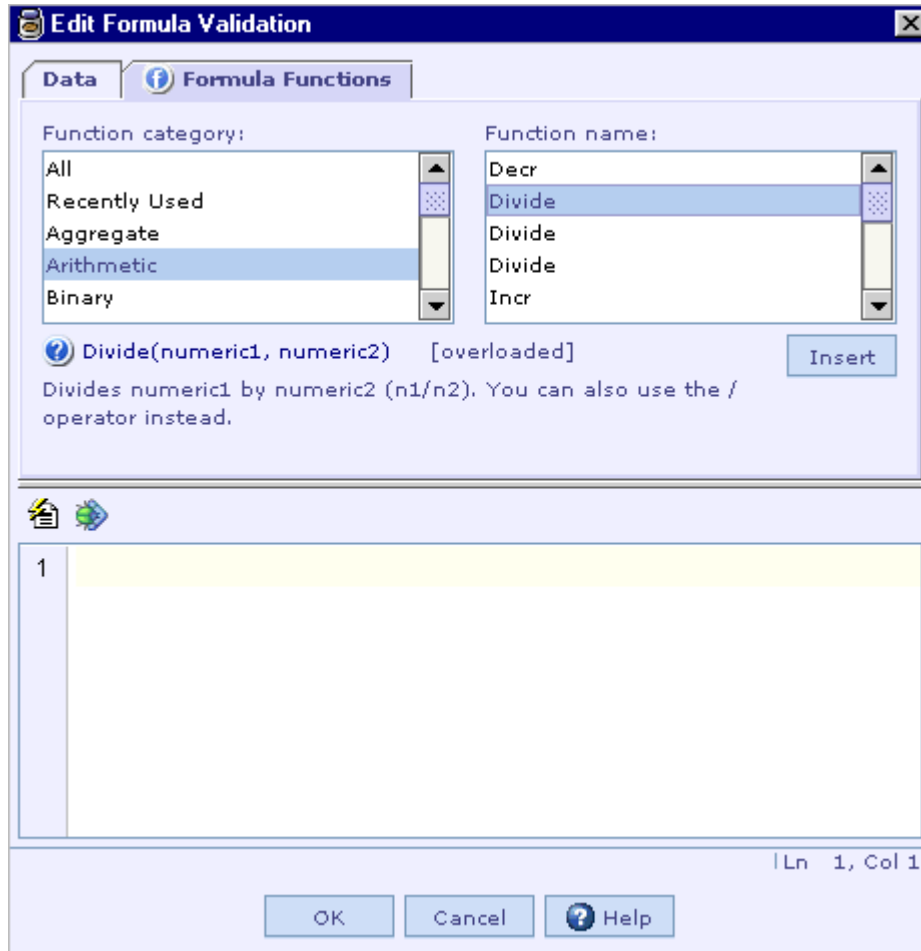
A splitter separates the top panel and bottom panel containing the formula editor. You can adjust the size, as you want. Line number would appear for the codes besides the code pane. The dialog size and the position of the splitter are now sticky (for the session). Any adjustments you make are retained the next time the dialog is shown.

See Also:

Syntax Highlighting
Auto Completion
Quick Function help
Code Reformater (Formula Beautifier)
Formula Validation (Automatic Error Checking)
Formula Tester
Locating Variable

Syntax Highlighting

In the formula editor of the **Edit Formula** dialog, the formula functions and string literal(s) of a formula are colored differently as shown in the following picture.

See Also:

Auto Completion

The **Edit Formula** dialog makes entering formula easier by supporting auto completion of function names in the formula editor.

Follow the Steps given below to auto complete function names while entering a formula in the formula editor:

1. Type in part of the function name.

2. Press Ctrl + Space.

3. A list box pops up with possible choices. Select the one you want and press Enter or '('.

See Also:

[Syntax Highlighting](#)
[Quick Function help](#)
[Code Reformater (Formula Beautifier)](#)
[Formula Validation (Automatic Error Checking)](#)
[Formula Tester](#)
[Locating Variable](#)
[Edit Formula Dialog](#)

Quick Function help

In the formula editor while entering a formula that involves functions, help for a function can be displayed without leaving the editor by pressing the Ctrl button and hovering the mouse pointer over the function name.
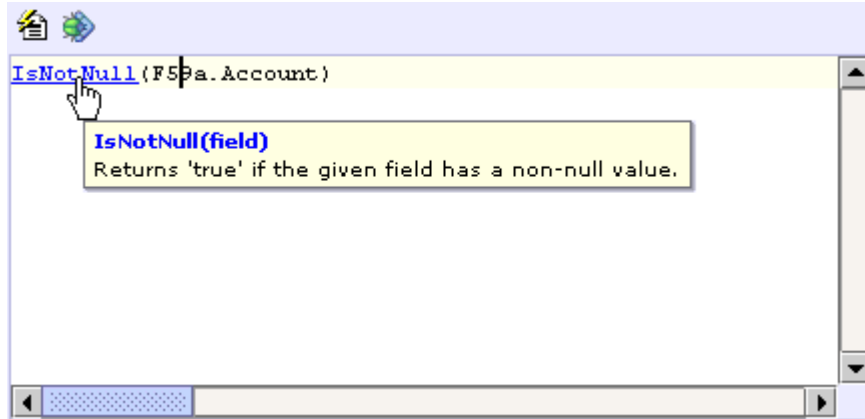
See Also:
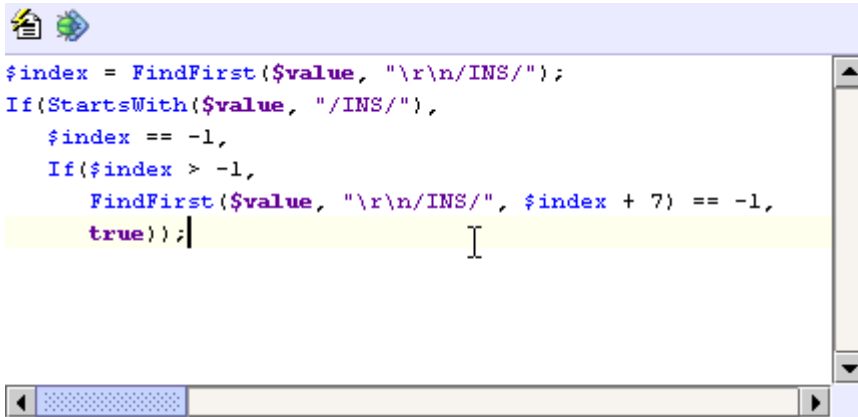
Code Reformater (Formula Beautifier)

The 'Reformat Code' button in the formula editor toolbar of the **Edit Formula** dialog can be used to auto-format the formula code. If a formula is typed with unwanted spaces or lines, it can be beautified using this feature. For example, consider the following formula:
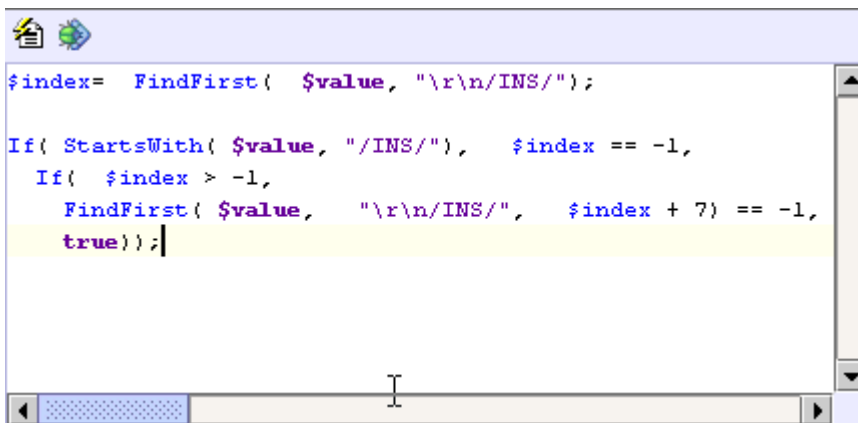
To format the above formula (remove unwanted spaces and lines) click the 'Reformat Code' button in the formula editor toolbar. The formula will be automatically formatted as shown below:
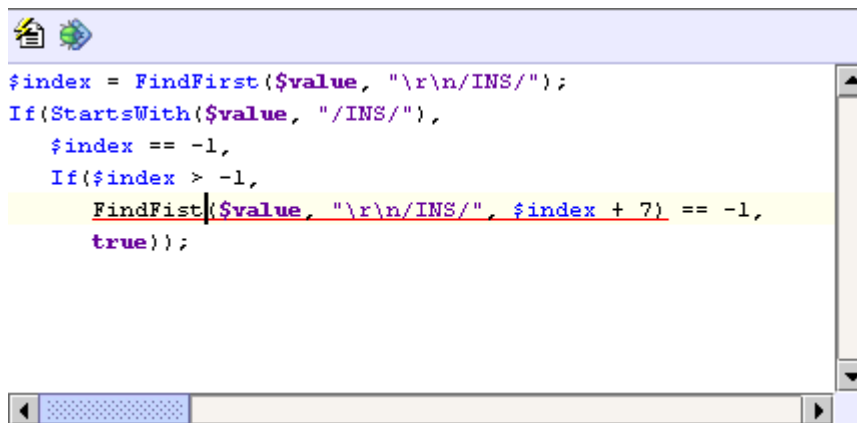
```
$index = FindFirst($value, "\r\n/INS/");
If(StartsWith($value, "/INS/"),
    $index == -1,
    If($index > -1,
        FindFirst($value, "\r\n/INS/", $index + 7) == -1,
        true));
```

See Also:

Syntax Highlighting
Auto Completion
Quick Function help
Formula Validation (Automatic Error Checking)
Formula Tester
Locating Variable
Edit Formula Dialog

Formula Validation (Automatic Error Checking)

While entering a formula in the formula editor of the **Edit Formula** dialog, the syntax and correctness of the formula are checked as it is typed. The offending code is underlined (in red) and error message is displayed in the status bar as shown in the following picture.

```
$index = FindFirst($value, "\r\n/INS/");
If(StartsWith($value, "/INS/"),
    $index == -1,
    If($index > -1,
        FindFist($value, "\r\n/INS/", $index + 7) == -1,
        true));
```
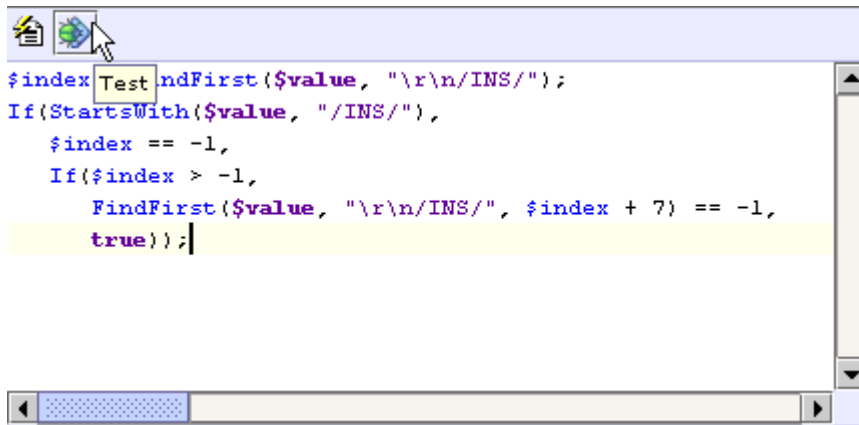
See Also:

Syntax Highlighting
Auto Completion

Formula Tester

This feature can be used to test a formula in isolation.

On clicking the 'Test' button in the formula editor, the **Test Formula** dialog is displayed as shown in the following picture. The user will be prompted for the value of all the input variables used in the formula. The formula is executed and the output(s) are displayed.
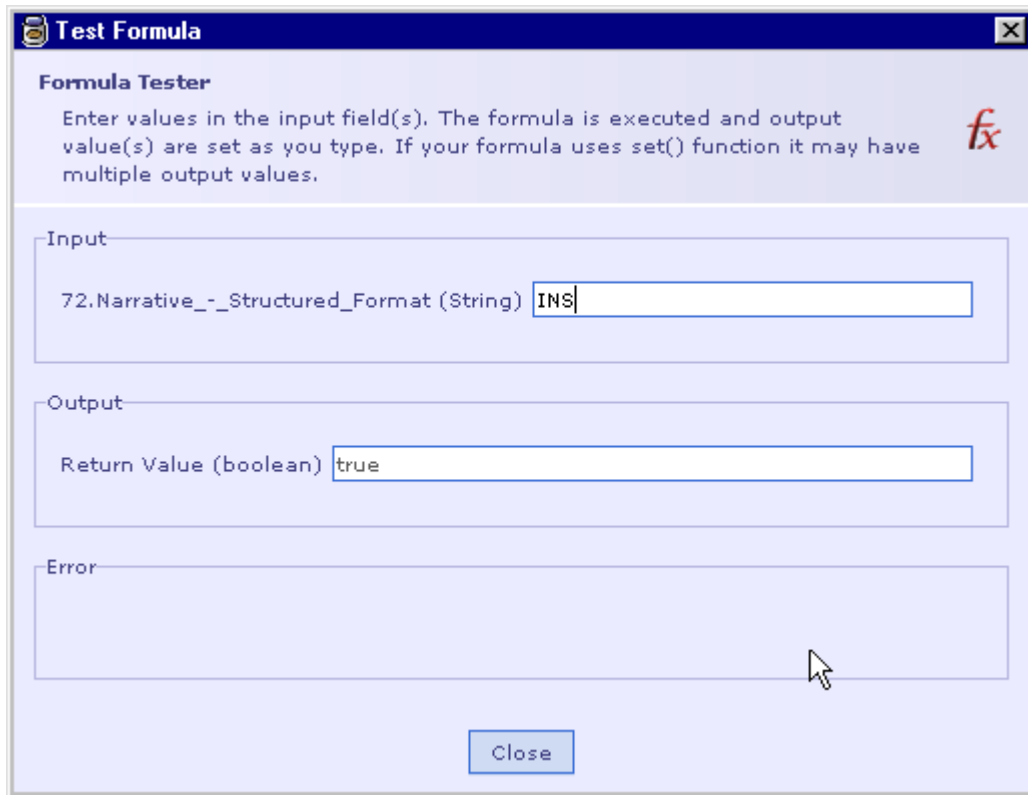
In some cases where sections and arrays indexes are used, this feature does not work well.

See Also:

Syntax Highlighting
Auto Completion
Quick Function help
Code Reformater (Formula Beautifier)
Formula Validation (Automatic Error Checking)
Locating Variable
Edit Formula Dialog

Locating Variable

To locate the definition of a variable used in the formula, move the mouse pointer over that variable with the CTRL key pressed. A hyper link is shown (similar to functions) as can be seen in the following picture. Clicking on it will take you to the definition of that variable. If it refers to a field, the field is selected in the Field list in the top panel of the **Edit Formula** dialog.

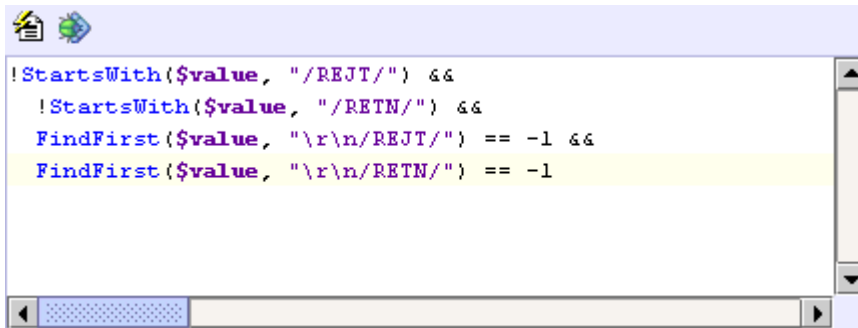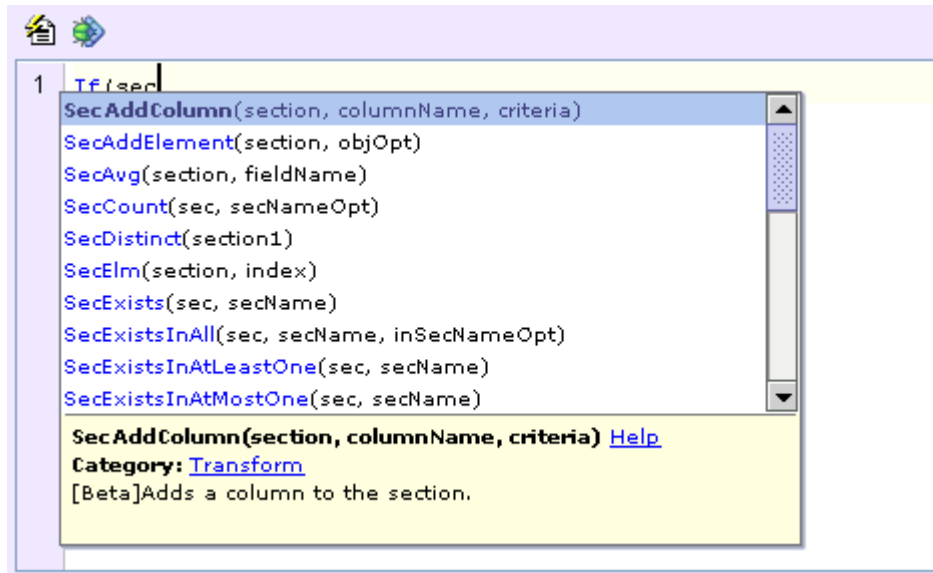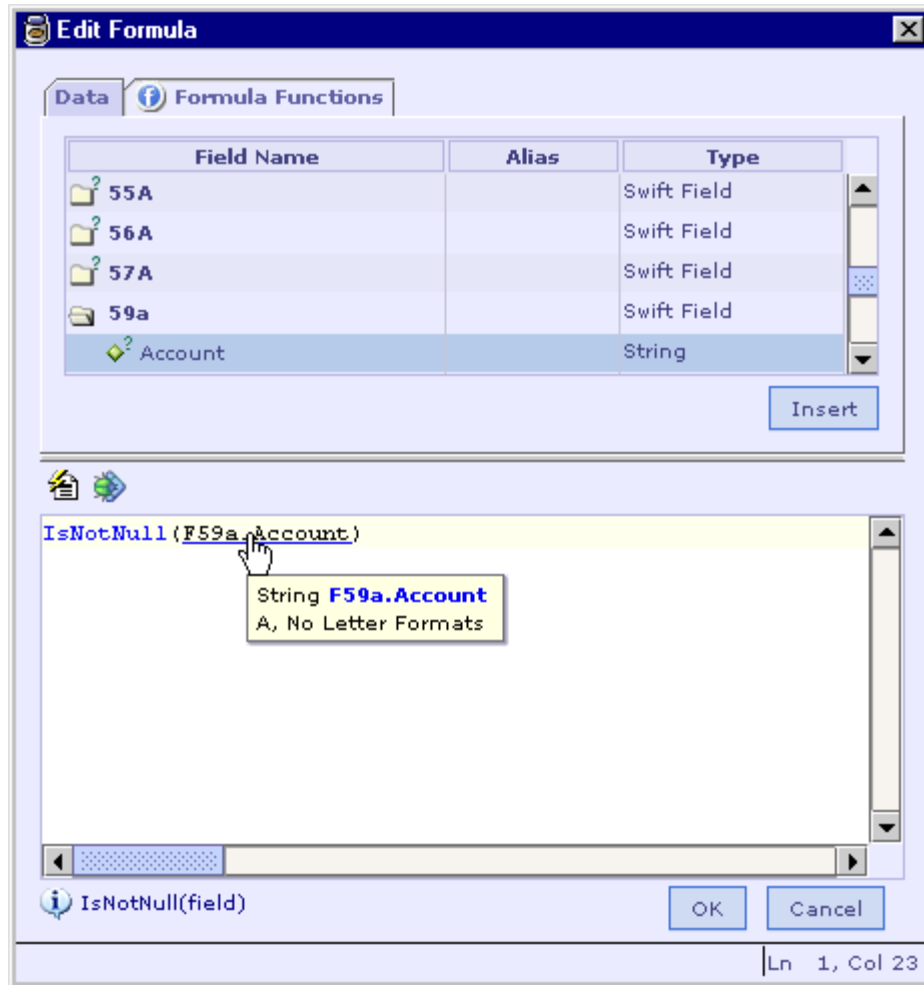**See Also:**

[Syntax Highlighting](#)
[Auto Completion](#)
[Quick Function help](#)
[Code Reformater (Formula Beautifier)](#)
[Formula Validation (Automatic Error Checking)](#)
[Formula Tester](#)
[Edit Formula Dialog](#)

Formula Edit Operations

Several edit operations can be performed on the formula entered in the formula text area. Select the text entered or a portion of the text and right click. A popup menu is displayed as shown below

```
If(IsNull(A.F17N.Indicator),
   null,
   If(Equal(A.F17N.Indicator,"N"),
      "Gross Settlement"
    , "Net Settlement"))
```

✂	Cut
📋	Copy
📋	Copy as HTML
📋	Paste
🔍	Find
🔍	Find Next
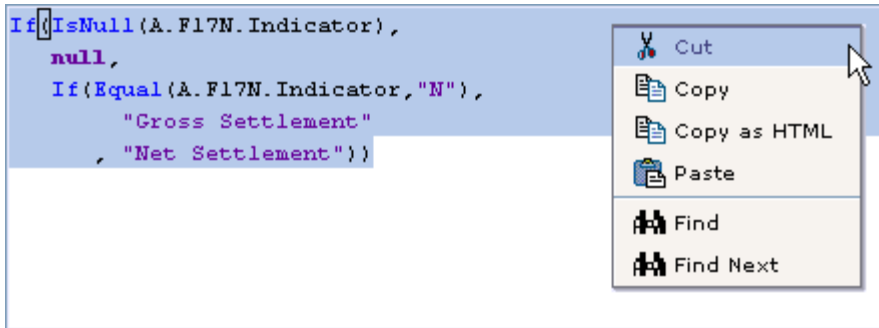
Select the '**Cut**' menu to remove the selected text.

Select the '**Copy**' menu to copy the selected text.

Select '**Copy as HTML**' menu to copy the selected text using HTML syntax. It can be saved into an HTML file.

Select the '**Paste**' menu to paste any content present in the clipboard into the text area.

To search for particular text in the formula select the '**Find**' menu. The Find dialog is displayed.

Find

Find What Net Settlement

Search Options

☑ Match Case

☐ Match Word

☐ Regular Expression

OK   Cancel
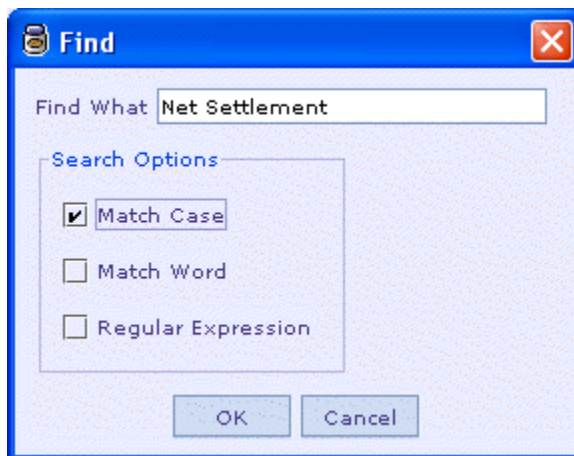
Specify the text to search for in the 'Find What' text field. You can also specify search options. Press OK. If a match occurs the text will be highlighted as shown below

```
If(IsNull(A.Fl7N.Indicator),
   null,
   If(Equal(A.Fl7N.Indicator,"N"),
      "Gross Settlement"
   , "Net Settlement"))
```

Select the '**Find Next**' menu if you need to search for the text again.

See Also:

Function Definition

Designer supports defining functions in the Formula language. The functions are defined in a Cartridge and are available globally in the cartridge. Once defined, a function can be used in formula code from anywhere in the cartridge.

Like functions in other languages, during definition you need to specify the parameters, return type and provide a body for the function. The body of the function should be in formula code. The parameters and return value should be of the specified type. During definition you can choose to use an 'Any' for parameters and return type which makes the functions behave like generic functions, applicable to multiple types.

See Also:

Defining a Function

To create a new function,

1. Right click the cartridge node, select 'Add Item-> New Function' as shown below.
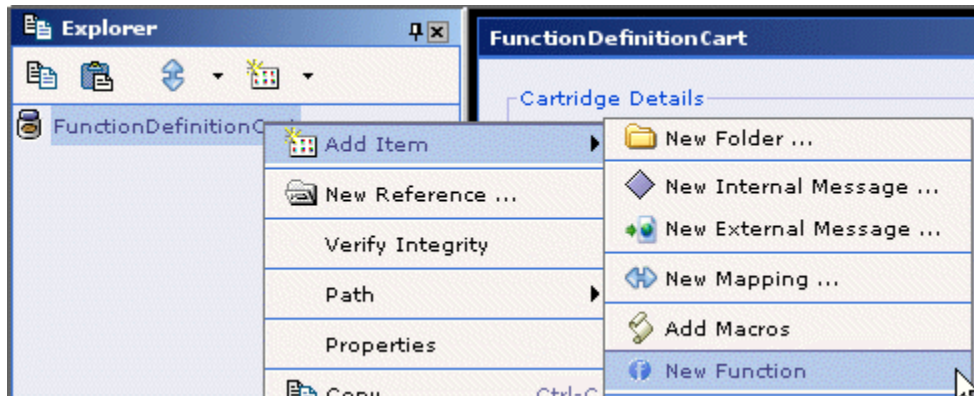
2. A new function definition will be added.
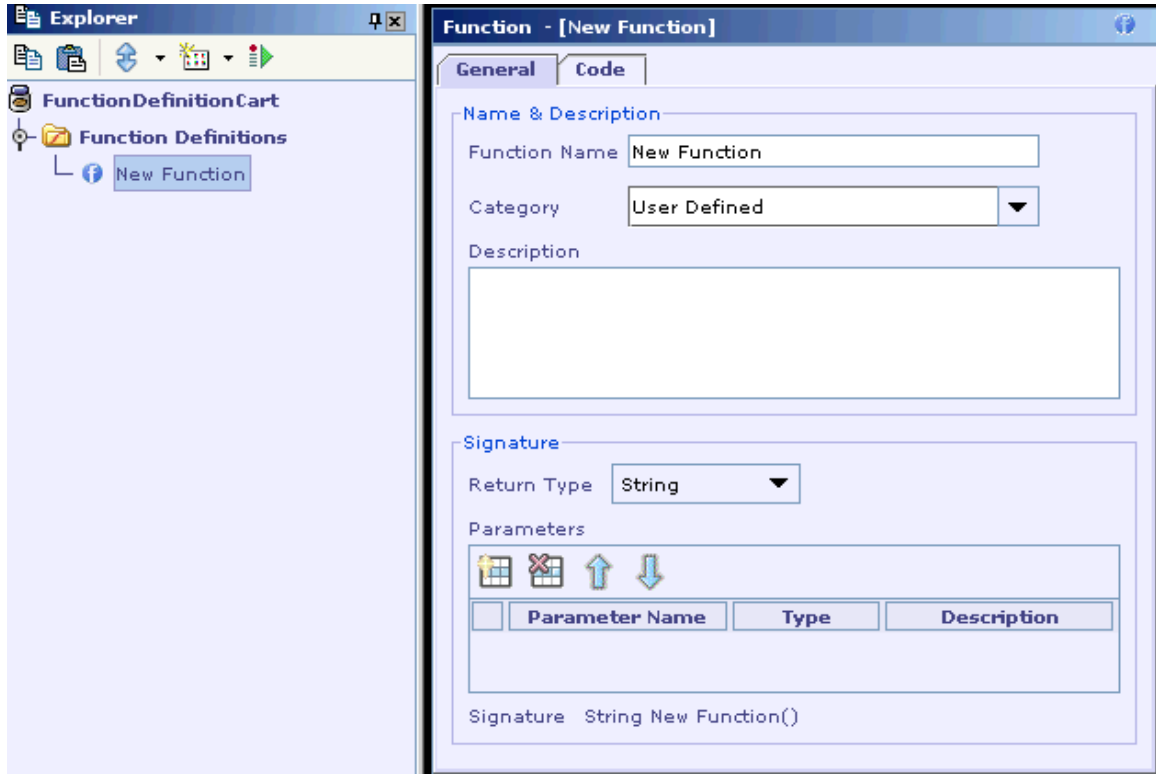
3. The function definition by default will be added under a folder 'Function Definitions'. Any function definition you add will be added under this folder.

4. You can also add a function definition by right clicking on the 'Function Definitions' folder and selecting 'New Function' item.
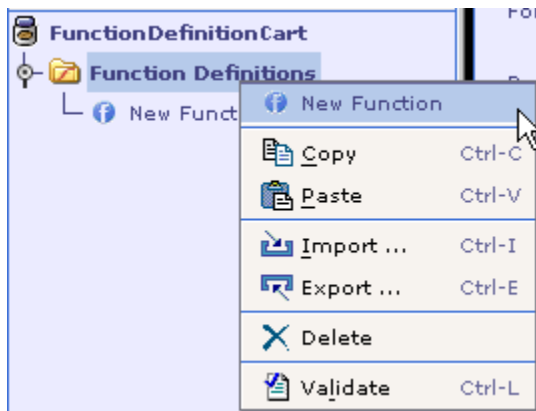
Note:

A function definition defined in a cartridge can be accessed only within that cartridge. It cannot be accessed from elsewhere.

Function Definition UI

After creating a function definition you can specify properties for it. The properties that can be specified for a function definition are

Name
Category
Description
Parameters
Return Type
Code

Name

The name of the function definition. The function should be accessed using this name. This is a mandatory property. The name should conform to identifier rules.

Function Name ConcatFieldValues

Category

The category of the function definition. The 'User Defined' category is selected by default. You can select any other category from the drop down list. In the 'Edit Formula' dialog the function will be listed under the specified category.

Category   User Defined

Description

Description for the function definition. This is not mandatory. When a function definition is selected in the functions list box the description specified here will be displayed.

See Also:

[Name](#)
[Parameters](#)
[Return Type](#)
[Code](#)

Parameters

The parameters for the function. A function can have multiple parameters or no parameter. Parameters are not mandatory for a function.

Properties of a parameter are

Name: Each parameter name should be unique.
Description: Description of the parameter. This is not mandatory.
Type: The parameter type. All the supported Designer types can be specified as parameters. Along with those types four new types can be specified.
    Any
    Token
    Section
    Message

See Also:

[Name](#)

Return Type

The return type of the function. The type can be any of the supported designer types. Five new types are also supported as return types.

Any
Token
Section
Message
Void

See Also:

Code

The actual operation that the function performs should be specified here.

You can choose to implement the operation of the function using either the formula language or the platform specific code.
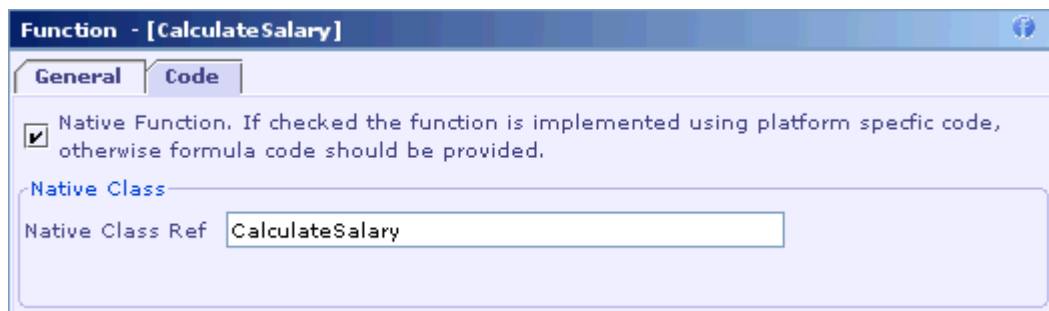
To implement the function using platform specific code, you have to select the 'Native Function' check box and specify a reference name in the 'Native Class Ref' text field. The reference name should be bound to the platform specific class (that

contains function implementation) using the 'Language Bindings' tab of the 'Code Generation Settings' dialog. The custom class must implement the **IInvokable** interface. See API documentation for more details.

Please refer New File from Template for easily creating a native function implementation class.
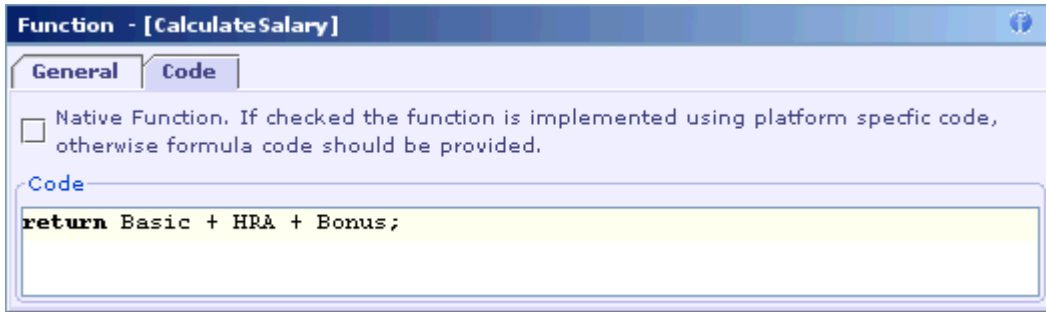
To implement the function using the formula language, you have to deselect the 'Native Function' check box (by default it is deselected) and specify the implemenation code in the Code text area.

You can click the [f] button or press 'F4' in the Code text area to view the 'Edit Formula' dialog.
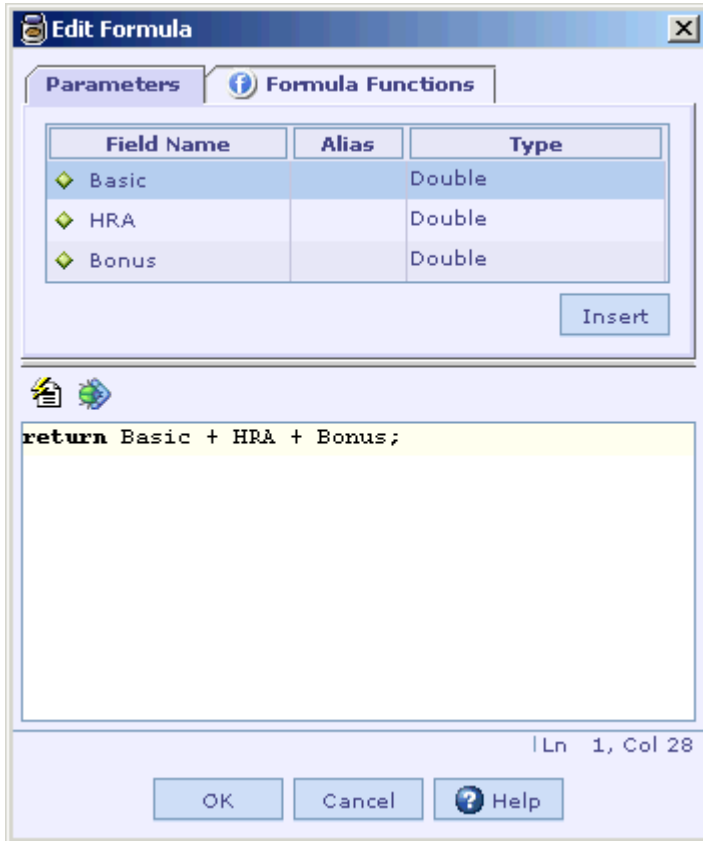
You can access all parameters that are defined for the function in the formula dialog.

Please note that the value returned by the function should correspond to the return type. If the return type is 'String' the function should return a String value. If the return type is 'Void' the function should not return any thing.

See Also:

[Name](#)
[Category](#)
[Description](#)
[Parameters](#)
[Return Type](#)
[Formula](#)
[Edit Formula Dialog](#)

Copy/Paste Support

Copy/Paste operations are supported for a function definition. Right click the function definition
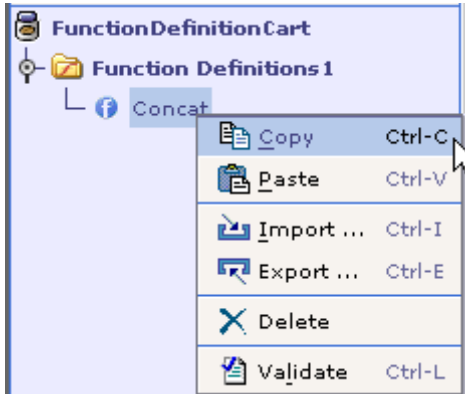
Select the 'Copy' menu item. The function definition will be copied. You can then paste it elsewhere.

Function definitions can also be exported/imported as XML files.

See Also:

Defining a Function
Working with Functions
Invoking Functions

Invoking Functions

You can access the function definitions in all places where formula support is provided (e.g. validation rules). Please note that a function definition can be accessed only within the cartridge where it has been defined. It cannot be accessed from outside the cartridge.

In the Edit formula dialog in 'Formula Functions' tab category 'Functions' will be present. When you select it, only those function definitions that have been added in the cartridge will be displayed in the functions list box. You can select a function definition from the ones available.
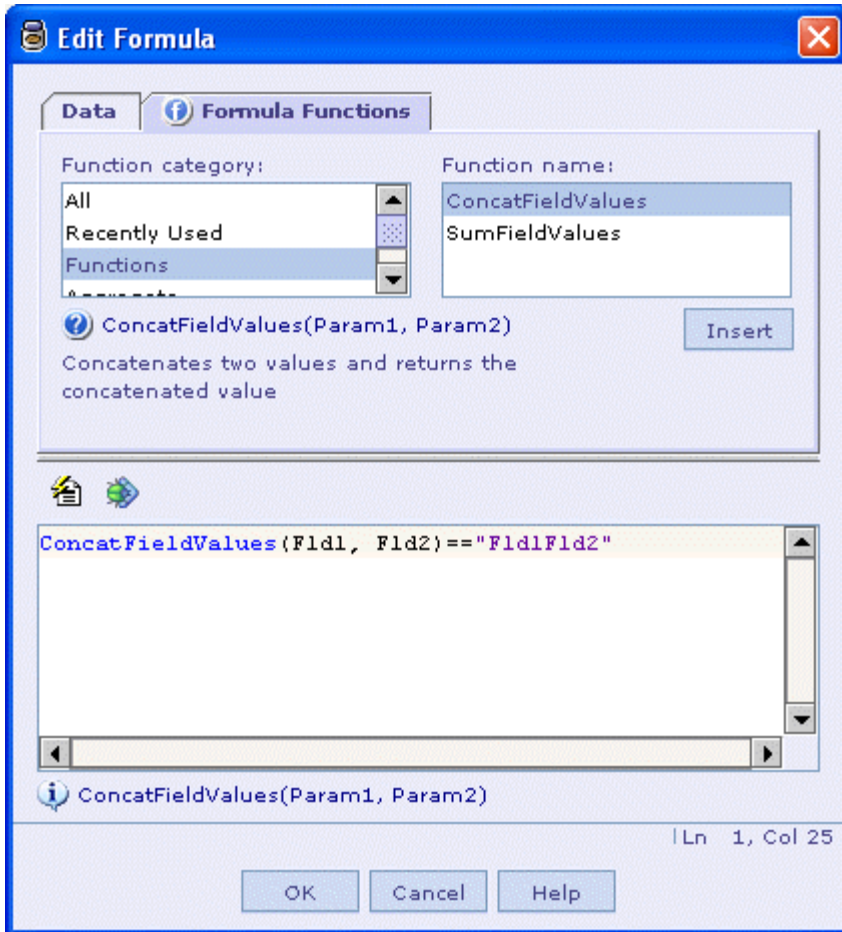
When a function definition is selected in the functions list box the description of the function that is specified while defining it will be displayed. The signature of the function will also be displayed.

Note:

The help links for the user defined function definitions will not work.

See Also:

Defining a Function
Working with Functions
Edit Formula Dialog

Working with Functions

In this section we consider specific cases and illustrate various possibilities in creating functions. There are three major classifications of functions. The classifications are based on the parameter type used while defining the function.

Simple Functions	Functions whose parameters and return types are all of known type
Parameterized functions	Functions with at least one parameter of type 'Any' (unspecified).
Token based functions	Functions with at least one parameter of type 'Token'.

See Also:

Testing the function
Defining a Function
Invoking Functions

Simple Function

Lets us assume that you want to create 'min' function, which returns the minimum of two integers. The function will take two parameters of type Integer and return the lesser of the two, again an Integer.

1. Create a new Function and set its name to 'min'.

2. Change the Return Type to 'Integer'

3. Add two Parameters 'val1' and 'val2' and set their type as Integer.

4. Enter a description of the function.
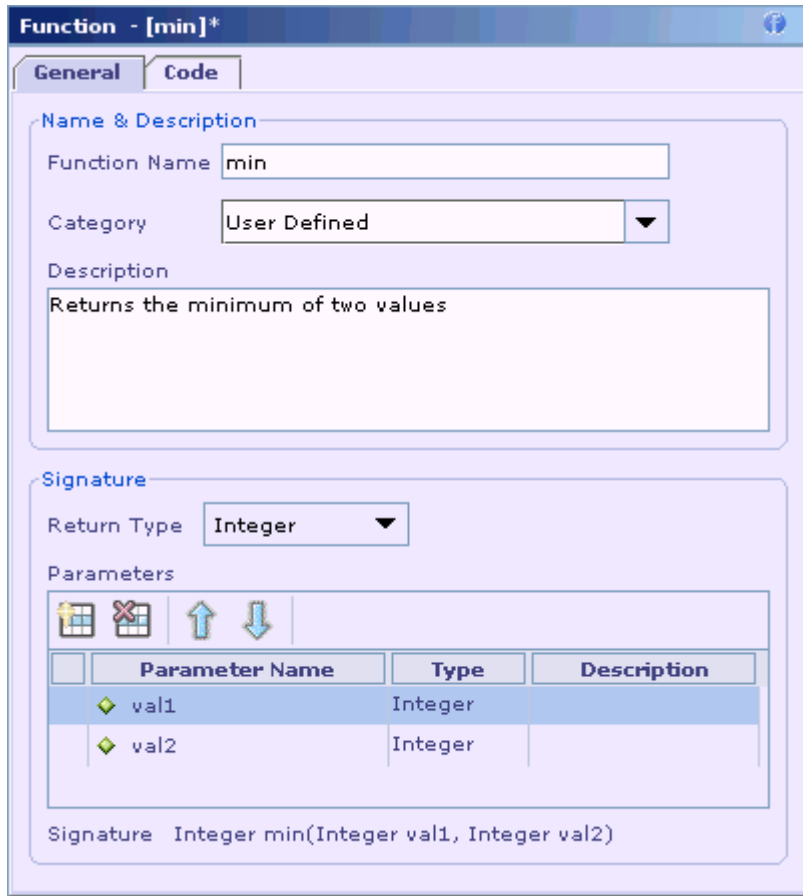
5. In the code area enter the body of the function.

```
val1 < val2 ? val1 : val2
```

You can also use an explicit return statement. If not it will use the value of the last top level expression in the body of the method.

In case of complex formula you can use the Edit Formula Dialog to enter the text. The main advantage of the dialog is that it provides immediate error feedback and has basic testing capabilities.

This function can be invoked from others locations in the cartridge where you are allowed to enter a formula. The newly added function appears in the function list in Edit Formula Dialog like any other predefined function.

See Also:

Parameterized functions
Token based functions
Testing the function
Defining a Function

Testing the function

There are two ways you can test this newly created function. You can test the code that forms the body of the function or you can invoke the function from elsewhere in the cartridge and check whether it returns correct values.

See Also:

Testing the body

1. To quickly test implementation of the above function, place the cursor in the Function code area (in the Function Definition UI) and Click F4. The Edit formula dialog pops up.
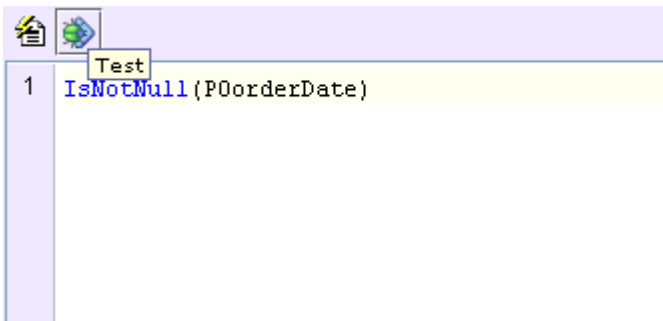
2. Click the 'Test' button in the toolbar. The Test Formula dialog is displayed.
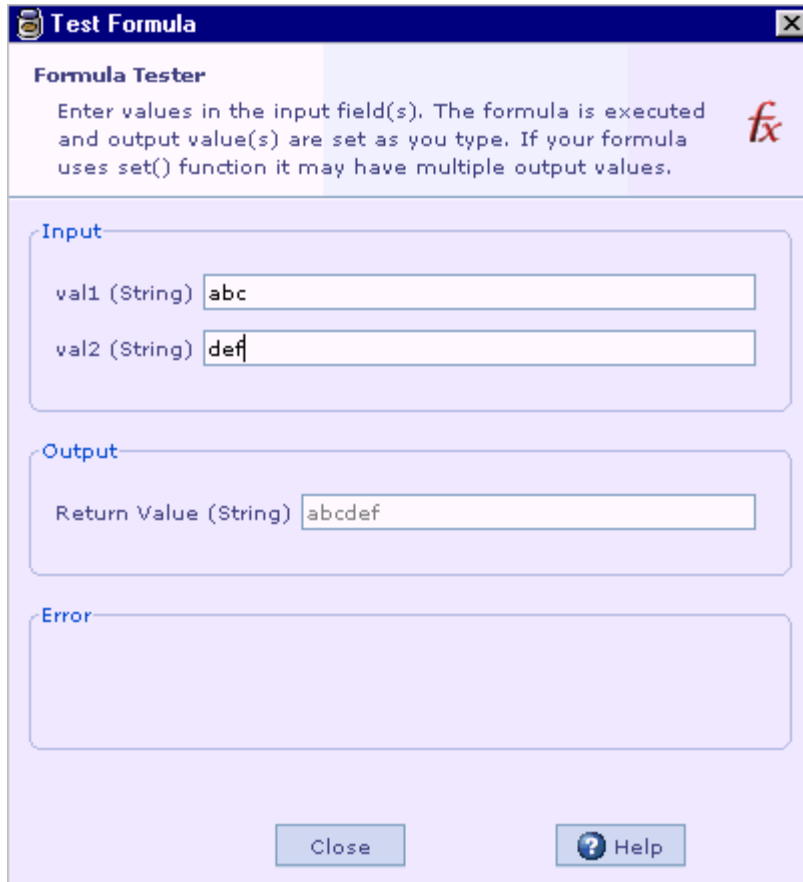
3. There will be a text field for each parameter to enter the input for function. Enter some integer values for the two parameters. The result is immediately computed and displayed in the 'Return' text field.

Here we are directly testing the body of the function; we do not actually invoke it.

See Also
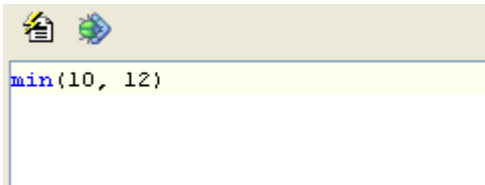
Test by invoking the function
Edit Formula Dialog

Test by invoking the function

The other way of testing this function, is to invoke it from some other location in the cartridge where formula is supported.
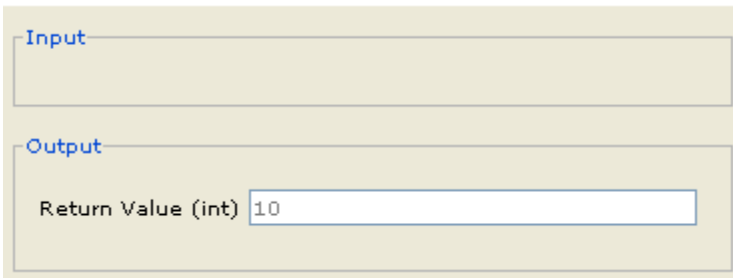
To do that, create a new dummy function definition. We will use the body of the function to invoke the 'min' function and test whether it works. It is also possible to test it from any other location where formula is supported by following the procedure below.

1. To test min function, place the cursor in the Function code area (of the dummy function) and Click F4. The Edit formula dialog pops up.

2. Enter the code 'min(10, 12)' in the code area. Here we are calling the min function we created earlier. Unlike the previous case, here we are actually trying to invoke the function and testing it from outside.

3. Click the 'Test' button in the toolbar. The 'Test Formula' dialog is displayed. Since we are not using any variables in the test code, it does not prompt for any arguments and displays the output/return value.

These two ways of testing appear very similar and one gets the impression that one of them is redundant. While the former is easy to do (because you don't have to create a dummy call point), the latter is more versatile. In particular, when we use parameterized functions, you cannot test it using the former approach, since types of the parameters are not known.

See Also:

Testing the body
Edit Formula Dialog

Parameterized functions

The min function we wrote is specific to Integer type. It takes two Integer parameters and returns an Integer as result. If you follow this approach, you need to define a min function for each type you want to support or need. It would be nice if we can generalize the same function to support all types.

```
val1 < val2 ? val1 : val2
```

If you look at the body of the function, it looks like it would work for any type that supports a less than operator. Since this operator is supported for all basic types (String, Double etc), the same body should work for all basic types. The only problem is that we are asked to specify the parameter type and the return type when we define the function.

Designer provides a way to "generify" this function definition. The trick is to not commit to a specific parameter/return type. In the 'Return Type' and 'Parameter Type' combo boxes, there is type called '*Any*'. If you chose this type, your function is not tied to a particular return type or parameter type respectively.

Lets try to parameterize the *min* function and make it work for all basic types.
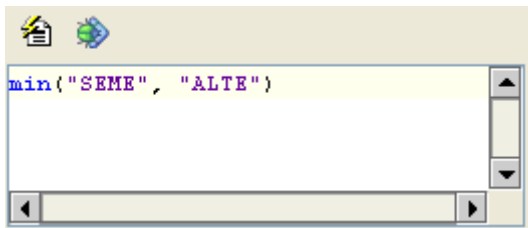
1. Change the Return type of the min function to 'Any'.

2. Change the type of the parameters val1 and val2 to 'Any'.

3. Since the body is not tied to a type we don't have to change it.
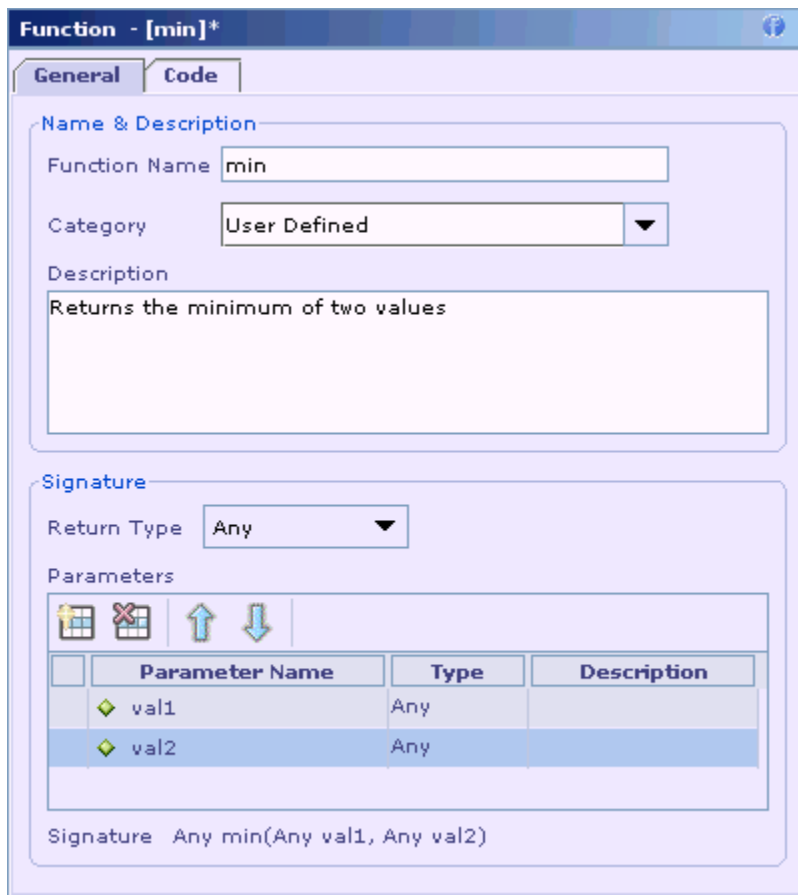
Now this function operates on two unspecified types and returns an unspecified type. Unlike C++ templates there is no way to specify (constrain) that parameter types should be same and the return type will be same as the parameter type.

Let us try to use this function and see how it works. Since the types are unspecified it is not possible to test just the body of the function. You need to invoke this function from elsewhere and check whether it works. When you invoke it with actual values, the types of the arguments are known. Using this information, it will synthesize a function definition with the known parameter types.
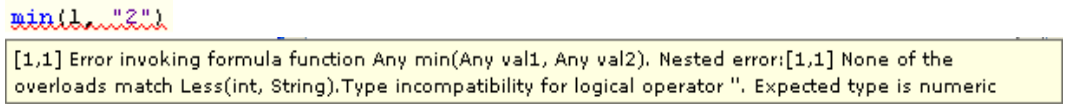
As before use a dummy function definition to test this function. Enter the code `min("SEME", "ALTE")`. We expect it to return the lesser of the two strings.

Use the Test dialog to verify this. Now change the code to min(12.5, 12.4) and again test it. You can see how the same function works for all the types that support the less than operator.

What would happen if you try to use it on a type that does not support it, for instance the boolean type; or if you mix incompatible types. Lets try it with an Integer and a String.

[1,1] Error invoking formula function Any min(Any val1, Any val2). Nested error:[1,1] None of the overloads match Less(int, String).Type incompatibility for logical operator ''. Expected type is numeric

Even as you type, the incorrect function call is underlined and the tool tip shows the error message. To do this, it has to replace the 'Any' parameter types with the invoked argument types (Integer & String) and check whether the body of the function 'min' is valid. So, errors if any, will actually occur in the body of the invoked (min) function. The error message shows the error at the call point as well as the error that occurred in the body of the invoked function (as nested error). You need this nested error information, if you have also written the actual function and want to fix its body. If you are interested in just using the function, you can ignore the nested error and treat the invocation as invalid.

How it works?

Parameterized functions work very similar to template functions in C++. During code generation, these functions are "instantiated" based on usage. By instantiation, we

mean that a method with the specific parameter and return types is generated. For instance, if the min function is invoked from two different places in the Cartridge with parameters of type Integer and String, then two overloads of the min method is generated in the chosen language.

If the min function is not used at all, code for it is never generated. This applies to non-parameterized functions as well; code is generated only for those functions that you use elsewhere in the cartridge. This feature can be very useful if you build a library of functions. Since you do not pay for what you don't use, you can include the full library and not worry about code bloat. For the two usages of min, the following code is generated.

```
public static String min(String _val1, String _val2) {
    return (TextFunctions.less(_val1, _ val2) ? _ val1: _ val2);
}

public static int min(int _ val1, int _ val2) {
    return ((_val1< _ val2) ? _ val1: _ val2);
}
```

In more complicated cases, the generated method name itself may be different for different instantiations of a function.

See Also:

Simple Function
Token based Functions
Testing the function
Formula

Token based functions

Generic function definitions let you create functions like min, which work on multiple types. But most of the functions you normally need that work on primitive types, are already bundled with the Designer Distribution. After exhausting such possibilities, you may wonder whether type parameterization is really useful. The real problems relate to usage of sections and its fields. For instance, you may have a piece of code that is used in number of places in say, mapping, except that the section and field names are different. Let us try and write a function secSumDouble1, which returns the sum of a double field in all the elements of a section.

Let us first implement a non-generic version.

Define a function secSumDouble1, with return type 'Double' and two parameters, 'section' of type Section and 'fieldName' of type String. The goal is to find the sum of the specified field in all elements of the section.

We have chosen the type 'Section' which means that a generic section object is passed to this method. In the body of the function you cannot directly access the fields of its elements; you need to use dynamic access, since the type of the section is not known. This function is not a generic function and only one copy of it is ever generated. This is the signature of this function.

Double secSumDouble1(Section section, String fieldname)

The body is as follows,

```
def sum = 0.0;
foreach($elm in section) {
     if(!isNull($elm, fieldName)) {
          sum += getDouble($elm, fieldName);
     }
}
return sum;
```

You can invoke this function as,

secSumDouble1(mySec, "myField")

As mentioned earlier, this is not specific to a section, but will work with any section as long the specified field is available. We have achieved this without using parameterized functions (Any type); but the cost is paid at runtime. Though, at call point, we know that the section has a field with the specified name, we don't directly access the field in the body, and instead we dynamically access it. Dynamic access has a cost associated with it.

The first step in generifying a function is to change the type of some of its parameters (whose type varies) to 'Any'. Since we want it work for specific section type, change the type of the section parameter to 'Any'. What about the fieldName? Should that be changed to 'Any' as well? fieldName is always going to be a string, but we will pass different values for it. For the time being let us retain it as String and look at the modified body.

```
def sum = 0.0;
foreach($elm in section) {
     if(isNull($elm.fieldName)) {
          sum += $elm.fieldName;
     }
}
return sum;
```

The main change is how the field is accessed from the element. Earlier we had used dynamic access; now we are using dot notation. This is much more efficient than earlier implementation.

But this does not work. We have used the notation $elm.fieldName to access a field. But fieldName is a variable (parameter) of type 'String'. When you use $elm.fieldName it would actually expect a field with name 'fieldName' to be present in the element. It will not use the value of the variable fieldName that you passed while invoking this function.

This is an important use case. If such a thing is not possible you have to resort only to dynamic access when you are working with sections and elements of sections. To support this case, there is type called 'Token'. If you select this type for a parameter, its value at invocation point is substituted for each occurrence in the body of the method. For instance if you invoke this function

secSumDouble2(mySec, "myField")

Then all occurrences of the token 'fieldName' are changed to 'myField' in the body of the function.

```
def sum = 0.0;
foreach($elm in section) {
    if(!isNull($elm.myField)) {
        sum += $elm.myField;
    }
}
return sum;
```

The invocation of this function is similar to what you would have done for a String Parameter, secSumDouble2(mySec, "myField"). The only restriction is that value passed should be literal and not a variable or other type of expressions. This is needed because the passed value has to replace the parameter name at design time in the function body.

This function is "instantiated" for every distinct section-fieldname pair. For instance, if you make two calls secSumDouble2(mySec, "myField") and secSumDouble2(mySec, "myAnotherField"), two copies of this function are generated. But if you again call secSumDouble2(mySec, "myField") from elsewhere a new copy is not generated. That is, the 'Token' type parameters, like 'Any' type participate in the instantiation of the function. A change in Token value will result a new copy being generated. In case of 'Any' typed parameter, a change in the argument type (not its value) will result in a new copy being generated.

Resources

The Resources design element allows you to separate constant values and [locale](#) specific messages from rest of the cartridge. In general, the Resources node allows defining name/value pairs called *resource items*. A resource item is used to represent either a constant or a message. When it is used to represent a constant, it behaves very much like a #define in C++ or a constant (public static final) in Java. When it is used to represent text messages, it resembles a resource bundle in Java or a RC in Windows development environment.

Separating or externalizing constants to a separate node/location has the following benefits:

The constant now has a meaningful name and description. This improves readability.

Your formula code is not littered with magic string/numbers; instead you use the name of resource item, which in general should be self-descriptive.

If a constant is to be used in multiple places, it is easy to manage its value at a single location. For instance, if the constant's value is redefined (say in the spec), it is easier to change it.

Externalizing messages to a separate node/location has the following benefits:

Resources node supports locale specific messages. You can provide the resource item's value for each locale you are interested in. This is the first step in [internationalizing](#) your cartridge.

Since the messages are listed in one place, a translator can easily localize, or translate into different languages.

All the user visible messages in your cartridge are listed together in one location. Even if you do not intend to support multiple locales, separating the messages can improve readability and documentation of your cartridge.

You can access a resource item by name in your formula code. At runtime or during code generation, the resource item name is substituted with its value. In case of a resource item representing a message, the appropriate value, based on the current

locale during runtime, is used. In case of a resource item representing a constant, the resource name is replaced with its value during code generation. The value is substituted either directly at the usage point, or is converted to a platform specific constant and referred.

See Also:

Resource Item
Resource Group
Working with Resources
Using a Resource

Resource Item

The Resources node allows you to define name/value pairs called resource items. The name of the item must conform to regular field name rules (no dots etc). Resource items are statically typed; all the Designer types are supported. The 'value' of the item should be of the specified type; that is, it should be possible to convert the value specified to the type of the item.

A resource item is used to represent either a constant value or a message. Designer supports the following resource items of the following types:

1. Simple Constants
2. List Type Constants
3. Messages

Simple Constant

Constants typically are magic numbers or strings, which have a business meaning. For instance, the character 'N' may be used to represent 'New Order'. These cryptic values appear in the input/output messages and are usually used as shorthand for the actual meaning. While processing a message, it is often required to interpret these values in the message. The following code, checks whether the status of the order is **filled**, and sets the destination field's value accordingly.

If(OrdStatus == '2', 0, ToInt(LeavesQty))

From the code it is not obvious what the magic character '2' means. What this value means is described elsewhere in the description of the field OrdStatus

```
Valid values for OrdStatus:
0 = New
1 = Partially filled
2 = Filled
```

```
3 = Done for day
4 = Canceled
…
E = Pending Replace (e.g. result of Order Cancel/Replace Request)
```

Though this information is available elsewhere, it does not make the formula code easier to read. You need to consult this description every time you read this piece of code. Also, since the same magic value needs to be used in number of such code fragments, it is possible to make a mistake and enter a wrong value.
The suggested approach is to externalize these magic values to the resource node and refer to it by a descriptive name in the formula.

If(OrdStatus == ORDER_STATUS_FILLED, 0, ToInt(LeavesQty))

Resource Name	Type	Value
◆ ORDER_STATUS_NEW	Character	0
◆ ORDER_STATUS_PARTIALLY_FILLED	Character	1
◆ ORDER_STATUS_FILLED	Character	2
◆ ORDER_STATUS_DONE_FOR_DAY	Character	3

Now it is possible to manage these values from a single location. A business analyst can quickly check whether the values used are correct and can change it if the business meaning of a constant changes.

Note:

Resource items of this type (constants) are typically not assigned locale specific values; the value is specified only for the default locale.

See Also:

Adding a Simple Constant
Using a Resource

List Type Constant

A resource item of this type associates a set of values to a resource name. Typically a resource item of this type is used to group related business constants into a single entity. For example, the resource name COUNTRIES might be used to represent a list of ISO country code values. This can then be used in an expression like **In(Country, COUNTRIES)**. In the picture given below, there are two list type constants: CURRENCIES and ALLOWED_TYPES.

Resource Name	Type	List	Value
◆ CURRENCIES	String	☑	ADP, ARS, AUD, BDT, BGL, BGN, BHD, BIF, BMD, BOB, BRL, BWP, CAD, CHF, CLF, CLP, CNY, COP, CRC, CYP, CZK, DJF, DKK, DZD,
◆ ALLOWED_TYPES	Integer	☑	1,2,3,4

Like a simple constant, the list type constant is accessed by name in a formula. The value is of list/collection type. You can iterate through the value of list using the **foreach** construct or use array operator to access a specific element. More commonly, you would use it as a parameter to the In() function to check whether a value is in the list.

During definition, a comma should separate the constants in the list. The comma-separated values are trimmed at both ends and hence you can leave leading/trailing spaces or you can break it up into number of lines if the list is long.

Note:

Like simple constants, resource items of this type are typically not assigned locale specific values.

See Also:
[Adding a List Type Constant](#)
[Using a Resource](#)

Messages

A resource item of this type is used for externalizing error messages or user visible texts. A message resource item is provided multiple values, one for each [locale](#) you want to support. At runtime, the value whose locale best matches the current locale is used.

Resources contain locale-specific objects. When you want to access a locale-specific resource item, a String object for example, you can load it from Resources that is appropriate for the current user's locale. In this way, the designer of the cartridge can write formula code that is largely independent of the user's locale isolating most, if not all, of the locale-specific information in Resources.

This allows designers to develop cartridges that can:

be easily localized, or translated, into different languages
handle multiple locales at once
be easily modified later to support even more locales

Typically resource items of this type are used in *action message* of validation rules, though it can also be used in any formula code. The action message in Designer defines the message associated with the exception that is thrown when a validation

fails. Since this exception's message is likely to be user visible, it needs to be presented in a language and format that the user understands. Designer has good support for this technique of externalizing locale specific resource, internationalizing them by supporting multiple locales and presenting them in localized form at runtime.

In many cases these messages are interspersed with values that are available only at runtime. For example, if you want to specify an action message that indicates an error in price value for a specified currency, it will be as follows:

"The amount value " + price + " for the currency " + currency + " has more than the expected number of digits"

Instead of one message string, we now have three parts. In general, it is not a good idea to internationalize such fragments for two reasons.

You have to externalize more number of strings (3 instead of 1)
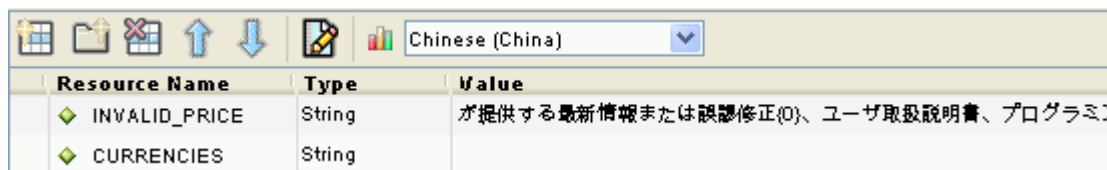
In another language, you may express the message completely differently. It does not make sense to translate the fragments.

The Format() function provides a means to produce concatenated messages in language-neutral way. Use this to construct messages to be displayed for end users. The Format() function takes a set of parameters, formats them and then inserts the formatted strings into the pattern at the appropriate places.

Format(INVALID_PRICE, currency, price);

Here the message pattern INVALID_PRICE is defined as a resource as given below:

The amount value {1} for the currency {0} has more than the expected number of digits

The message contains placeholders (zero based number), the value for which will be provided at runtime. The placeholders are substituted with the parameters provided to the Format() function. The placeholder {0} is substituted with first parameter (after the pattern), {1} with the second and so on.

The support for externalizing messages from the formula, along with simplicity of Format() function makes it easy to internationalize user visible error messages.

```
<?xml version="1.0" encoding="UTF-8" ?>
<TransformException>
    <Type>TransformException</Type>
    <Message>が提供する最新情報または誤認修正12.257、ユーザ取扱説明書、プログラミング ガイド
</Message>
    <ErrorCode>E1</ErrorCode>
    <Severity>error</Severity>
    <Cascadable>true</Cascadable>
    <FieldName>Price</FieldName>
    <FieldID>Price</FieldID>
</TransformException>
```

See Also:
[Adding a Message](#)
[Customizing Locales](#)
[Entering Locale Specific Message Pattern](#)

Internationalization

Internationalization is the process of designing an application so that it can be adapted to various languages and regions without engineering changes. Sometimes the term internationalization is abbreviated as i18n, because there are 18 letters between the first "i" and the last "n."

An internationalized program has the following characteristics:

With the addition of localization data, the same executable can run in multiple [locales](#).

Textual elements, such as status messages and error messages, are not hard coded in the program. Instead they are stored outside and retrieved dynamically.

Culturally-dependent data, such as dates and currencies, appear in formats that conform to the end user's region and language.

It can be localized quickly.

See Also:
[Messages](#)

Locales

On the Designer platform, a locale is an identifier for a particular combination of language and region and its variant. The runtime has a concept of default locale. This locale is derived based on the configuration of the operating system in which it is running. You can also customize it by starting the runtime with the appropriate command line arguments. For instance, to change the default locale, in case of Java

runtime, you can start the VM with the following additional properties.

```
java –Duser.language=fr –Duser.country=cn –Duser.variant=Quebec …
```

The above command sets the default locale to French Canadian.

The language code and the country code used are ISO defined two letter codes. As mentioned earlier, when you localize a [message](#), the message that is used at the runtime is based on the default locale. If a message value is not specified for the default locale, it tries to use the message specified by stripping off the variant (French Canadian). If it still can't find a message, it strips off the country part and uses the message defined for the language (French). If it again fails, it uses the message defined for the 'Default' locale. That is, for a resource item representing a message, the appropriate value is searched in the following order of locale:

1. Language + Country + Variant
2. Language + Country
3. Language
4. Default

See Also:

[Internationalization](#)

Resource Group

Often you would find that you want to define a set of related constants. For example, you may want to define the valid values for 'Order Status' as a separate set of constants.

Valid values for OrdStatus:
0 = New
1 = Partially filled
2 = Filled
3 = Done for day
4 = Canceled
...
E = Pending Replace (e.g. result of Order Cancel/Replace Request)

You need to come up with names for these constants. To avoid name clashes, it is very common to prefix the constant's name with group's name. So the constant's name would be ORDER_STATUS_NEW etc. Since this is a very common case, Designer supports a way to categorize such constants under a group. The group should have a unique name and each of the constants under it should have unique name within the group. It is okay for constants within the group to have the same as a top level constant or a constant under a different group.

Thus the main benefit of groups is that you get a different namespace to define your constants. In that sense they are similar to enums in Java and C++. The other benefit is that while viewing the constants in Resource UI, you can easily skip over groups of constants that you are not interested in (by collapsing the group).

ORDER_STATUS	Section	
New	Character	0
Partially filled	Character	1
Filled	Character	2
Done for day	Character	3
Canceled	Character	4

To access a constant defined under a group you need to use the dot notation. In above example, you should use ORDER_STATUS.New.

See Also:
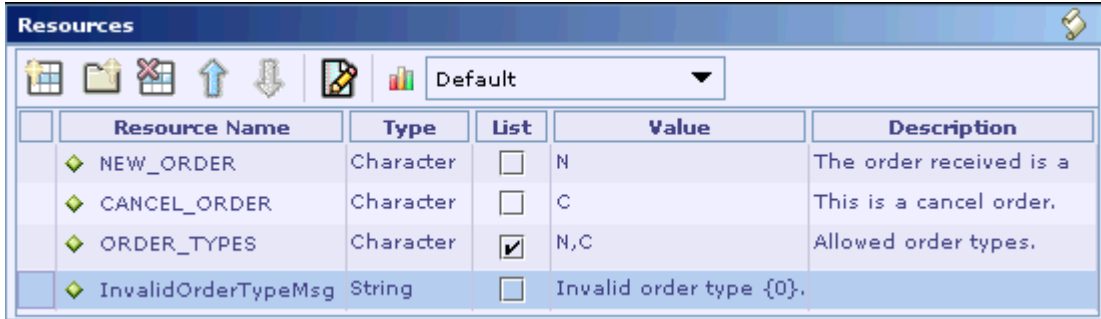
Adding Resource Group
Resource Item
Using a Resource

Working with Resources

This first step in using [Resources](#) is to create the Resources node. The Resources UI shown below is used to manage locales and resource items including locale specific messages.

	Resource Name	Type	List	Value	Description
◆	NEW_ORDER	Character	☐	N	The order received is a
◆	CANCEL_ORDER	Character	☐	C	This is a cancel order.
◆	ORDER_TYPES	Character	☑	N,C	Allowed order types.
◆	InvalidOrderTypeMsg	String	☐	Invalid order type {0}.	

After creating the Resources node, you can add, edit and remove [resource groups](#) and [resource items](#). You can also add/remove locales supported in the cartridge and [localize your messages](#).

See Also:

[Adding Resources](#)
[Adding Resource Group](#)
[Adding a Constant Resource Item](#)
[Deleting Resource Items](#)
[Arranging Resource Items](#)
[Customizing Locales](#)
[Entering Locale Specific Message Pattern](#)

Adding Resources

Follow the steps given below to create the Resources node:

1. Select the cartridge node.

2. Select the **Add Item** > **Add Resources** menu item from its shortcut menu.
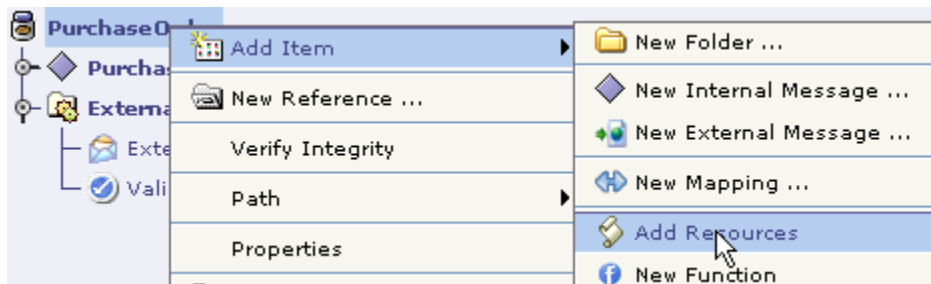
The Resources design element will be added as a child node of the cartridge node.

See Also:

[Working with Resources](#)
[Using a Resource](#)

Adding Resource Group

Follow the steps given below to add a new [resource group](#):

1. Click on the ![icon] icon in the Resources UI toolbar.
   A new row is added in the Resources table for the resource group.
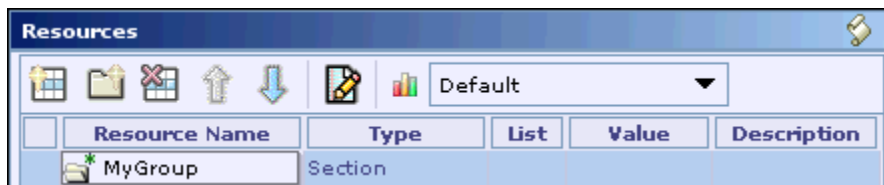
2. Enter the name of the resource group in the 'Resource Name' column.

3. Enter the description of the resource group in the 'Description' column.

Once you have added a resource group, you can easily add new resource items to that group by first selecting the resource group row and then clicking on the ![icon] icon.

See Also:

[Adding a Simple Constant](#)
[Adding a List Type Constant](#)
[Adding a Message](#)

Adding a Constant Resource Item

Designer supports two types of constant [resource items](#): simple constants and list type constants. The following sections explain the steps required to add resource items of these types.

Adding a Simple Constant

Follow the steps given below to add a new [simple constant](#) resource item:

1. Select the 'Default' item from the 'Active Locale' drop down list in the Resources UI toolbar, if it is not already selected.

2. Click on the icon in the Resources UI.

3. Enter the name of the resource item in the 'Resource Name' column.

4. Select the Designer type of the resource item from the drop down list displayed in the 'Type' column.

5. Specify the value associated with that resource name in the 'Value' column.

6. It should be noted that the value should correspond to the type of the resource item. Otherwise an error is thrown during validation.

7. Enter the description of the resource in the 'Description' column.

Adding a List Type Constant

Follow the steps given below to add a new list type resource item:

1. Select the 'Default' item from the 'Active Locale' drop down list in the Resources UI toolbar, if it is not already selected.

2. Click on the icon in the Resources UI.

3. Enter the name of the resource item in the 'Resource Name' column.

4. Select the Designer type of the resource item from the drop down list displayed in the 'Type' column.

5. Select the 'List' check box.

6. Specify the set of values associated with that resource name in 'Value' column.

7. The Comma character should be used to separate values in the list. It should be noted that each value should correspond to the type of the resource item. Otherwise an error is thrown during validation.

8. Enter the description of the resource in the 'Description' column.

See Also:

Adding a Message
Using a Resource

Adding a Message

A message resource item is used for externalizing error messages or user visible texts. Once added, a message can be localized for the locales supported in the cartridge.

Follow the steps given below to add a new message type resource item:

1. Select the 'Default' item from the 'Active Locale' drop down list in the Resources UI toolbar, if it is not already selected.

2. Click on the icon in the Resources UI.

3. Enter the name of the resource item in the 'Resource Name' column.

4. Select the 'String' as the Designer type of the resource item from the drop down list displayed in the 'Type' column.

5. Enter the message pattern associated with that resource name in 'Value' column.

6. Enter the description of the resource in the 'Description' column.

See Also:

Customizing Locales
Entering Locale Specific Message Pattern
Adding a Simple Constant
Adding a List Type Constant
Using a Resource

Deleting Resource Items

Follow the steps given below to remove a set of resource items.

1. Select the items to be removed.

    SHIFT-click in the selection column to select a set of continuous fields and CTRL-click in the selection column to select any non-continuous field without affecting the current selection.

2. Click on the ![icon] icon to remove the selected items.

See Also:

Working with Resources

Arranging Resource Items

Follow the steps given below to rearrange a set of resource items.

1. Select the items to be rearranged.

    SHIFT-click in the selection column to select a set of continuous fields and CTRL-click in the selection column to select any non-continuous field without affecting the current selection.

Click on the **Move Selection Up** ![up arrow] icon to move the selected items up by one row or the **Move Selection Down** ![down arrow] icon to move the selected items down by one row.

See Also:

Working with Resources

Customizing Locales

The user can add/remove locales supported in a cartridge by using the 'Available Locales' dialog. At runtime, the best possible locale is used for displaying the messages. Once a locale is added, the user can then proceed with localizing messages in that locale.

Adding a Locale

Follow the steps given below to add a new locale:

1. Click on the 'Customize Locales' icon 📊 present in the Resource UI toolbar.

   The 'Available Locales' dialog is displayed.
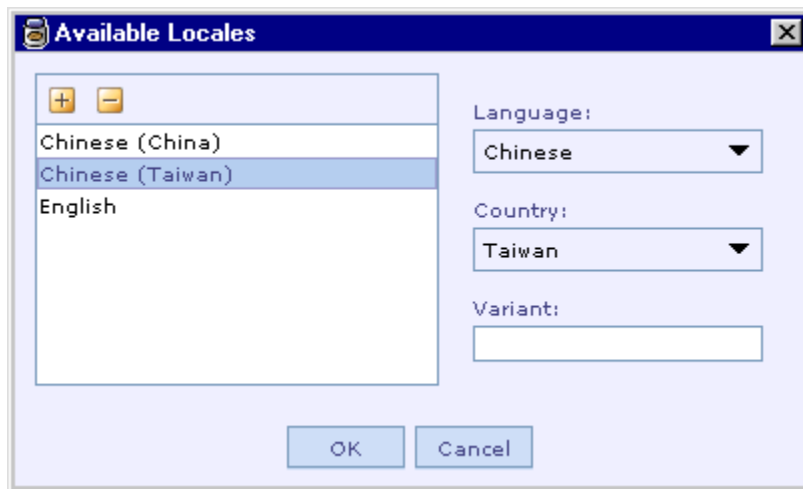
2. Select the language to be supported by from the corresponding drop down list.

3. Select the country to be supported by from the corresponding drop down list, if required.

4. Enter a variant in the corresponding text box, if required.

5. Now click on the 'Add Locale' tool icon ➕ to add that locale to the supported locales list.

Removing Locales

Follow the steps given below to remove a set of locales from the supported locales list.

1. Click on the 'Customize Locales' icon 📊 present in the Resource UI toolbar.The 'Available Locales' dialog is displayed.

2. Select the locales to be removed.

3. SHIFT-click to select a set of continuous items and CTRL-click to select any non-continuous item without affecting the current selection.

4. Click on the ⊟ icon to remove the selected locales.

See Also:

Adding a Message
Entering Locale Specific Message Pattern

Entering Locale Specific Message Pattern

Once a message is added using the 'Default' locale, the display text to be used for other locales supported in cartridge can be entered using the 'Edit Values' dialog displayed by first selecting the message to be localized in the Resources table and then clicking on the 📝 icon in the Resources UI toolbar.
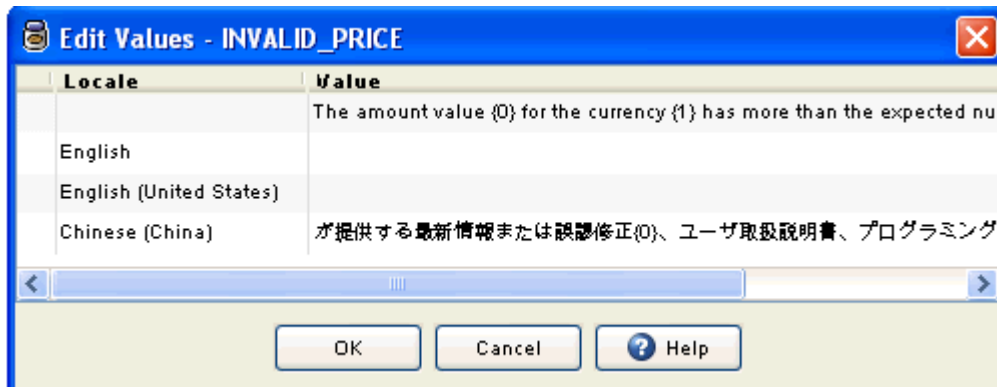
The names of locales supported in that cartridge are listed in the 'Locale' column and the first row displays the display text entered for the 'Default' locale. Now the translator can enter the display text for the other locales in the corresponding rows of the 'Value' column.

See Also:

Messages
Customizing Locales
Using a Resource

Using a Resource

Once a resource item is defined in the Resources UI, it is available globally. This means that it can be used from any design element of the cartridge just by using the name of the resource item. As mentioned before the resource name is substituted with its value wherever it is used.

To use a resource item, for e.g. in a formula, you can select it from the 'Resources' tab of the 'Edit Formula' dialog and then click on the 'Insert' button. If you move the

mouse cursor over the resource name when the CTRL key is depressed, a tool tip will pop up as shown below with its value and description.
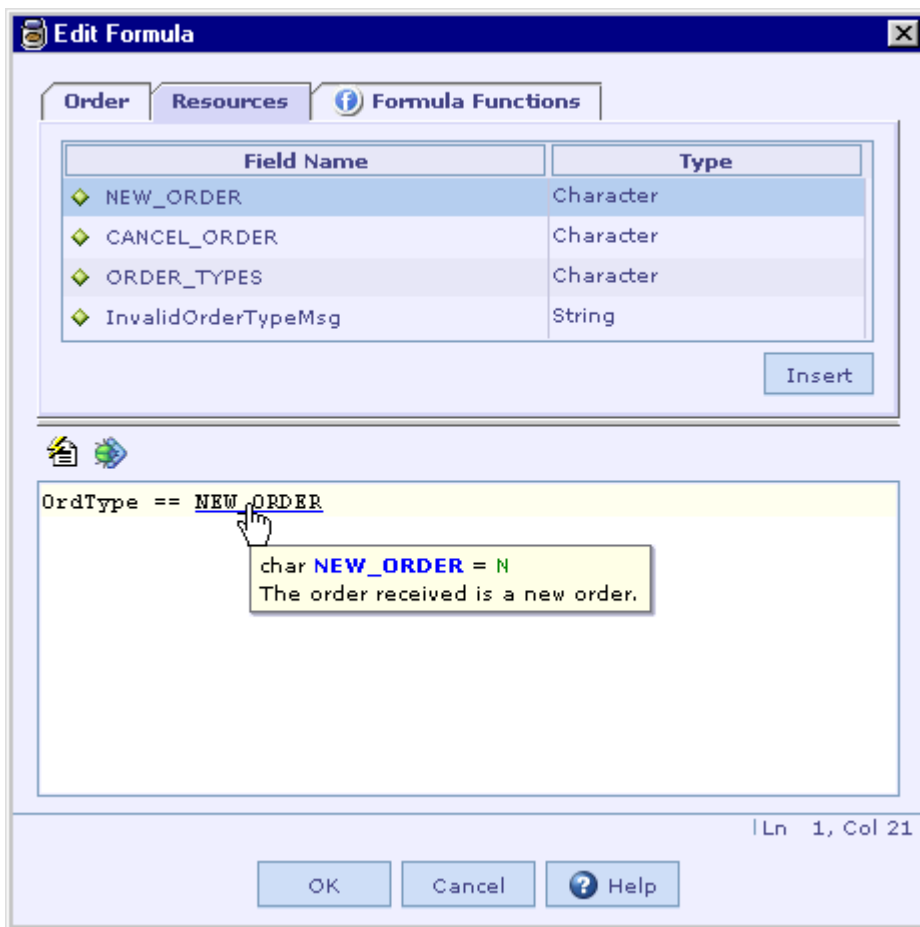
In the example shown in the above picture, the resource NEW_ORDER is of character type and it represents the value 'N'.

See Also:

Working with Resources

Cartridge References

In a team environment or in large projects you may want to partition your project into number of cartridges. For instance, messages can be composed in one (or more) cartridge, mappings in another and the message flow linking them in another. The goals are,

> Partitioning the application into number of cartridges, facilitates a team of developers/analysts to work on the same project. Each person can work on a separate cartridge, containing a subset of the project.

The cartridges used can be shared across projects. For instance, message definitions in a project may be useful in some other project as well. You can separate these messages into a separate cartridge and reference the cartridge in multiple projects.

When you breakup a project into number of cartridges, you need a way to refer to one cartridge from another. For instance, to work on mapping, you need the message definitions; if messages are defined in a separate cartridge, you need a way to refer to these message definitions from the cartridge in which the mappings are defined.

Cartridge references allow you to define a reference from one cartridge to another cartridge so that the entities defined in the referenced cartridge are accessible from the main cartridge. The entities include message definitions, mappings, message flows, function definitions and resource definitions.

Note

When a cartridge is included in another cartridge, it is marked as read-only. You cannot edit the entities in the referenced cartridge.

The entities defined in the referenced cartridge are accessible in the main cartridge. For instance, you can refer to a mapping defined in the included cartridge in a message flow of the main cartridge.

See Also:

Reference Links
Partitioning Your Application
Build Process
Best Practices
Adding Cartridge Reference
Removing Cartridge Reference

Reference Links

When you add a cartridge as a reference, Designer records the location of reference relative to the main cartridge. The contents of the cartridge are not copied into the main cartridge. The referenced cartridge is displayed as a node under a "References" folder. You can view the referenced cartridge by expanding this node. The referenced cartridge is marked as read-only and you cannot modify the entities defined in it.

References have the following implications,

Since Designer stores the relative location of the references, it is important that the cartridges are located under a common root folder. It is not recommended to have a cartridge under C:\mycartridges\myapp and refer to another cartridge under C:\globalcarts\. Such a configuration will tie you to one machine.

When you open the main cartridge, the referenced cartridge must be available at the same relative location. Otherwise the main cartridge will fail to open.

Concurrent modifications to the referenced cartridges are not immediately visible in the main cartridge (when it is open). You need to close the main cartridge and reopen to view the updated reference.

Referenced cartridges are not editable from the main cartridge; this applies even to its code generation settings. When you [build](#) the main cartridge, the referenced cartridges are built with the default or the saved code generation settings.

When you add a reference to a cartridge, the cartridges referred to by the reference are also treated as references to the main cartridge. That is, the entities available in the indirect references are also available from the main cartridge.

Cyclic references are not supported. That is, you cannot have a cartridge referring to itself directly or indirectly. For instance, the cycle, `A -> B -> C -> A` is not supported. Though cycles are not allowed, it is okay to refer to the same cartridge multiple times indirectly.

```
A.car
 |-> B -> C
 |-> D -> C
```

In the above case, the Cartridge C is indirectly referenced from A twice. This is allowed and in many cases is unavoidable. Though C is available twice, the entities defined in C are included just once.

Designer uses the absolute path of a cartridge to recognize that the cartridges are the same. In the above case, if you had made multiple copies of C.car and included them separately in B.car and D.car, it would have been treated as a different cartridge, leading to name collisions and type incompatibilities.

See Also:

[Adding Cartridge Reference](#)
[Removing Cartridge Reference](#)
[Cartridge References](#)

Partitioning Your Application

The way you partition your application has an impact on the way your development efforts and build process function in a team environment.

There are three main models to consider for partitioning solutions and projects.

1. Single Cartridge
2. Multiple interdependent cartridges
3. Multiple independent cartridges

See Also:

Cartridge References

Single Cartridge

All the entities in the project are defined a single cartridge. The cartridge is self-contained and has everything it needs to build and execute it. The cartridge, for all purposes, represents the application.

Advantages

The single cartridge model offers the following advantages:

1. Managing the cartridge in source control, building the cartridge, etc. is all very simple.

2. It is ideal for simple projects.

Disadvantages

1. The entities defined in the cartridge cannot be easily reused in another project. You can copy and paste entities across cartridge, but that may lead to maintenance problems.

2. It is difficult for more than one person to work on the application at the same time. It is possible to diff and merge concurrent changes in the cartridge, but it is not recommended.

See Also:

Partitioning Your Application

Multiple Interdependent Cartridges

The entities in the project are defined in more than one cartridge. The higher-level cartridges, which define mappings, message flows etc, depend on cartridges that provide definitions for the messages and functions. Typically, there is a main cartridge that defines the application's main entry point (for e.g a message flow). This cartridge directly or indirectly references all the cartridges in the application. Building the main cartridge is sufficient to build the entire application.

Advantages

The multiple interdependent cartridges model offers the following advantages:

1. Referenced cartridges, which contain message definitions, function definitions can be shared across multiple applications.

2. It is possible for a team of developers/analysts to work on the application. Each person can work independently on a part of the application, by separating it into a cartridge.

See Also:

Partitioning Your Application

Multiple Independent Cartridges

The entities in the project are defined in more than one cartridge. These cartridges are essentially independent of each other. It is possible that some of these cartridges depend on other cartridges; but they do not depend on each other. That is, there is no main cartridge, which directly or indirectly references all other cartridge.

To build the application, you need to build all the top-level cartridges separately. The application now contains all the assemblies of the top-level cartridges and their dependencies. If these top-level cartridges depend on the same lower level cartridge, you need to ensure that assembly of the lower level cartridge is included just once in the final application.

Advantages

There are no particular advantages in this model. It is similar to multiple interdependent cartridges model.

Disadvantages

The build process is more complicated. You can solve this problem by defining a main cartridge, which simply references all the top-level cartridges. This would make similar to multiple interdependent cartridges.

See Also:

Build Process

When you build a cartridge, all other cartridges added as references to it are automatically built (if needed). This includes direct and indirect references.

1. It first computes the set of all cartridges that need to be built. It does this by starting from the main cartridge and adding its references to a set and so on. If a cartridge is referenced multiple times directly/indirectly, it occurs just once in the set.

2. The build order is identified by using the cartridge dependency information. Since cyclic dependency between cartridges is not supported, it always leads to a predictable and safe build order.

   The cartridges with a dependency (references) are built only after the references are built. All the cartridges (including references) are built in exactly the way they would have been built separately. The outputs produced are also identical. For instance if the main cartridge reference to cartridge A1, A1 is built as part of the build process of main, and the platform specific assemblies generated for A1 are identical to the case where A1 is built separately. Just like the Main.car references A1.car, Main.assembly will also depend on (refer to) A1.assembly. That is, dependency relationship between the assemblies will be similar (if not identical) to the dependency relation between the cartridges.

   ```
   Main.car    -> Main.assembly
   A1.car      -> A1.assembly
   A2.car      -> A2.assembly
   ```

3. As an optimization, if the referenced cartridges have already been built and are up to date, the build process, excludes it from the current build. This can lead to considerable saving in time.

Note:

We have used the term "assembly" to refer to a platform specific package of classes that we generate. It is Jar in case of Java.

See Also:

[Partitioning Your Application](#)
[Cartridge References](#)

Executing Cartridge with References

When you [build cartridges](#) that refer to other cartridges, it results in number of assemblies; one for the main cartridge and one each for the referenced cartridges. This section explains how to execute the generated assemblies in various platforms.

Simulator

You don't have to do anything special to execute the main cartridge in the [Simulator](#). When you start the Simulator, it automatically deploys all the assemblies of the main cartridge as well as the dependent assemblies that were generated while building the referenced cartridges. In effect, the entities displayed in the Simulator's drop down include all the entities that are contained in the main as well as those in referenced cartridges. This behavior is identical to having a cartridge that is a union of all cartridges directly or indirectly referenced (contains all the entities defined in the referenced cartridges). This behavior is the same in the Simulator of all the platforms.

Java/EJB Platform

Simple Runtime

Add all the Jars that were generated during the build process, including those for the referenced cartridges in the classpath.

EJB Server

Deploy all the Jars that were generated including those for the referenced cartridges. All the Jars should be specified as EJB module in the application.xml file. They should all be available in the application directory.

In the code generation settings, you have an option to generate EAR file. If you have selected this option, the generated EAR would automatically include all the generated jars (including for references). It will also include dependent runtime libraries that are used by the generated Jars. In short, the EJB Jar includes all required dependencies and is ready to be deployed.

Best Practices

1. Each [cartridge](#) should always be in a directory of its own. The code generator creates a working directory under the cartridge directory (such as java), to generate source files and build to it. Having multiple cartridges in a single directory would mean that this code generator directory would be shared. This can lead to a situation where generated files of the second cartridge overwrite some of the files of the first.

2. If your application contains multiple cartridges, keep all of them under (not directly) an application folder. Since cartridges point to each other by their relative location, it would be safe to xcopy (or zip) the entire application folder, say, to another machine.

```
AppFolder
  |
  |--- Car1Folder
  |      |
  |      |--- Car1.car
  |
  |--- Car2Folder
  |      |
  |      |--- Car2.car
```

   Other nested configurations are also possible; the key is to have a root folder, which contains everything. It is not recommended to have a cartridge under C:\mycartridges\myapp and refer to another cartridge under C:\globalcarts\. Such a configuration will tie you to one machine.

   This directory structure is very important if you want to maintain your cartridges under source control.

3. The cartridge name should preferably be same as the cartridge file name. By "name of cartridge" we always mean the name you have specified in the cartridge and not its file name. It is a good idea to keep these names the same to avoid confusion.

4. Do not define [messages](#)/[mappings](#)/flows with same name, either in the same cartridge or in cartridges that are likely to be used in the same application. At runtime, these entities can be looked up from a global LookupContext. This means that any conflict in names can be disastrous. Designer detects such name

collisions in the same cartridge or in referenced cartridges during validation. But, since you can assemble an application from multiple cartridges, it is your responsibility to name them uniquely. One solution is to avoid using generic names like 'NewOrder'; instead use a qualified name like "FIX41NewOrder".

5. For applications that require [internationalization](#), it is a good idea to define the localized messages as [resource](#) in a separate cartridge and refer to it from all other cartridges. That way it becomes simple to add a new language/locale without changing several application cartridges.

6. If you have number of [functions](#) that are shared by many cartridges, then it is a good idea to separate the functions into a separate function library cartridge.

7. Do not define more than one [function](#) with same name and same number of parameters and with similar or conflicting parameter types. For instance defining functions f(int) and f(Any) can lead to ambiguities. Note that, currently Designer does not detect this ambiguity and binds to one of these functions.

8. Unit & Integration testing. The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as you expect. Each unit is tested separately before integrating them into modules to test the combined functionality. In case of Designer, the units are the messages and mappings. Make sure that you test each message definition separately before you define a mapping for it. It is possible to test a message definition (in the Simulator as well as from command line) by passing raw input (say in the form of a file). Record the outputs of such tests and make sure that output remains consistent after you make changes to the message definition or upgrade to a new version of Designer. This same approach must be followed for mappings. Any change in the output points to a regression in the definition.

    Message flows are typically used to integrate, the lower level units. Integration testing identifies problems that occur when units are combined. By using a test plan that requires you to test each unit and ensure the viability of each before combining units, you know that any errors discovered when combining units are likely related to the integration (message flow) itself. This method reduces the number of possibilities to a far simpler level of analysis.

See Also:

[Partitioning Your Application](#)
[Adding Cartridge Reference](#)
[Removing Cartridge Reference](#)

Adding Cartridge Reference

To add a reference, right-click the Cartridge node and select 'New Reference' menu item from the short-cut menu that appears.
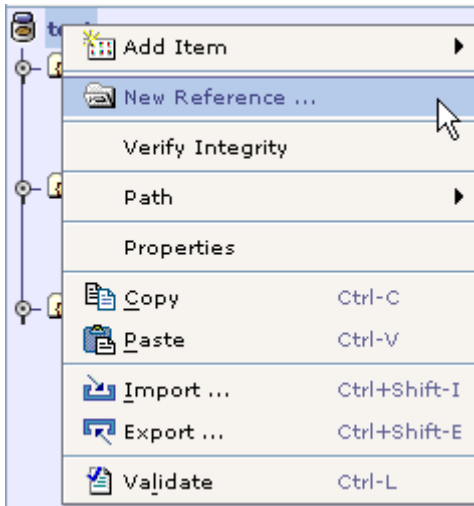
A file open dialog is displayed where you can select the cartridge file to be included.
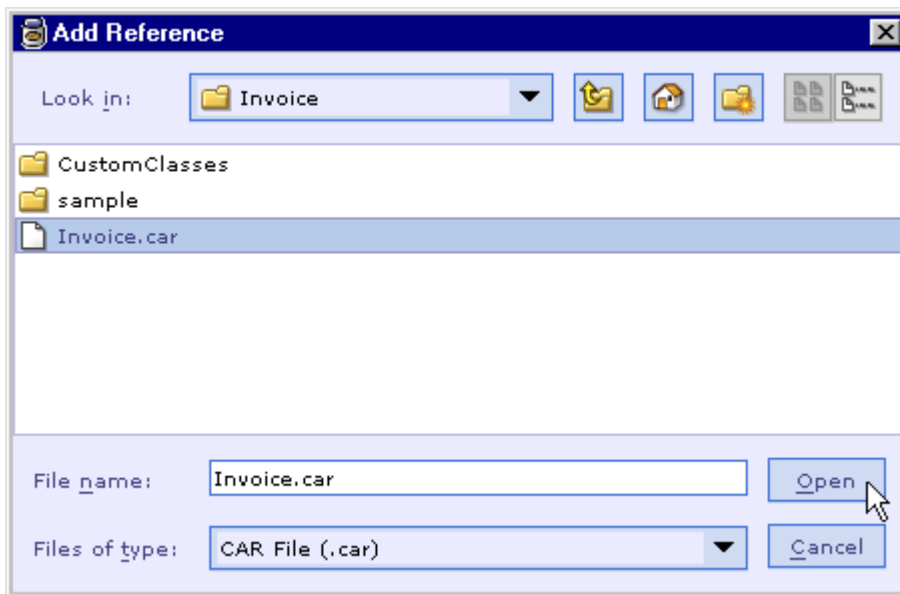
The selected cartridge is added under a 'References' folder as shown below.
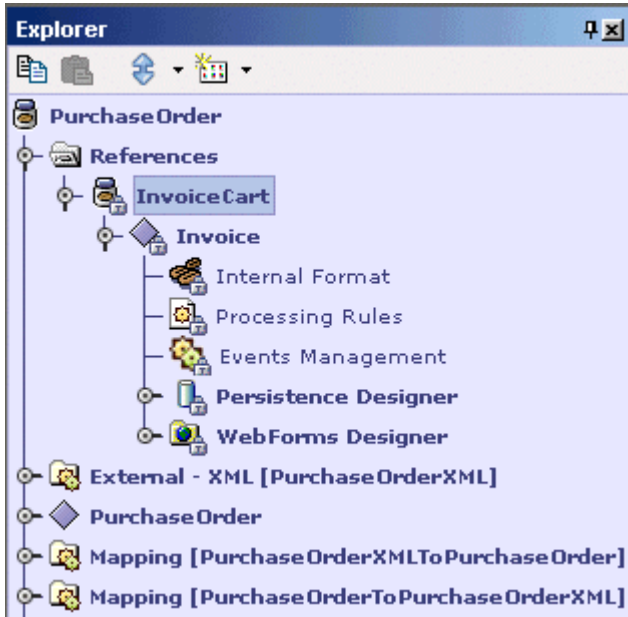
Note that read only nodes are shown with a lock icon overlaid on top of the nodes icon. The lock indicates that it cannot be modified.

See Also:

Refreshing Cartridge Reference
Removing Cartridge Reference
Best Practices
Cartridge References

Refreshing Cartridge Reference

The 'Refresh References' menu item when selected, updates the main cartridge for the changes made in the reference cartridge. To update a change made in the reference cartridge, right click the references node and select 'Refresh References' menu item.

You can refresh a referenced cartridge by following the steps below.

1. Open the reference cartridge and make the changes and save.

2. Open the main cartridge, right click the 'References' node and select 'Refresh references' menu item.

The changes made in the reference cartridge will be updated in the main cartridge. Selecting the 'Refresh References' menu item does not affect the mapping or any of the referenced cartridge nodes usage in the main cartridge.
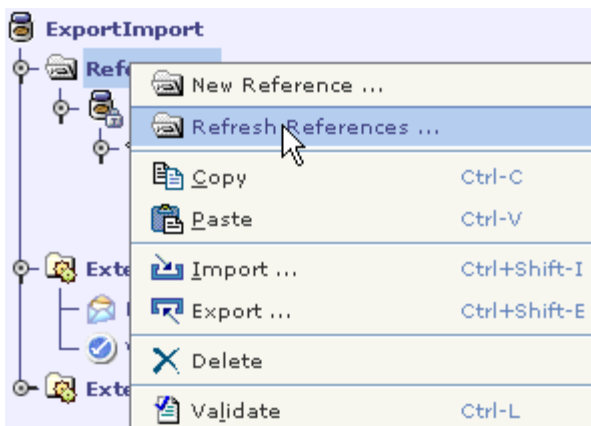
See Also:

Removing Cartridge Reference
Best Practices
Cartridge References

Removing Cartridge Reference

You can delete top-level references by following the steps given below.

1. Right Click the cartridge node of the top-level reference to be deleted and select 'Delete' menu item.

2. Click 'Yes' in the delete message box that appears to remove the cartridge reference.

See Also:

Adding Cartridge Reference
Best Practices
Cartridge References

Fixing Broken References

When you open a cartridge with broken references, Designer allows you to fix them.

Follow the steps given below to fix broken references in the cartridge being opened.

1. For each broken reference, Designer will display the 'Cartridge Reference Error' message box and ask you whether you wish to fix it.
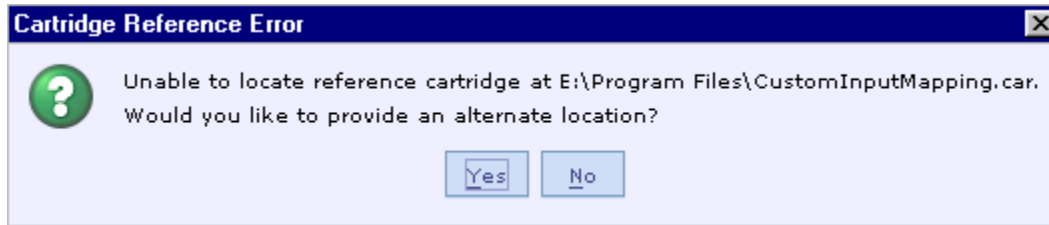
Cartridge Reference Error

? Unable to locate reference cartridge at E:\Program Files\CustomInputMapping.car.
   Would you like to provide an alternate location?

   [Yes]   [No]

2. If you select the **Yes** button, it will show you the 'Resolve Cartridge Reference' dialog.

Resolve Cartridge Reference

Look In: Main

sample
MessagesAndFlows.car

File Name:

Files of Type: Cartridge Files (*.car)

[Open] [Cancel]
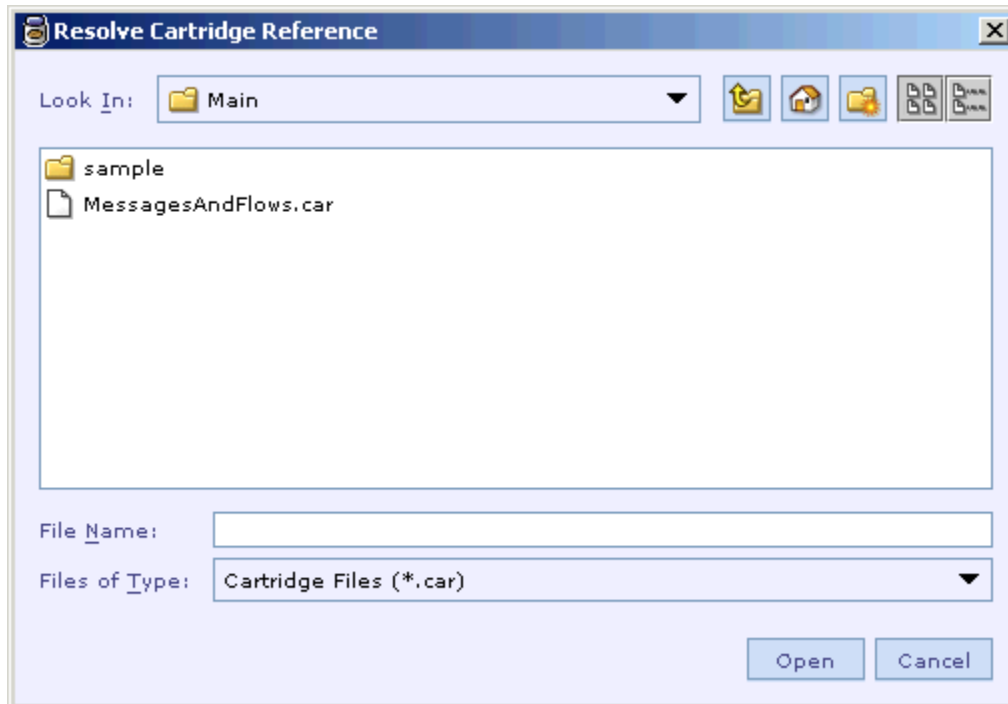
3. Navigate to the directory that contains the cartridge corresponding to the broken link and select it.

4. Click the **Open** button.

5. When all the broken links are fixed, the main cartridge will be opened in the Designer window.

When you save the cartridge, Designer will save the new locations selected for the broken references.

See Also:

Best Practices
Cartridge References

Code Generation

Once a cartridge design is completed, it can be [generated into platform specific code](#) based on the specified [code generation settings](#). Currently [Designer](#) supports code generation in Java/EJB. The generated code is bundled into components as specified in the code generation settings. These components can then be [deployed](#) under platform specific Runtime. Once deployed, the Runtime becomes capable of performing transformations defined by that cartridge.

Note

Your Designer installation may not contain support for all these platforms.

See Also:

[Selecting the Default Platform](#)
[Designer User Interface](#)
[Cartridge](#)
[Message](#)
[Message Mapping](#)
[Formula](#)
[Function Definition](#)
[Resources](#)
[Cartridge References](#)
[Simulator](#)
[Working With Cartridge Designer](#)

Code Generation Settings Dialog

The 'Code Generation Settings' dialog allows you to specify the information used in generating the platform specific components. This dialog for the [default platform](#) can be invoked by selecting the **Build** > **Code Generation Settings** menu item or by clicking on the **Code Generation Settings** button 🎨 in the main window toolbar.

See the section [Java/EJB Code Generation Settings Dialog](#) for information on settings used for generating Java/EJB components.

Java/EJB Code Generation Settings Dialog

The settings to be used for generating Java/EJB components is categorized into the following tabs:

[General Tab](#)
[Code Generation Tab](#)
[Language Bindings Tab](#)

Each tab consists of a set of related settings. You should select a tab and then change the settings given in that tab as required. When finished, click on the OK button to accept the changes made to the settings. Clicking on the Cancel button will reject the changes made. Please note that you should save the cartridge to permanently save the changes.

See Also:
Generating a Cartridge
Deploying a Cartridge
Code Generation

General Tab (Java/EJB)

This tab allows you to specify compilation settings that are not specific to the cartridge but to the current instance of Designer.
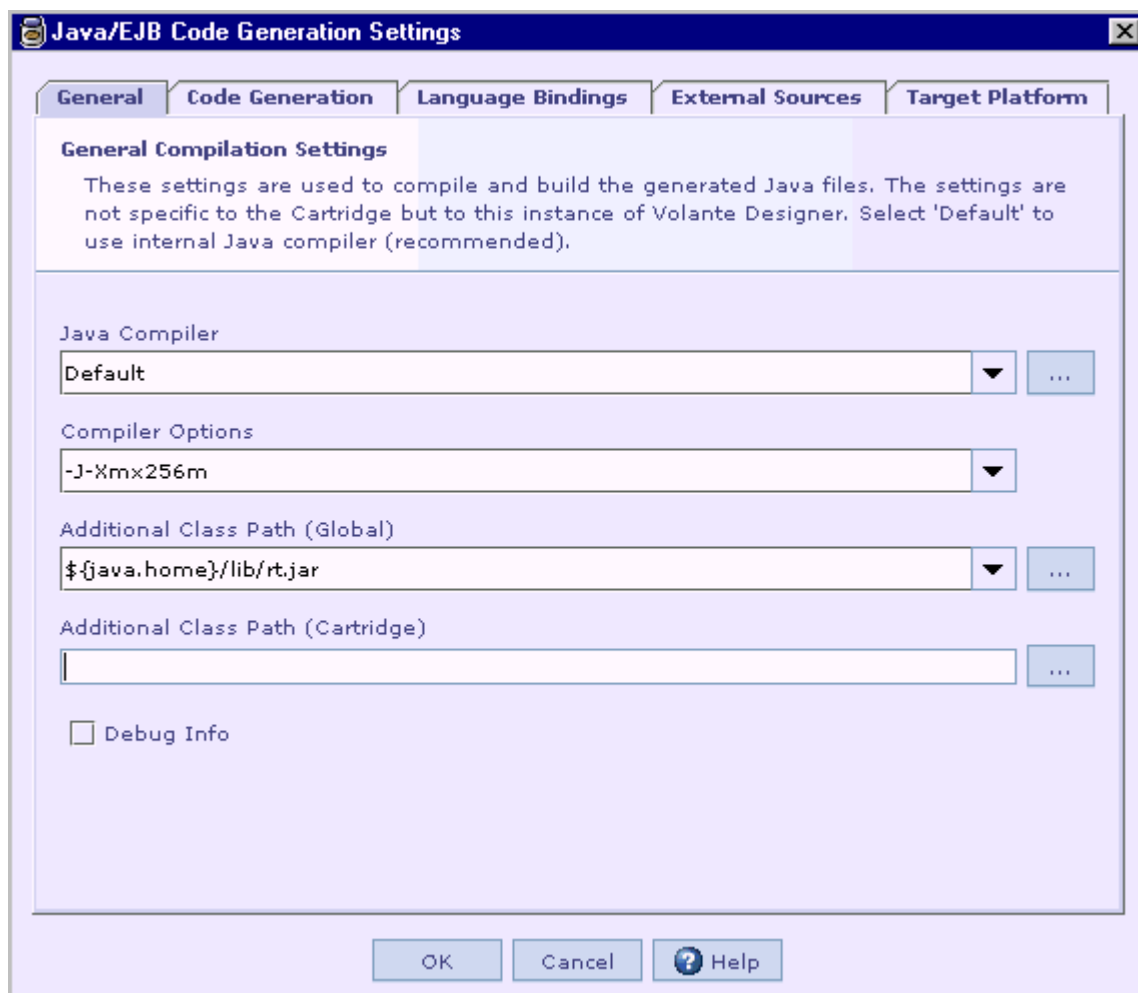
Java Compiler

Here you specify the Java compiler to be used for compiling the Java source files, both the generated and the external Java files.

Select the 'Default' option to use the internal Java compiler (recommended).

Compiler Options

This is used to specify the extra compiler options.

Additional Class Path (Global)

Occasionally you may need to specify additional classpath to compile your external sources, during code generation. This is needed if your source files depend on some other custom or third party Jars. This option is available at global level (Designer instance) and also on a per cartridge basis. This classpath setting is applicable for all cartridges. The other classpath setting 'Additional Class Path (Cartridge)' is applicable only for the cartridge in which it is specified.

Please note that specifying Jar files in the 'Additional Class Path' setting makes them available only to Designer, but not to the runtime. See the Additional Modules section for adding a Jar file as a library file of the generated EAR file.

Additional Class Path (Cartridge)

This is used to specify the path of the Jar files to be included in the classpath during compilation of the Java source files. This classpath setting is applicable only for the cartridge in which it is specified. The other classpath setting 'Additional Class Path (Global)' is applicable for all cartridges.

Please note that specifying Jar files in the 'Additional Class Path' setting makes them available only to Designer, but not to the runtime. See the Additional Modules section for adding a Jar file as a library file of the generated EAR file.

See Also:
Code Generation Tab
Language Bindings Tab
External Sources Tab
Target Platform Tab
Code Generation Settings Dialog

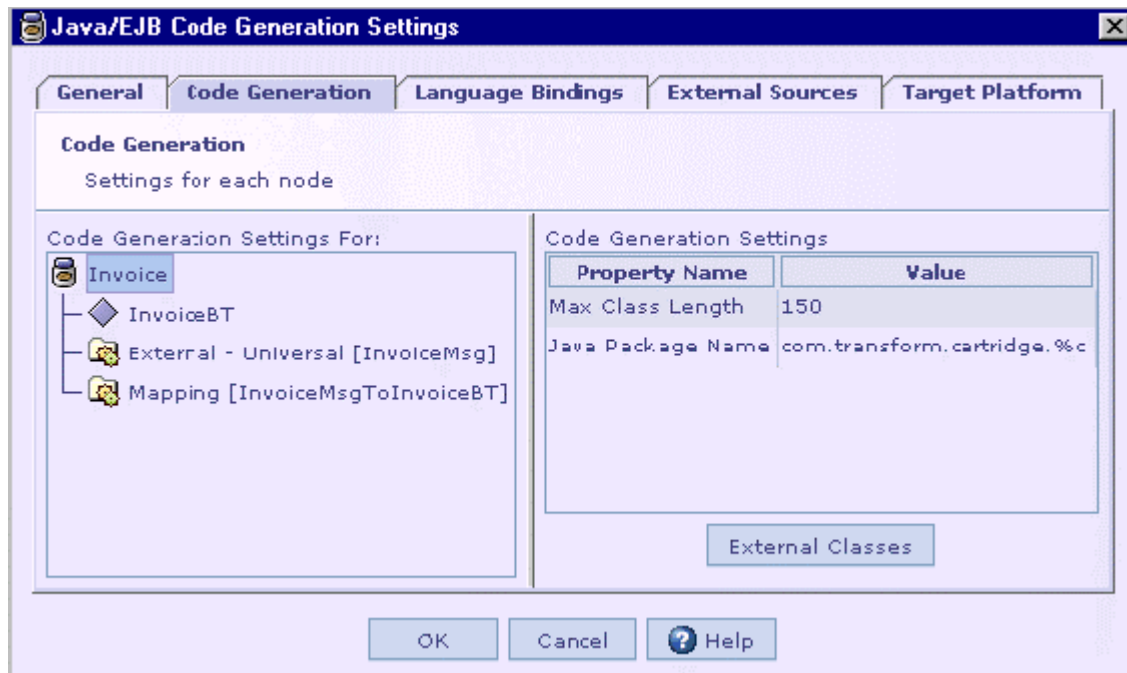# Code Generation Tab (Java/EJB)

This tab allows you to specify settings related to the generated components.

The left pane of this dialog displays design elements of the cartridge in a tree structure and the right pane displays a list of properties corresponding to the design element selected on the left pane. The properties displayed for a design element differ based on its type.

Please note that if the cartridge contains references, the referenced cartridges and their child elements are displayed in the tree structure on the left pane. Selecting a design element of a referenced cartridge will display its properties on the right pane, but they are read only. So you cannot change the properties of a referenced cartridge from the main cartridge that references to it. The cartridge used as a reference should be opened separately to make changes to its code generation settings.

Java Package Name

Here you specify the Java package name to be used for the generated Java source files.

Jar Name

This property is displayed only for the design elements such as messages, message mappings, message flows and persistence designer, which will be generated into components. It is used to specify the name of the Jar file into which the component generated for the current design element will be bundled.

Please note that you can specify the same Jar name for multiple design elements or use a different Jar name for each of the design elements. In case of using the same Jar name for multiple design elements, the components generated for these design elements will be bundled into a single Jar file.

By default, this property is set as '%c.jar'. Here, %c represents the cartridge name; that is, the name of the root design element in the cartridge. This is just a sensible default and you can always change the Jar name of each component as required.

Manifest Entries

This property is displayed only for the design elements, which will be generated into components. Here you specify Jar files to be included in the runtime class path of the generated component.

Please note that Jar file names specified in this property are used in generating the manifest file of the generated component and it does not make the Jar files available to the runtime. See the [Additional Modules](#) section for adding a Jar file as a library file of the generated EAR file.

Max Class Length

This is a cartridge level setting that applies to all the design elements of the cartridge. It allows you to specify the maximum length of the generated Java class names. The name of classes exceeding this limit will be mangled to a shorter name.

The default value of this property is 150 and the upper limit is 255.

Data Source

This property is displayed only for the design elements, which will be generated into components. Here you specify the data source reference name used in the corresponding design element. The reference name should be bound to the actual server resource. See the section [Resource References](#) for more information on specifying server specific bindings.

Generate MDB

This property is displayed only for a message flow design element. Set this property to 'true' to generate a Message Driven Bean (that will listen for client requests in the form of JMS messages) in addition to generating the normal session bean for the corresponding message flow design element. Setting this property to 'false' will generate only the session bean.

Unique Key Table

This property is specific to persistence designer design element. It is used to specify the name of the table that will be used for generating unique key values.
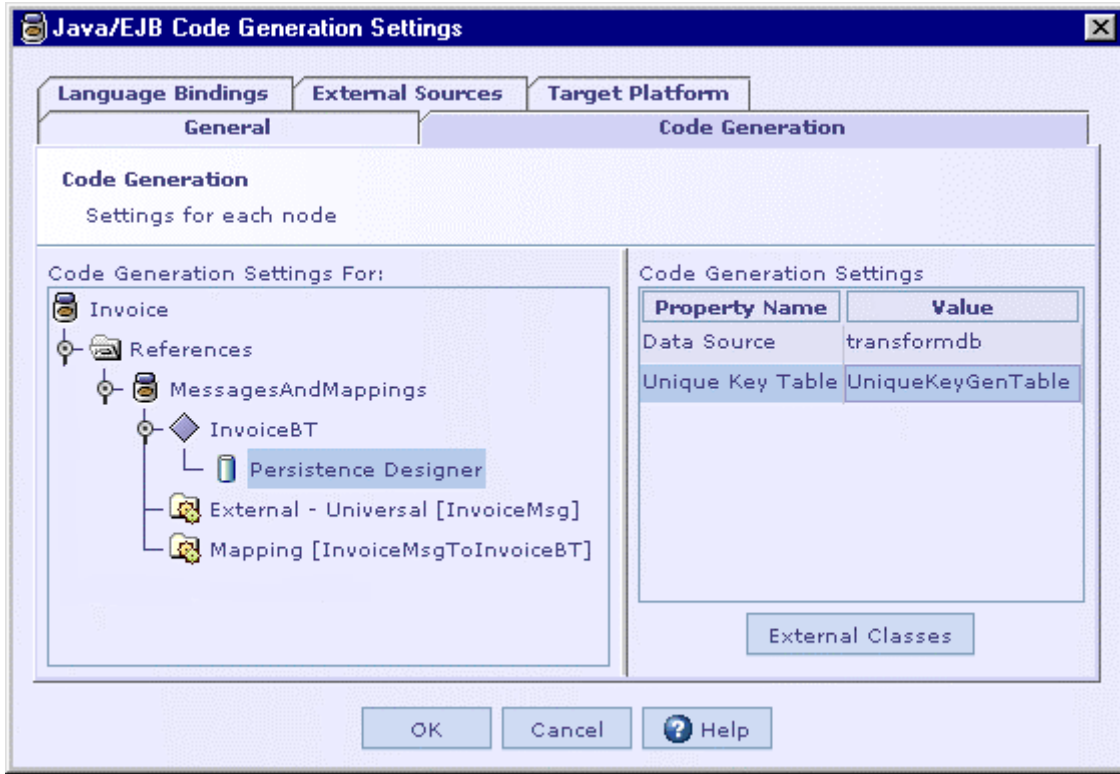
Adding External Java Classes

You can specify external Java class files (not Java source files) to be bundled with the component generated for a design element by following the steps given below.

You can include external Java classes to the code generation settings of a cartridge in two other ways:

You can bundle the required Java class files into a Jar file and add it to the manifest entry of the generated component.

You can add Java source files (instead of Java class files) to the code generation settings of the cartridge, which will be compiled and the resultant class files will be bundled with the generated component. See the section Adding External Java Source Files for more information.

See the section Language Bindings Tab for information on how to bind class name references used in a cartridge to the actual Java classes.

Steps to add external Java class files to the code generation settings of a cartridge:

1. Compile the Java source files into Java class files.

    It is recommended that the compiled Java class files are available under a child directory of the cartridge directory.

2. In the 'Code Generation' tab of the 'Java/EJB Code Generation Settings' dialog, select the required design element in the tree structure displayed on left pane.

3. Click on the 'External Classes' button at the bottom of the right pane.

4. The 'External Classes' dialog will be displayed.
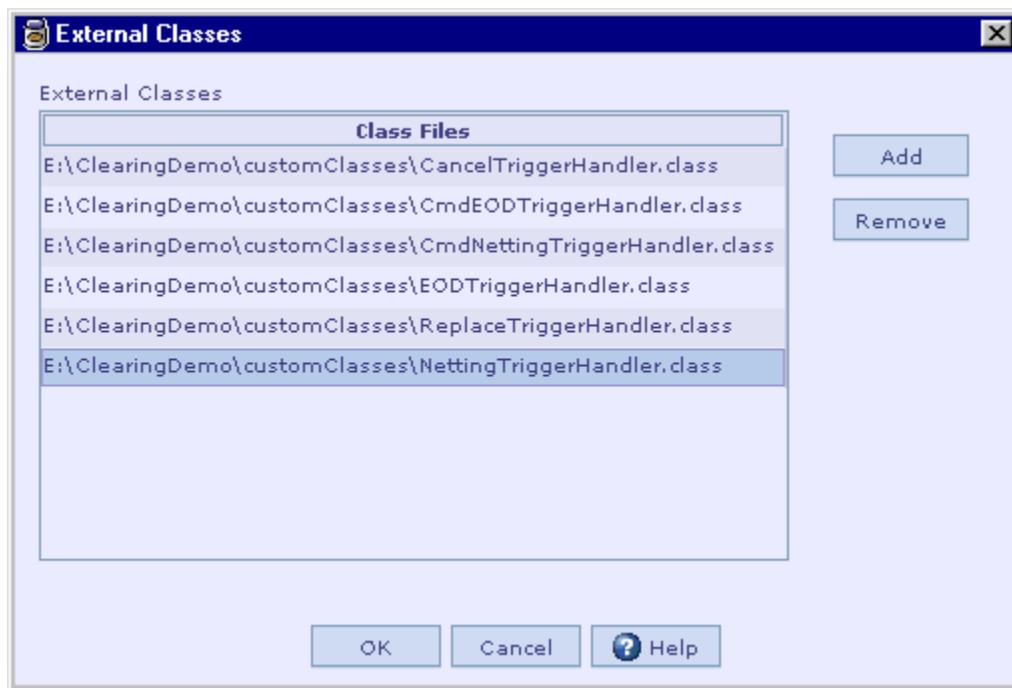
5. Click on the 'Add' button.

6. The file Open dialog will be displayed.
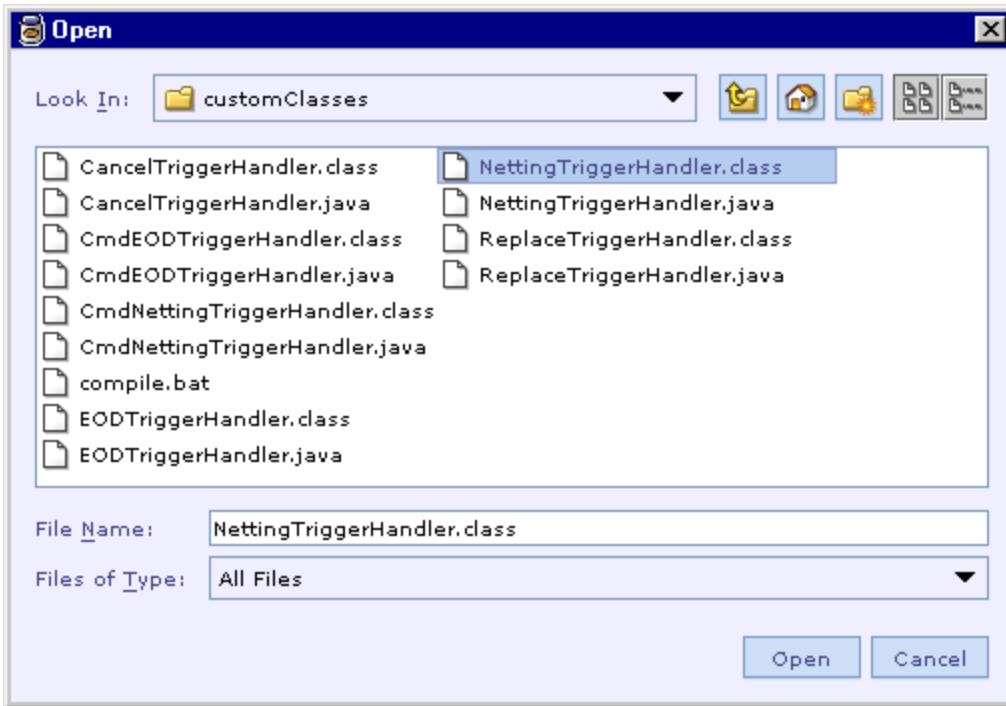
7. Navigate to the directory that contains the required Java class files and select them.

8. Click the 'Open' button.

The files will be included in the 'Class Files' list of the 'External Classes' dialog.

Note:

To remove class files from the 'Class Files' list, select them from the list and then click on the 'Remove' button of the 'External Classes' dialog.

See Also:
General Tab
Language Bindings Tab
External Sources Tab (Java/EJB)
Target Platform Tab
Code Generation Settings Dialog

Language Bindings Tab (Java/EJB)

This tab allows you to bind class name references used in a cartridge to actual Java classes. These classes are used for customizing some processing such as validation of a field, mapping between two messages, etc. These classes, along with the helper classes if any, should be included to the code generation settings of the cartridge before you proceed with code generation and deployment.

You can include external Java classes to the code generation settings of a cartridge in three ways:

You can specify the Java class files (not Java source files) to be included in the generated component. See the section Adding External Java Classes for more information.

You can bundle the required Java class files into a Jar file and add it to the manifest entry of the generated component.

You can add Java source files (not Java class files) to the code generation settings of the cartridge, which will be compiled and the resultant class files will be bundled with the generated component. See the section Adding External Java Source Files for more information.

Please note that class name references can be specified for the following items:

Validation rule of an internal/external message field
Message mapping of all types
Processing of an internal message
Processing of an internal message field
'Invoke External' activity of a message flow

All the class name references used in the cartridge are listed under the 'Reference' column of this tab. For each of these references, you have to specify the actual Java class name in the 'Java Class' column of the corresponding row. By default the reference name itself is displayed as the Java class name. You have to change it to the actual Java class name along with the package name prefix.
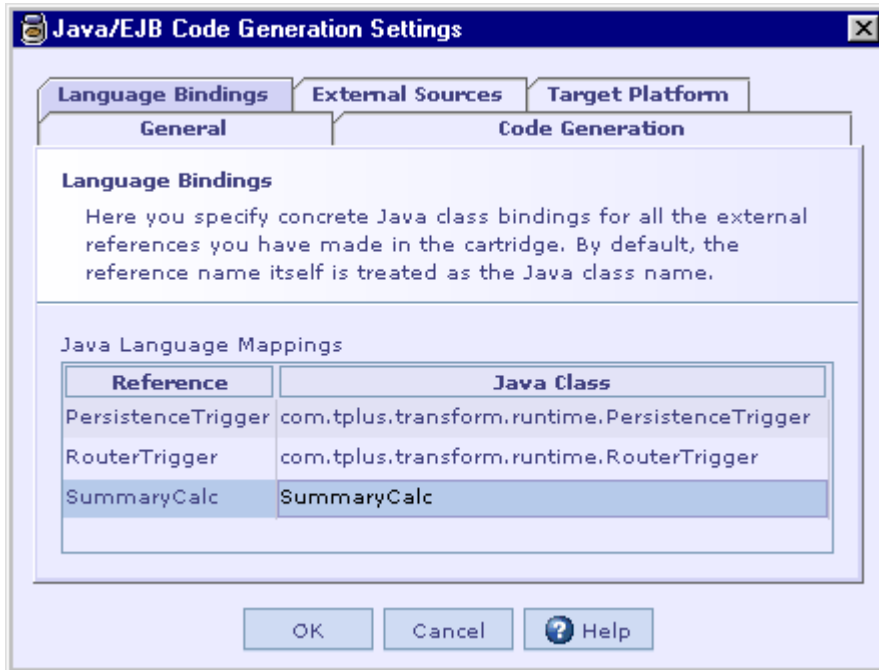
See Also:
General Tab
Code Generation Tab
External Sources Tab
Target Platform Tab
Code Generation Settings Dialog

External Sources Tab (Java/EJB)

This tab allows you to add external Java source files (not Java class files) to the code generation settings of a cartridge so that they are compiled and the resultant class files are bundled with the generated component.

You can also specify a directory containing Java source files so that they are compiled and the resultant class files are bundled with the generated component.

You can include external Java classes to the code generation settings of a cartridge in two other ways:

You can specify the Java class files (instead of Java source files) to be included in the generated component. See the section Adding External Java Classes for more information.

You can bundle the required Java class files into a Jar file and add it to the manifest entry of the generated component.

See the section [Language Bindings Tab](#) for information on how to bind class name references used in a cartridge to the actual Java classes.

Adding External Java Source Files

Follow the steps given below to add external Java source files to the code generation settings of a cartridge:

1. In the 'External Sources' tab of the 'Java/EJB Code Generation Settings' dialog, click on the 'Add Files' button.

    The 'Add Source Files' dialog will be displayed.
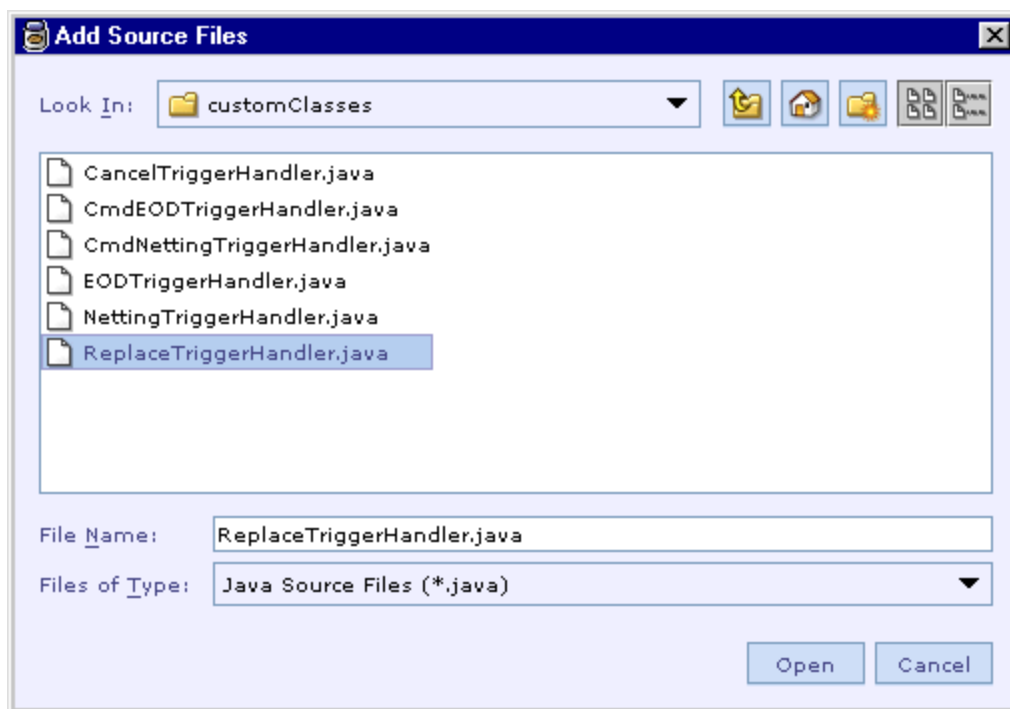
2. Navigate to the directory that contains the required Java source files and select them.

3. It is recommended that the Java source files are available under a child directory of the cartridge.

4. Click the 'Open' button.

5. The files will be included to the external source files list.
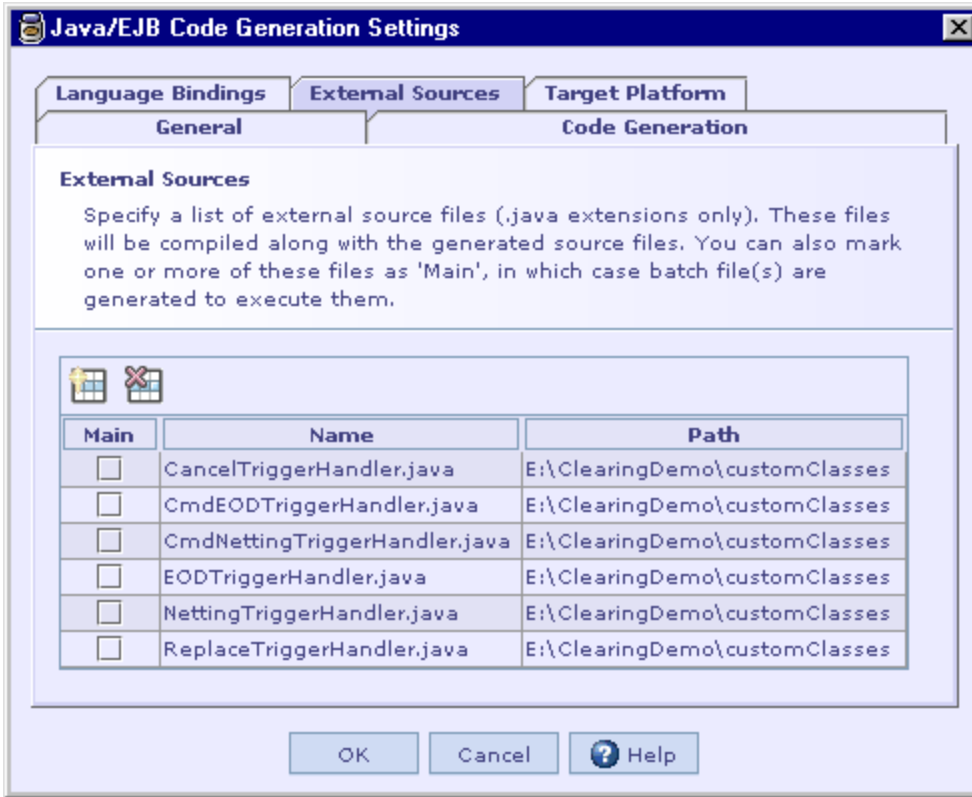
6. Select the 'Main' check box for one or more source files to mark them as application class files.

   For each file marked as an application class file, a batch file that invokes the corresponding application class will be generated.

Adding Directories

You can add a directory to the code generation settings of a cartridge so that the source/resource files under that directory are included in the code generation process. In case of source files, they are compiled and the class files are bundled with the generated components. In case of resource file, they are simply bundled with the generated components.

Follow the steps given below to add directories containing source/resource files to the code generation settings of a cartridge:

1. In the 'External Sources' tab of the 'Java/EJB Code Generation Settings' dialog, click on the 'Add Files' button.

2. The 'Add Source Files' dialog will be displayed.
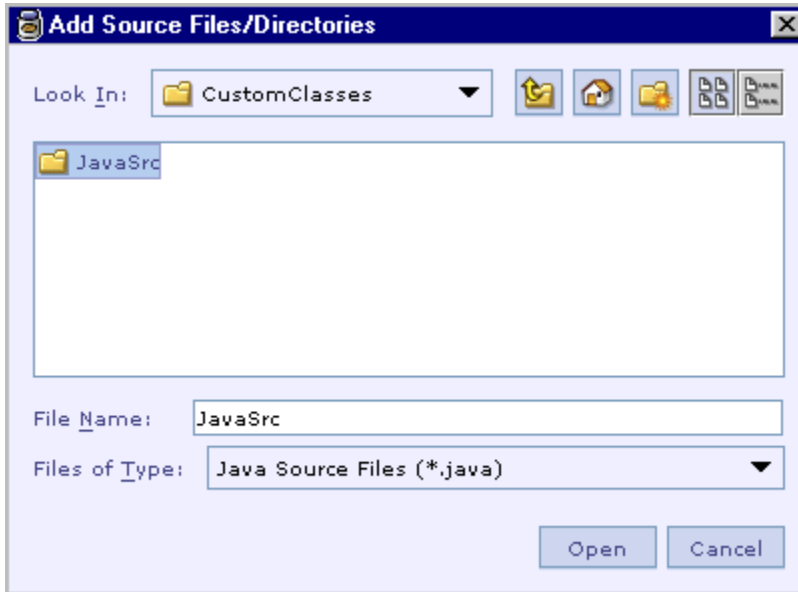
3. Navigate to the directory that contains the required source/resource files and select it.

4. It is recommended that the source/resource files are available under a child directory of the cartridge.

5. Click the 'Open' button.

   The directory will be included to the external sources list.
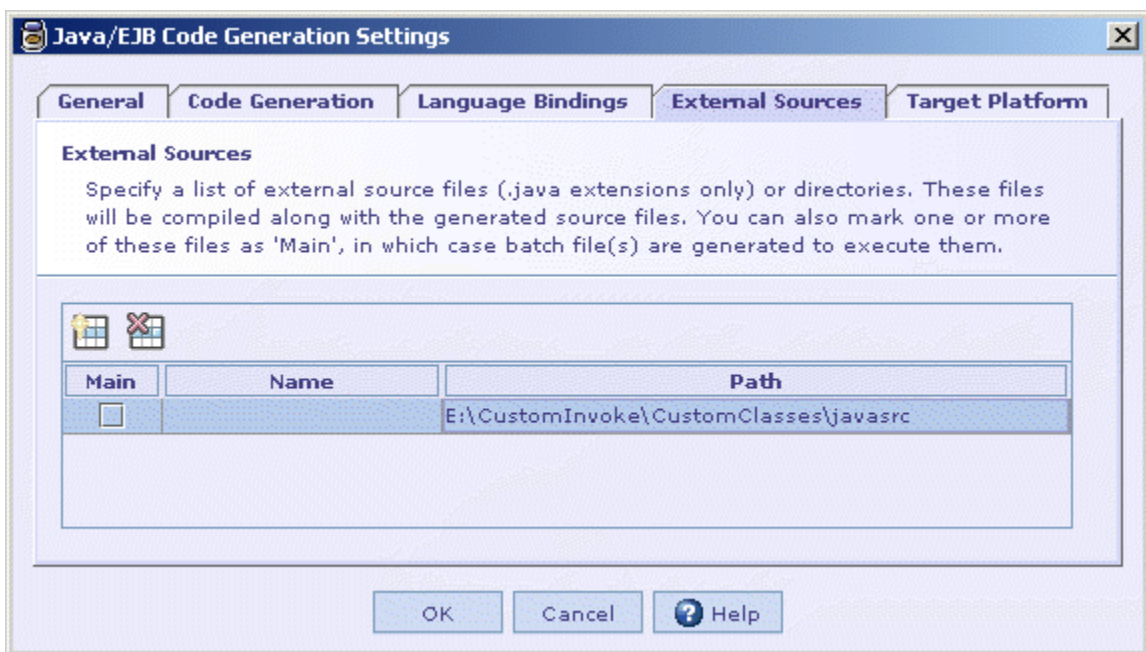
Note:

To remove files/directories from the external sources list, select them from the list and then click on the 'Remove Selected Files' button.

See Also:
General Tab
Code Generation Tab
Language Bindings Tab
Target Platform Tab
Code Generation Settings Dialog

Target Platform Tab (Java/EJB)

This tab allows you to specify the target server platforms for the generated components. You can also specify server specific deployment information for the generated EJB components. It also provides options for generating EJB application (EAR) and for adding additional modules to the generated EAR.
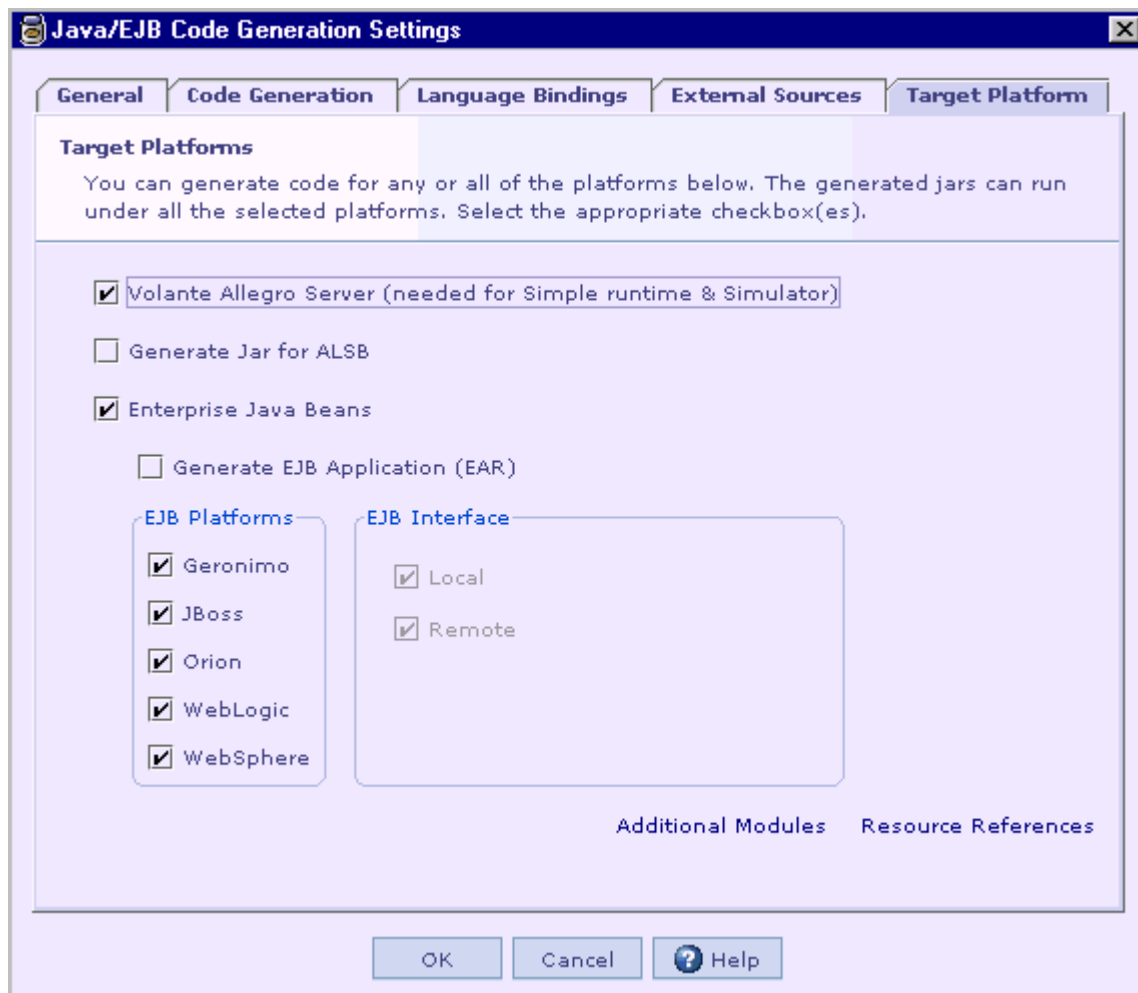
Generate Jar for ALSB

Select this check box, if you require a merged jar (that includes cartridge Jar(s), referenced cartridge Jar(s) and required standard Jars) for deployment under ALSB.

Enterprise Java Bean

Select this check box, if you intend to deploy the generated components into application servers such as WebLogic, WebSphere, JBoss and so on. Selecting this option will generate EJB components.

Generate EJB Application (EAR)

Select this check box, if you want to generate an EJB application by bundling the generated components and the modules specified in the [Additional Modules](#) dialog.

EJB Platforms

This allows you to generate server specific deployment descriptors for the generated EJB components so that you can straight away deploy them in the server. The supported EJB servers are JBoss, Geronimo, WebLogic, WebSphere and Orion. Select a check box corresponding to a server so that the generated EJB component is bundled with the deployment descriptor for that server platform.

Additional Modules

The 'Additional Modules' dialog displayed when clicking on the 'Additional Modules' link is used for packaging additional modules (in addition to the generated EJB components) into the generated EAR. You can add web, EJB, application client and library modules and the corresponding tags are generated in the application deployment descriptor (application.xml) of the generated EAR.

Follow the steps given below to add a module to the list of additional modules to be packaged into the generated EAR.

1. In the 'Target Platform' tab of the '[Java/EJB Code Generation Settings](#)' dialog, click on the 'Additional Modules' link.

2. The 'Additional Modules' dialog will be displayed.
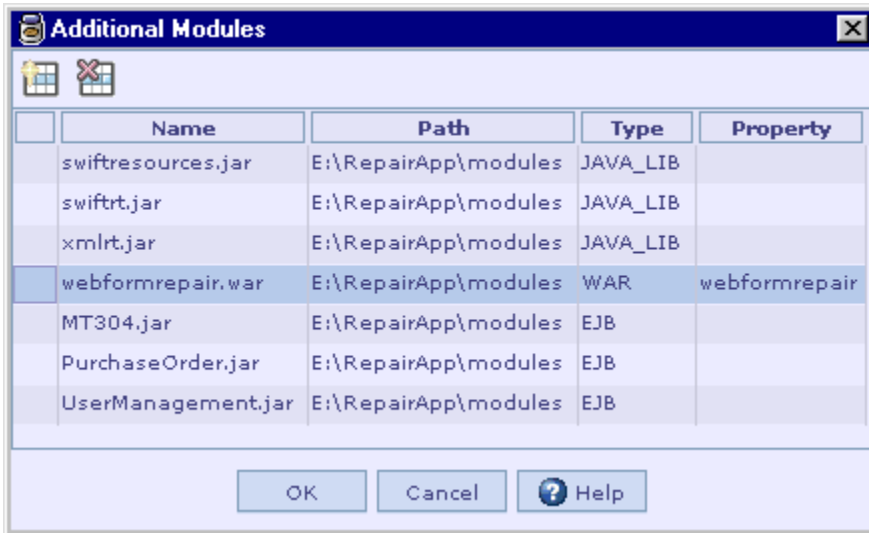
3. Click on the button 🔲.

   The File Open dialog will be displayed.

4. Navigate to the directory that contains the required module file and select the file.

5. Click the 'Open' button.

   The module file will be included to the additional modules list.

6. Select the type of the module from the drop down list displayed in the 'Type' column.

7. If you have added a web module, you can specify the context root of the web application in the 'Property' column.

Note:

To remove module files from the additional modules list, select them from the list and then click on the 🔲 button.

Resource References

The 'Resource References' dialog displayed when clicking on the 'Resource References' link is used to bind the resource reference names used in the cartridge to server specific resource names. See the Binding Server Resources section for more information.

Binding Server Resources

Entities defined in a cartridge refer to server resources. These references are specified in the ejb-jar.xml file during code generation. Before or during deployment, you need to map these resource references to server specific resource names. This procedure has the following limitations,

The mapping/binding tools are server specific. The way you map in WebSphere (using Web UI) is completely different from that of WebLogic (Builder tool). Worse still, the open source servers do not provide such a tool (or it is very bad). You have to manually edit the deployment descriptor XML (which is zipped in ejb jar, which in turn is zipped in an EAR).

Mapping is error prone, because servers do not properly report errors when you miss a mapping.

It is repetitive. Whenever you make a change to the cartridge and rebuild, the mappings are lost and you start all over again.

This feature enables providing this information in Cartridge itself. Though this information is specific to the server instance and should ideally be part of deployment process, it is very convenient to make it part of the build process for the following reasons:

You can specify resource mapping independently for each server supported.

The resource mapping information is saved along with cartridge and converted to server specific deployment descriptors every time you build.

You do not have to learn about the deployment descriptors of each server. The configuration screen is simple and same for all the servers.

To specify the bindings,

Select the ['Target Platform' tab](#) in the [code generation settings dialog](#).

Click on the 'Resource References' link. The dialog shown below is displayed. All the resources that are referenced from the cartridge are displayed. You need to fill the last column (Binding), for each of the servers you are interested in.
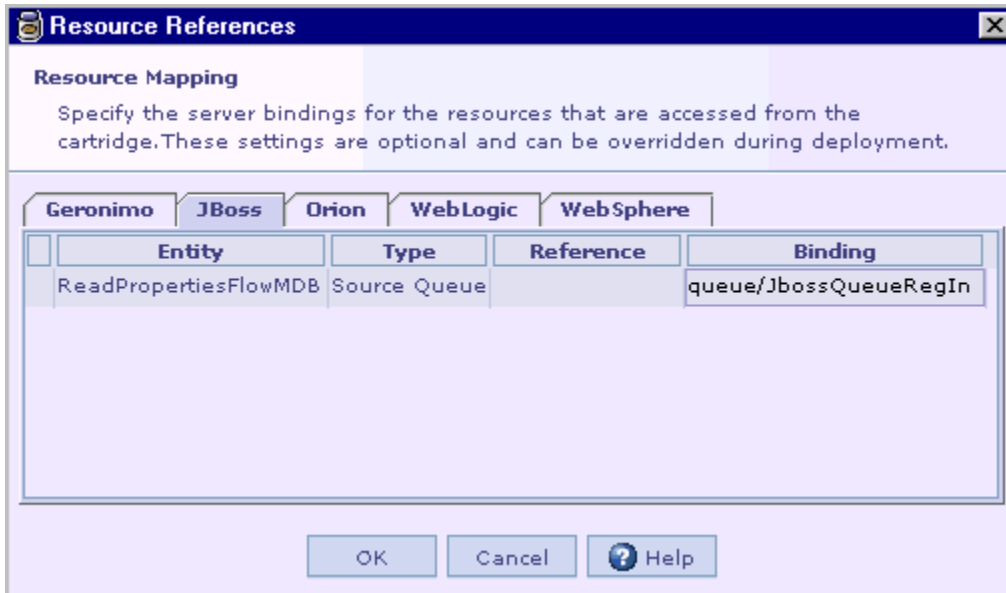
Note:

This feature helps in the process of binding Cartridge references to server resources. It does not help you in creating those resources (e.g DataSource) in the server. You have to create the resources as you normally do; using server specific tools.

You can always change the bindings you specified in the Cartridge later during deployment using server specific procedure. That is, this feature can simply be used to specify some defaults, which you override later.

See Also:
Code Generation Settings Dialog
Selecting the Default Platform
Generating a Cartridge
Deploying a Cartridge

Selecting the Default Platform

If you are going to work on a single platform/language, you can set the same as the default platform by selecting "Select Default Platform…" menu item from the Build menu. If you have set a Default Platform, then you can use the shortcut key **F7** for generating the cartridge and the shortcut key **F5** invoking Simulator.
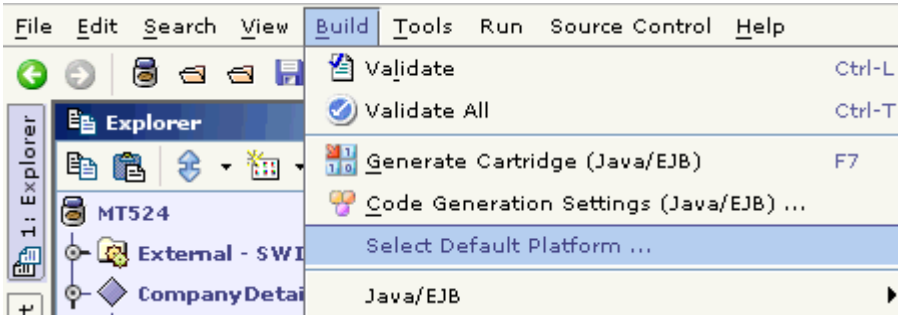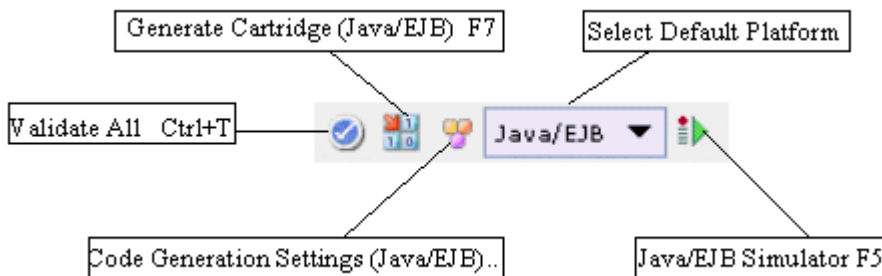
In the **Default Platform** dialog box that appears, you can check the radio-button.

Automatically, the selected platform is set as default in the Build menu as seen in the following picture.
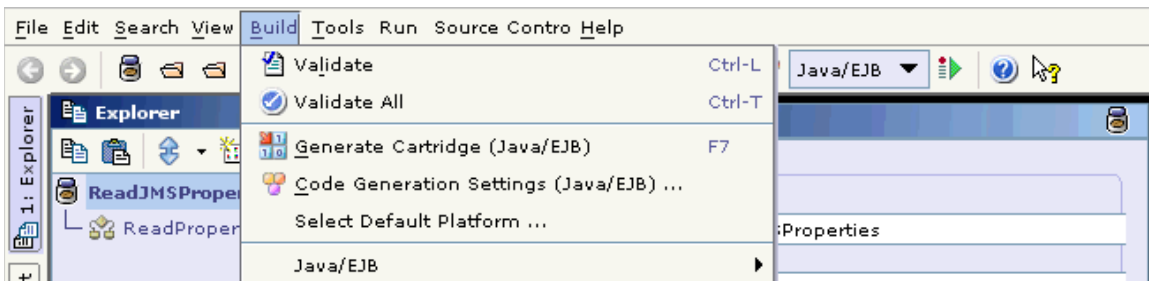
Now, you can use the short-cut key **F7** to generate cartridge in Java platform and the short-cut key **F5** to open Java Simulator.

You can also use the 'Select Default Platform' drop down list available in the main window toolbar to select the default platform. Other common functionalities that can be easily accessed from the main window toolbar include validating the cartridge, generating the cartridge, invoking the Code Generation Settings dialog and invoking the Simulator application.

See Also:

Code Generation Settings Dialog
Binding Server Resources

Generating a Cartridge

The **Build > Generate Cartridge** menu item or the Generate Cartridge button [icon] in the main window toolbar can be used for generating platform specific components representing the transformations modeled by a cartridge.

See Also:

[Code Generation Settings Dialog](#)
[Binding Server Resources](#)
[Selecting the Default Platform](#)
[Deploying a Cartridge](#)
[Code Generation](#)

Deploying a Cartridge

Once a cartridge is generated for a specific platform, the generated components can be deployed under the corresponding Runtime. Once deployed, the Runtime becomes capable of performing transformations defined by that cartridge.

Selecting the **Tools > Deployer** menu item brings up a submenu that displays platform specific Runtime. Selecting one of them brings up the platform specific **Deploy/Undeploy** dialog box that can be used to deploy the newly generated components and/or undeploy previously deployed components.

See Also:
[Code Generation Settings Dialog](#)
[Binding Server Resources](#)
[Selecting the Default Platform](#)
[Generating a Cartridge](#)
[Code Generation](#)

Simulator

The **Cartridge Simulator** application allows to test the internal/external message, message mapping and message flow components generated for a cartridge easily without the additional overheads of starting the application servers. It can also be used to test persistence and querying of internal messages using different database servers. See the section [Testing a Cartridge with Persistence Support](#) for more information.

Designer provides the Simulator application for the Java platform. The Simulator for the current platform can be invoked from the **Run** menu of Designer as shown in the following picture. The Designer main window toolbar also contains an icon for easy access of the Simulator application.
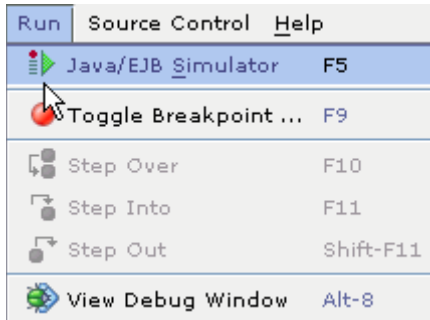
As Simulator is integrated with Designer, cartridges can be tested directly from Designer itself. The advantage is that, it is enough to launch Simulator only once, making it easier to test the cartridges then and there whenever a change is made.

Note

To launch Simulator in a platform supported by Designer, the following steps should be fulfilled:

Create a new cartridge or open an existing cartridge in Designer.

Generate code for the cartridge in that platform implementation.

Configuring Simulator. This is not needed unless you want to change the default settings of Simulator.

The features described in this document apply to Java Simulator. Where it is applicable to a particular platform, it is explicitly mentioned.

See Also:

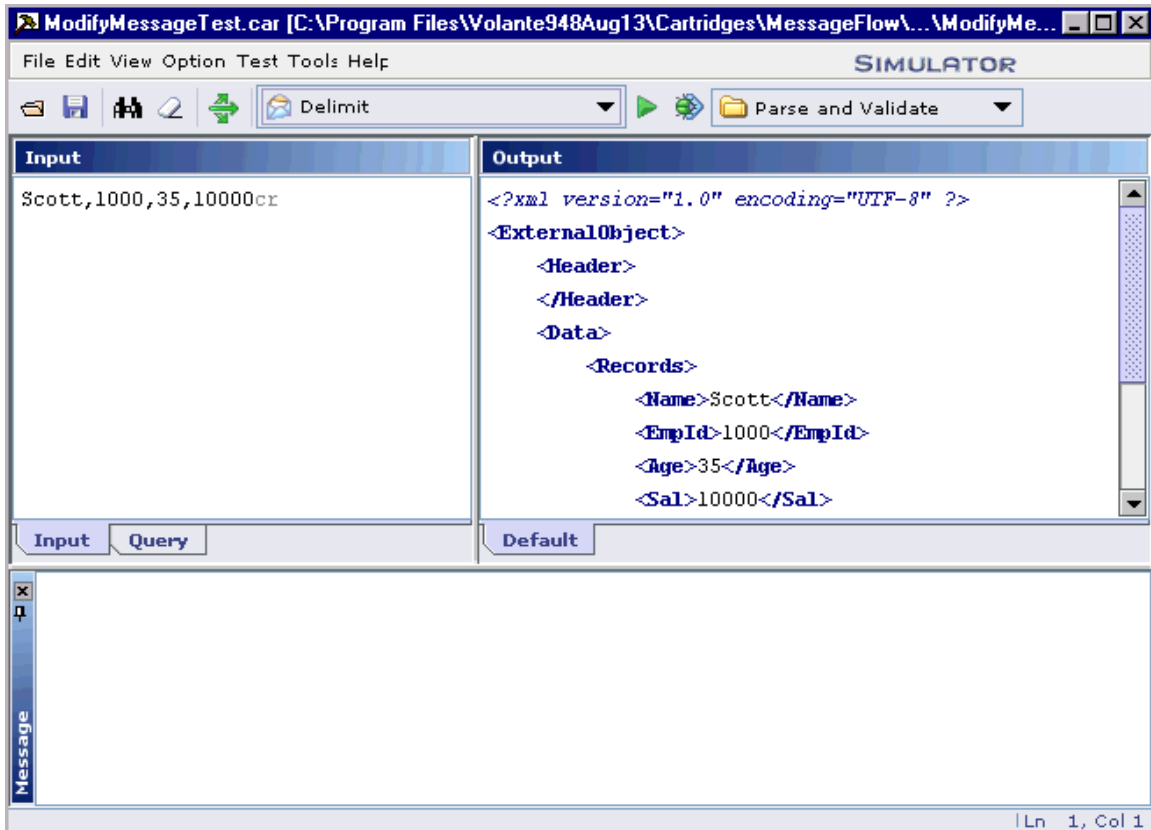Simulator User Interface

Testing with Simulator

Miscellaneous Features

Persistence in Simulator (Java/EJB)

Test Data Generation

Frequently Asked Questions

Simulator User Interface

The simulator UI by default looks like the one shown below.

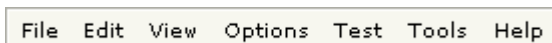The components of Simulator UI are listed below.
Menu Bar
Tool Bar
Input Pane
Output Pane
Message Window
Trace Window

Simulator Menu Bar

The Simulator menu bar is shown below and it provides all actions related to Simulator.

File Menu

Open Input From File …	Opens an input data file into the Input pane. In case of a message flow that accepts multiple input, selecting this menu item opens the data file into the currently selected text area corresponding to a 'Binary' type input variable.
Save Input …	Saves the content of the Input pane into a file.
Save Output …	Saves the content of the Output pane into a file.
Recent Input Files	Displays a list of the most recently used files.
Exit	Closes the current Simulator window.

Edit Menu

Find (Ctrl+F)	Displays the **Find** dialog, which is used to search for the specified text in the Input/Output pane.
Find Next (F3)	Resumes the search operation.
Clear Input	Clears the content of Input pane.
Clear Output	Clears the content of Output pane.

View Menu

Message (Alt+2)	Toggles display of the Message window.
Trace (Alt+3)	Toggles display of the Trace window.

Options Menu

Cascade Exceptions	Enables/disables cascading of exceptions when executing an external message.
Enable Tracing	Enables/disables generation of trace messages when executing a message flow.

Test Menu

[Generate Test Case](#)	Generates a single test case for the selected internal/external message in the [Input pane](#) of Simulator.
[Generate Test Data Set](#)	Generates a set of test data files for the selected internal/external message in the specified directory.
[Run Test Data Set](#)	Allows the user to execute test data files stored in the specified directory.
[Define DataGen Spec](#)	Allows the user to define data generation specification for the elements of the required data format. Test cases can be generated based on that specification.

Tools Menu

Submit (F5)	Starts processing of the input based on the currently selected message flow, message mapping or message.
Redeploy Cartridge	Redeploys the jars generated for the currently open cartridge. This has to be done whenever the cartridge is modified and code is generated. This also needs to be done whenever you change the [data source reference mapping](#).
[Diff Input/Output](#)	Allows the user to compare the contents in Input and Output panes.
[Resource Mapping](#)	Allows the user to map the data source reference name used in a cartridge to one of the [data sources configured in Simulator](#) and thereby allows the user to connect to different database servers.
[Create Schema](#)	Allows the user to create schema based on the database table design of a Persistence Designer.
[Data Entry](#)	Allows the user to generate an internal message test case just by entering data for the fields of that internal message without worrying about XML syntax.

Help Menu

Contents	This is a link to the Simulator help.
About Simulator …	Displays the version and copyright information about Simulator.

See Also:

[Simulator User Interface](#)

Simulator Tool Bar

The tool bar contains the following icons for accessing frequently used operations.

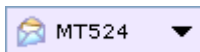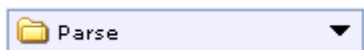|	
	Opens an input data file into the [Input pane](#).
	Saves the content of the [Input pane](#) into a file.
	Displays the **Find** dialog, which is used to [search for the specified text](#) in the Input/Output pane.
	Clears the content of both Input and Output panes.
	Redeploys the jars generated for the currently open cartridge. This has to be done whenever the cartridge is modified and code is generated. This also needs to be done whenever you change the [data source reference mapping](#).
MT524	The user can select the message, message mapping or message flow to be executed from the 'Select Message Flow/Message to execute' drop down list. It lists all the internal/external messages, message mappings and message flows available in the cartridge.
(F5)	The Submit button starts processing the input based on the currently selected message flow, message mapping or message.
	The Debug button starts debugging for the currently selected message flow, message mapping or message.
Parse	The drop down list with output options is displayed only when an internal/external message is selected for execution. The user can select one of the output

options: 'Parse', 'Parse and Validate' or 'Parse and Write'
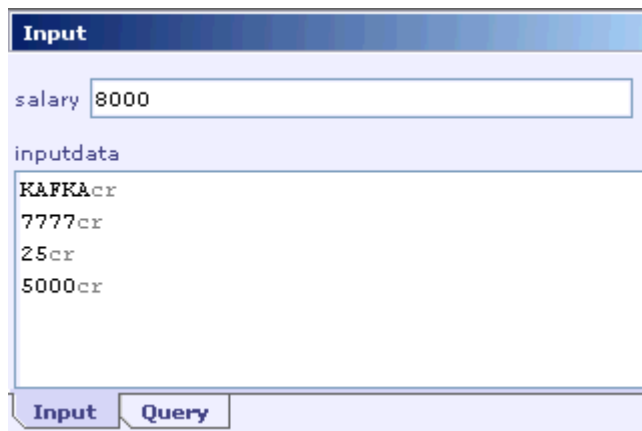
See Also:

[Simulator User Interface](#)
[Simulator Input-Output Options](#)

Input Pane

The **Input** pane has two tabs - Input and Query. The **Input** tab is where the data is fed either by typing directly or by opening a data file using the **File > Open Input from File...** menu item. The **Query** tab is used to execute queries that interact with the database directly. Refer [Invoking Queries](#) to know more about Query tab.

In case of message flows that accept more than one input arguments, the Input pane is laid out as a form with text field and/or text area input elements based on the type of the corresponding input variable. For a simple type input variable, a text box is shown to accept the input and for a 'Binary' type input variable, a text area is shown to accept the input.

The Input pane shown in the above picture corresponds to a message flow that accepts two input arguments: 'salary' and 'inputdata'. While the 'salary' input variable is of 'Integer' type, the 'inputdata' variable is of 'Binary' type. To enter input into a text area (corresponding to a 'Binary' type input variable) by opening a data file, first select that input area and then open the input data file by using the **File > Open Input from File...** menu item.

The shortcut menu of the Input pane allows the user to cut (Ctrl+X), copy (Ctrl+C) and paste (Ctrl+V) its content. The **Copy as HTML** menu item creates a copy of the content currently selected in the Input pane using HTML syntax and it can be saved into an HTML file. The **Find** and **Find Next** menu items allow to [search for text](#) within the Input pane.
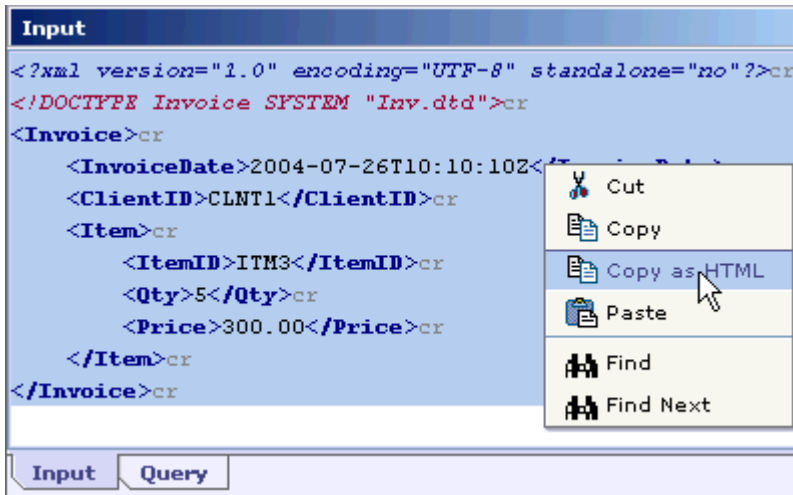
See Also:

Simulator User Interface
Invoking Queries

Output Pane

The **Output** pane has a **Default** tab to which the output generated by executing a message or message mapping is written by default. When a message flow with multiple output variables is executed, a separate tab is created in the **Output** pane to write the value of each output variable as shown in the following picture.
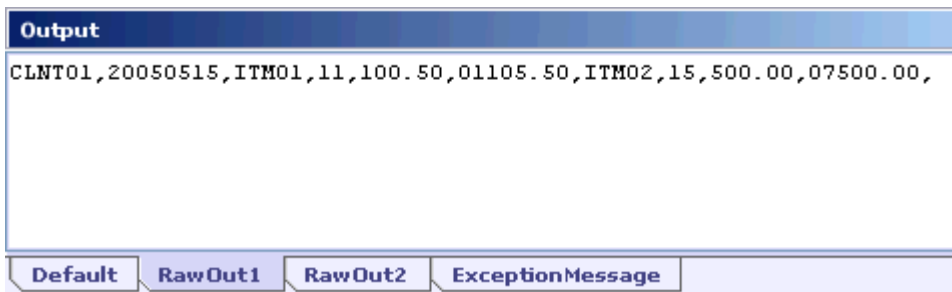
The user can close a particular output tab or all the output tabs except the **Default** tab by selecting the corresponding menu item from the shortcut menu as shown in the following picture.
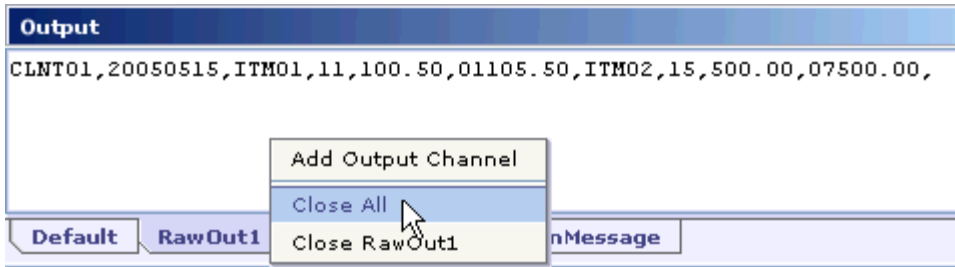
The user can add new output channels apart from the default one of the Output pane.

The Output pane also allows the user to cut (Ctrl+X), copy (Ctrl+C) and paste (Ctrl+V) its content. The **Copy as HTML** menu item creates a copy of the content currently selected in the Output pane using HTML syntax and it can be saved into an HTML file. The **Find** and **Find Next** menu items allow you to search for text within the Output pane.

See Also:

Simulator User Interface

Simulator Message Window

The Message window displays any error generated on executing the input data. The user can toggle the display of the Message window either by selecting the **View** > **Message** menu item or by pressing the shortcut key Alt-2.
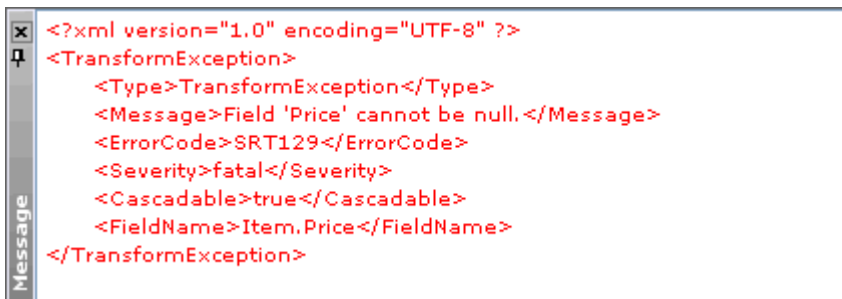
The user can copy the content of the Message window by first selecting the content and then selecting the 'Copy' menu item from the shortcut menu.

The 'Clear' menu item of the shortcut menu clears the current content of the Message window.

See Also:

Simulator User Interface

Trace Window

The Trace window allows the user to view the trace messages generated during the execution of a message flow. The user can toggle the display of the Trace window either by selecting the **View** > **Trace** menu item or by pressing the shortcut key **Alt-3**.

The shortcut menu of the Trace window provides for copying and clearing of its content.

See Also:

Simulator User Interface

Log Of SQL Statements

SQL statements that are executed at runtime are logged in the "runtime.log" present in the log directory of Designer home. For e.g. while executing a query, addition or removal of records from/into a database, are logged in the 'runtime.log'. This can also be viewed in the simulator 'Trace' window as follows.

```
[volante.runtime.persistence.InvoicePM] - [datasource=transformdb]Executing insert into Item (Invoice_FK, ItemID, Qty, Price)
[volante.runtime.messageflow.PersistInColumnsFlow] - Executing activity PersistInColumnsFlow.Serialize1(Serialize Invoice)
[volante.runtime.message.Invoice] - serialize Invoice
[volante.runtime.messageflow.PersistInColumnsFlow] - Executing activity PersistInColumnsFlow.Stop1(Stop)
[volante.runtime.messageflow.PersistInColumnsFlow] - Message Flow PersistInColumnsFlow completed
[volante.runtime.tm] - Commiting transaction Transaction [GlobalId=Revathi//0]
[volante.runtime.persistence.InvoicePM] - [datasource=transformdb]Executing SELECT Invoice.Invoice_PK,Invoice.InvoiceDate,I
[volante.runtime.persistence.InvoicePM] - [datasource=transformdb]Executing SELECT Invoice.Invoice_PK,Invoice.InvoiceDate,I
```

See Also:

[Testing with Simulator](#)

Testing with Simulator

Follow the steps below to test an entity defined in a cartridge using the Simulator. These steps apply for testing messages, mappings and message flows.

1. Create or open the cartridge and generate code for it.

2. Launch Simulator from the **Build** menu.

   The Simulator in turn starts the Volante Server and the database server. (These servers are launched when Simulator is started for the first time after Designer is started. These servers are shutdown only when Designer is stopped.) The settings of these servers can be configured using the Simulator configuration file. See the section [Configuring Simulator](#) for further information. On successful launching of these servers, Simulator displays all message flows, message mappings and messages in the cartridge.
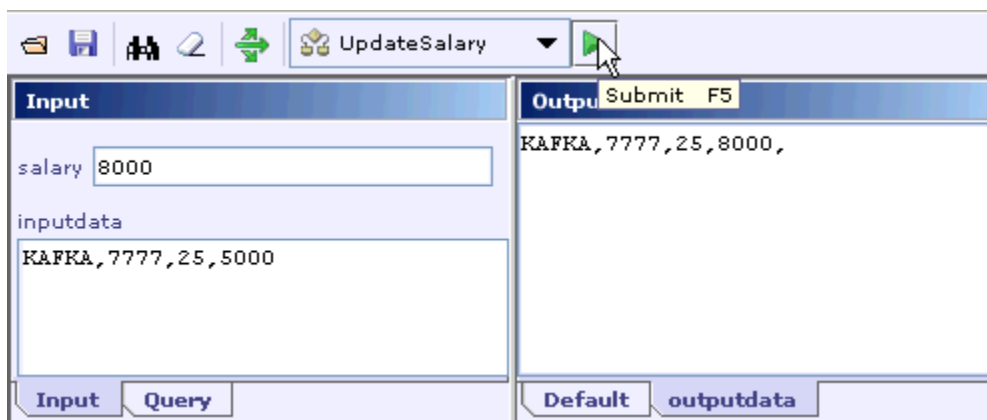
3. Select the message flow, message or message mapping to be executed.

4. In case of a message, select the processing that needs to be done. This can be one of 'Parse', 'Parse and Validate' and 'Parse and Write'.

5. Enter the data into the [Input pane](#) or open an input test file using the **File > Open Input from File** menu item.

   In case you have selected a message flow with multiple inputs, the Input pane is laid out as a form with text field and/or text area input elements based on the type of the corresponding input variable. For a simple type input variable, a text box is used to accept the input and for a 'Binary' type input variable, a text area is used to accept the input. To enter input into a text area by opening a data file, first select that input area and then open the input data file by using the **File > Open Input from File...** menu item.

6. Click the **Submit** button to start execution.

7. The output of the transformation and/or errors generated is displayed in the [Output pane](#) and the [Message window](#) respectively.

Note

Once Simulator is launched and if any change is made in the opened cartridge or another cartridge is opened, generate code and redeploy the cartridge to update Simulator.
For each cartridge that you opened, you can open a Simulator window to test that cartridge independently. A Simulator instance is attached to corresponding Designer window. Closing the Designer window also closes the associated Simulator Window.

See Also:

[Simulator Input-Output Options](#)
[Simulator User Interface](#)
[Cartridge Simulator](#)

Simulator Input-Output Options

The following table shows the possible transformations that can be done using Simulator and related notes in each case.

Input	Output	Transformation	Comments
External Message	Parse	-	Input is format specific. Output is an XML representation of the Input. No validation takes place.
External	Parse and	-	Input is format specific.

Message	Validate		Output is an XML representation of the Input. The validations specified in the Validation Rules node of the external message are carried out.
External Message	Parse and Write	-	Input is format specific. Output is format specific. The validations specified in the Validation Rules node of the external message are carried out.
Message Flow	-	As defined in the message flow.	The input and output are specific to the message flow. There can be zero or many input/output for a message flow.
Internal Message	Parse	-	Input should be an XML representation of NO. Output is an XML representation of the NO.
Internal Message	NOXMLOut	-	Requires some trigger (router) at Internal Message. Input should be an XML representation of NO. Output is an XML representation of NO. Processing (including validations) specified in processing rules are carried out.
Internal Message	Output Message	NO -> Output	Requires some trigger (router) at Internal Message. Input should be an XML

representation of NO.

Output is format specific.

Processing (including validations) specified in Processing Rules node and output validations are carried out.

Input/Output for 2.x features (deprecated)

Input Message	Parse	-	Input is format specific. Output is an XML representation of the Input. No validation/processing takes place.
Input Message	NOXMLOut	Input->NO	Requires some trigger (router) at Internal Message. Input is format specific. Output is an XML representation of NO. Input validations and processing (including validations) specified in Processing Rules node are carried out.
Input Message	Output Message	Input -> NO NO -> Output	Requires some trigger (router) at Internal Message. Input is format specific. Output is format specific. Input message validations, processing of internal message, internal message validations and output message validations are carried out.

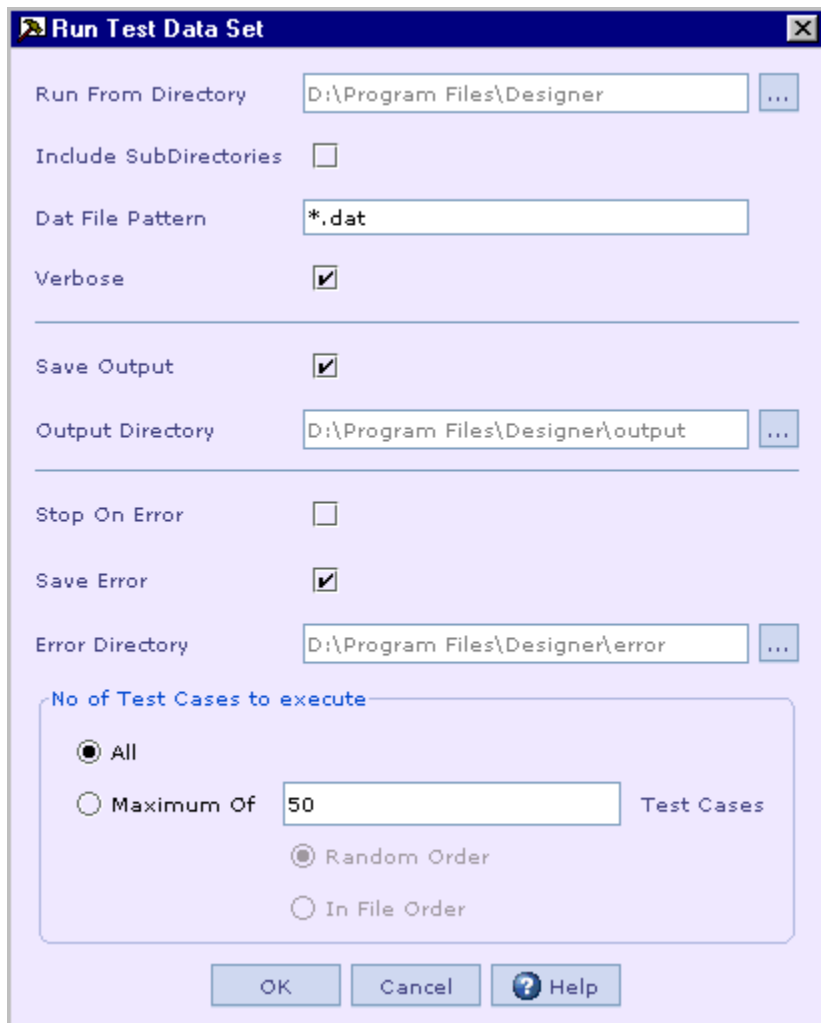**See Also:**

[Testing with Simulator](#)

Running Test Data Set

The **Run Test Data Set** dialog allows the user to execute test data files stored in the specified directory.

Follow the steps given below to execute a set of test data files.

1. Select the Test > Run Test Data Set menu item.

   The **Run Test Data Set** dialog will be shown.

2. In the **Run From Directory** field, specify the root directory that contains the test data files to be executed.

   You can click on the ellipsis button to bring up the 'File Open' dialog box and select the required directory.

3. Select the **Include SubDirectories** check box, if you want the subdirectories under the test root directory to be included in the test run.

4. In the **Dat File Pattern** field, specify the pattern of the file names to be included in the test run.

5. Select the **Verbose** check box, if you want the success or failure message needs to be displayed in the Message window as soon as a file is run.

6. Select the **Save Output** check box, if you want to save the output of the files that have passed.

   Select the directory into which the output files should be saved.

7. If the **Save Output** check box is selected, the ellipsis button besides the **Output Directory** field is enabled so that you can select the output directory.

8. It should be noted that if the test data set contains subdirectories, the same subdirectory structure will be created under the specified output diectory and the output of the valid files under the subdirectories will be saved under the corresponding subdirectories.

9. Select the **Stop On Error** check box, if you want to stop the test run as soon as the failure of a test file.

10. Select the **Save Error** check box, if you want to save the error message of the files that have failed.

11. Select the directory into which the error files should be saved.

    If the **Save Error** check box is selected, the ellipsis button besisdes the **Error Directory** field is enabled so that you can select the error directory.

    It should be noted that if the test data set contains subdirectories, the same subdirectory structure will be created under the specified error diectory and the error message of the invalid files under the subdirectories will be saved under the corresponding subdirectories.

12. In the **No of Test Cases to execute** section, select the **All** option to run all test files. On the other hand, you can specify the maximum number of test cases to be executed in the corresponding text field.

13. If you have opted to run the maximum number of test cases, you can select between whether to select test files in random order or to select test files in the normal file order by selecting the appropriate option button.

14. Click on the **OK** button to start running the test data files. The following progress message box will be displayed.
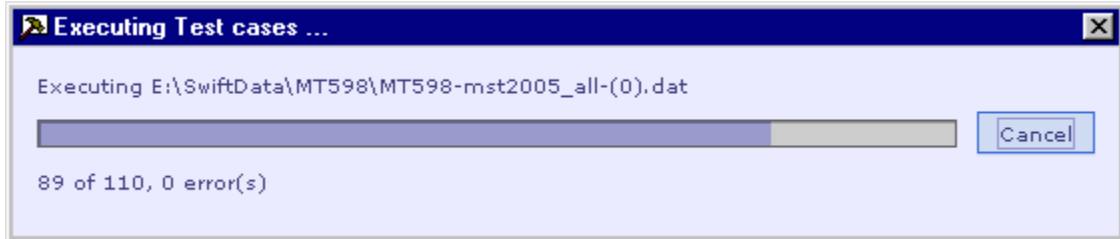
15. You can click on the **Cancel** button on this message box to abruptly stop running the test data set.

See Also:

Defining Data Generation Specification
Generating a Test Case
Generating Test Data Set
Entering Test Data for Internal Messages
Testing with Simulator

Configuring Simulator (Java/EJB)

The simulator.xml file under the **<installation dir>\config\simulator** directory is used to configure the Java/EJB Simulator.

Given below is the sample content of simulator.xml file:

```xml
<?xml version="1.0"?>
<simulator>
   <server port="1606"
    datasources="${designer.home}/config/simulator/data-sources.xml" />
   <launch name="hsqldb" application="java" wait="false" shutdown="true">
   <arg value="-classpath" />
   <arg value="${designer.home}/lib/ext/hsqldb.jar" />
   <arg value="org.hsqldb.Server" />
   <arg value="-port" />
   <arg value="1608" />
   <arg value="-database" />
   <arg value="${designer.home}/config/simulator/database/volantedb" />
   </launch>
</simulator>
```

Here, the 'server' tag is used to define the settings of Volante Server that Simulator launches.

The 'port' attribute of the 'server' tag specifies the port to be used by Volante Server.

The 'datasources' attribute of the 'server' tag specifies the file that contains the definitions of the data sources to be used by Volante Server.

The 'launch' tag is used to specify the applications to be launched for Simulator. These applications are launched when Simulator is started for the first time. Here, it is used to specify the HSQL database server to be launched when Simulator is started.

The 'application' attribute of the 'launch' tag specifies the command to be executed.

The 'arg' tag represents the argument/option to be passed to that command. Its 'value' attribute specifies the actual value. Each option should be specified in a separate 'arg' tag and should be in the correct order.

See Also:

Testing with Simulator

Simulator Options

The Cascade Exceptions and Enable Tracing menu items available under the Options menu of Simulator provide features that help in testing and debugging of cartridges.

See Also:

Testing with Simulator

Generate Trace Messages

The user can enable generation of trace messages during execution of a message flow by checking the **Options** > **Enable Tracing** menu item. You can then view the

trace messages in the Trace window by selecting the **View** -> **Trace** menu item. The Trace window displays the execution log or trace.

```
[volante.runtime.tm] - Starting  new transaction Transaction [GlobalId=Revathi//28]
[volante.runtime.messageflow.serialize] - Message Flow serialize started
[volante.runtime.messageflow.serialize] - Executing activity serialize.Start1(Start)
[volante.runtime.messageflow.serialize] - Executing activity serialize.Parse1(Parse1)
[volante.runtime.message.delimit] - parse delimit
[volante.runtime.messageflow.serialize] - Executing activity serialize.Serialize1(Serialize)
[volante.runtime.message.delimit] - serialize delimit
[volante.runtime.messageflow.serialize] - Executing activity serialize.Parse2(Parse2)
```

Right clicking inside the 'Trace' window displays a pop up with menu items **Clear**, **Copy** and **Go to Definition**. The 'Clear' menu item when selected, clears the content in the 'Trace' window and the 'Copy' menu item when selected copies the selected text.

Selecting 'Go To Definition' menu from any line in 'Trace' window, navigates to the location (an activity in a message flow, an Internal/External message or a mapping) specified in the line. For example, consider selecting 'Go To Definition' menu from the following line.

```
[volante.runtime.messageflow.serialize] - Executing activity serialize.Start1(Start)
```

The 'Start' activity in the message flow UI would be selected.

See Also:

Testing with Simulator

Cascade Exceptions

The user can enable cascading of exceptions during execution of an external message by checking the **Options** > **Cascade Exceptions** menu item. When this option is enabled, Simulator will proceed with processing the external message even when there are exceptions. These exceptions are listed in the Message window. When this option is disabled by unchecking the **Options** > **Cascade Exceptions** menu item, the Simulator will stop processing the external message as soon as it comes across an exception.

See Also:

Testing with Simulator

Miscellaneous Features

These features make working with Simulator more easier/interactive.

Feature	Description
Syntax Highlighting	The test data files are syntax highlighted in Simulator.
Viewing of Non-printable Characters	Non-printable (binary) characters are shown as # followed by its code value.
Search Text	This feature allows you to search for a specific text in the Input/Output pane.
Compare Input and Output	This feature allows you to compare the contents in Input and Output pane.
Adding Output Channels	This feature allows you to add a new output channel to the Output pane.

Syntax Highlighting (Data)

In Simulator, the test data files are syntax highlighted. This works for SWIFT, XML and FIX. For example, in case of SWIFT, Tags and Qualifiers are displayed in different colors as could be seen from the following picture.

The 'cr' at the end of the line is visible. The 'lf' is not shown because the line feed results in the next character moving to the next line (hence need not be shown explicitly, to avoid noise).

See Also:

Testing with Simulator

Viewing of Non-printable Characters

Non-printable (binary) characters are shown as # followed by its code value.

To type a non-printable character, type the ASCII number (e.g 25) with CTRL pressed. CTRL key should be released only after you complete the number. You can type a character by giving its code value with CTRL pressed. CTRL key should be released only after you complete the number. For eg., to type 'A' you have to press CTRL and type 65.
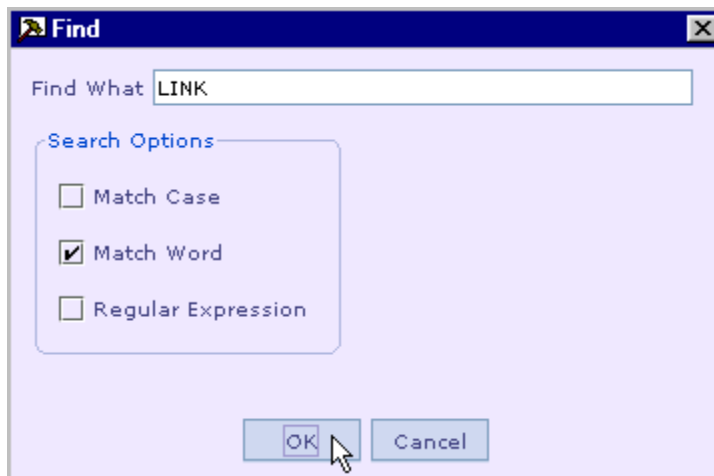
See Also:

Testing with Simulator

Search Text

To search for a specific text in the Input/Output pane, refer the following steps:

1. Keep the cursor in the required pane (Input/Output).

2. Select **Edit** > **Find** menu item.

   The **Find** dialog is shown.

3. Type the required text in the 'Find What' text box.

4. Select the required search options.

   Match Case

   You can select this option if you want to do case-sensitive search.
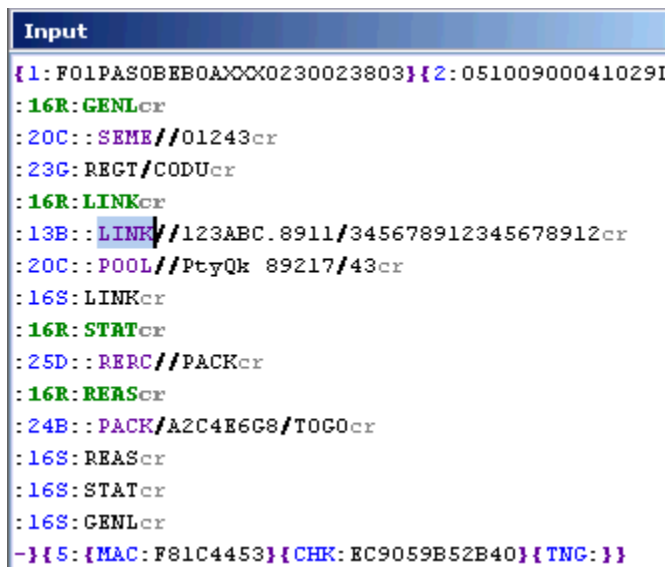
   Match Word

   You can select this option if you want to search for an exact word. For eg., if you want to search for a word, say, "SEC". If you have not selected this option, then the word "SECTION" if found will also be highlighted as it has "SEC" in it.

   Regular Expression

   You can also give Regular Expression in the "Find What" text box.

5. Click the OK button to start searching for the specified text.

6. If found, the specified text will be highlighted as shown in the following picture.

7. Select **Edit** > **Find Next** menu item to search for the next occurrence of the specified text.

   For this, you can alternately use the shortcut key **F3** or the **Find Next** menu item from the shortcut menu of the Input/Output pane.

See Also:

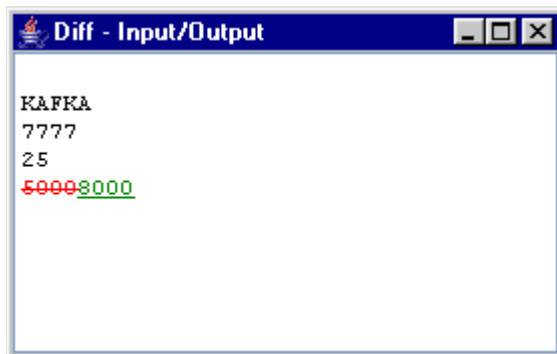Testing with Simulator

Compare Input and Output

You can compare the contents in Input and Output pane using the **Tools** > **Diff Input/Output** menu item.
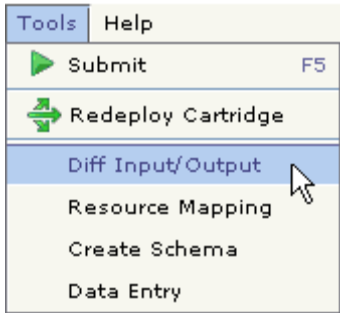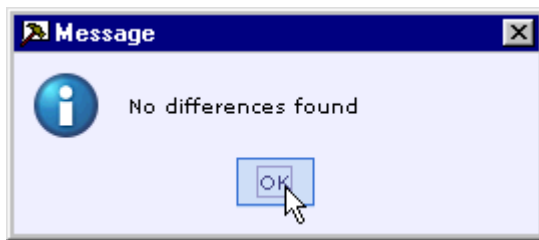
In the following picture there is a difference between the contents in the 'inputdata' text area of the **Input** pane and **Output** pane in the 4th line.
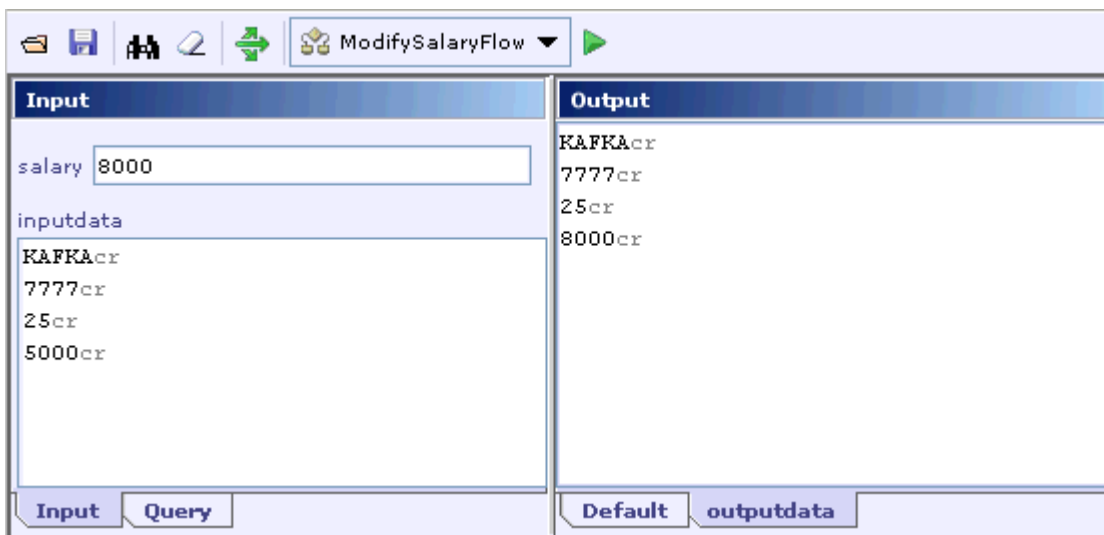
When you select **Tools** > **Diff Input/Output** menu item, you will get a window where the difference is highlighted as shown below.

If the contents in the **Input** and **Output** panes are identical, you will get a message dialog as under:

See Also:

Adding Output Channels

In some cases, an application might have multiple channels of output, or the user might have to send output through a particular channel (not the default one). In such cases, an error message similar to the one given below will be shown during execution.

```
<TransformException>
    <Type>TransformException</Type>
    <Message>Unexpected error</Message>
    <Severity>1</Severity>
    <Cascadable>false</Cascadable>
    <Error-Phase>Internal Message</Error-Phase>
    <StackTrace>
    null
Type=TransformException, Message=Unable to lookup RMI output device
'SecondOut', ErrorCode=null, Severity=1, Cascadable=false
```

In this case, the output is sent through the output channel (output.device) 'SecondOut', but Simulator is not configured to receive it. Hence the error.

The user can add new output channels apart from the default one.

Follow the steps given below to add an Output Channel. The terms output channel and output device are used interchangeably. They refer to the same entity.

1. Right click besides the **Default** tab in the Output pane to popup the shortcut menu with the **Add Output Channel** menu item.
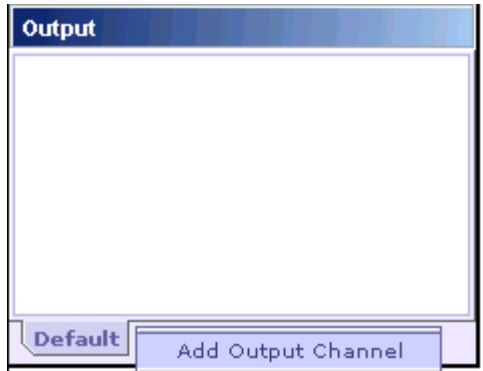
2. Select that menu item to bring the **New Channel** dialog.

3. Enter the output device name in the **Channel Name** text box and click OK.
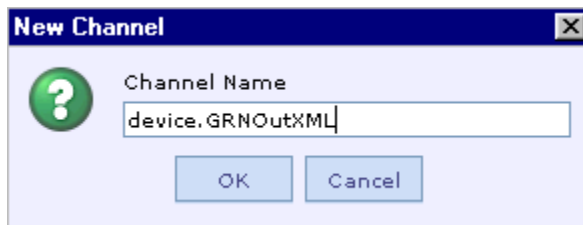
A new tab with the given channel name appears in the Output pane of Simulator as shown below. Now if the input is given and the **Submit** button is clicked, the above error disappears and the corresponding output appears in the newly added output channel.

See Also:

Testing with Simulator

Persistence in Simulator (Java/EJB)

Simulator allows to test persistence and querying of internal messages using different database servers by providing for mapping the data source reference name used in a cartridge to one of the data sources configured in Simulator.

For testing persistence support in Simulator, the user needs to

1. Define the data sources to be available in Simulator.

2. Map the data source reference name used in a cartridge to one of the data sources available in Simulator.

3. Create the data base schema.

Defining Data Sources

To connect to a database server from Simulator, configure it in the **data-sources.xml** file under the **<installation dir>\config\simulator** directory as shown in the following example.

```xml
<?xml version="1.0"?>
<data-sources default-data-source="simulatordb" >
    <data-source
        connection-driver="oracle.jdbc.driver.OracleDriver"
        url="jdbc:oracle:thin:@200.200.200.10:1521:orcl"
        username="krishnan"
        password="krishnan"

class="com.volante.component.server.jdbc.ManagedJDBCDriverConnectionPool"
        dialect="oracle"
        location="oracledb"
        name="oracledb"
    />
    <data-source

class="com.volante.component.server.jdbc.ManagedJDBCDriverConnectionPool"
        connection-driver="com.mysql.jdbc.Driver"
        dialect="mysql"
        location="mysqldb"
        name="mysqldb"
        password=""
        url="jdbc:mysql://localhost/test"
        username=""
    />
    <data-source

class="com.volante.component.server.jdbc.ManagedJDBCDriverConnectionPool"
        connection-driver="org.hsqldb.jdbcDriver"
        dialect="hsql"
        location="hsqldb"
        name="hsqldb"
        password=""
        url="jdbc:hsqldb:hsql://localhost"
        username="sa"
    />
    <data-source

class="com.volante.component.server.jdbc.ManagedJDBCDriverConnectionPool"
        connection-driver="COM.ibm.db2.jdbc.net.DB2Driver"
        dialect="db2"
        location="db2db"
        name="db2db"
        password="db2admin"
        url="jdbc:db2://200.200.200.5/toolsdb"
        username="db2admin"
```

```
        />
        <data-source
            connection-driver="org.hsqldb.jdbcDriver"
            url="jdbc:hsqldb:hsql://localhost:1608"
            username="sa"
            password=""

    class="com.volante.component.server.jdbc.ManagedJDBCDriverConnectionPool"
            dialect="hsql"
            location="simulatordb"
            name="simulatordb"
        />
        <data-source

    class="com.volante.component.server.jdbc.ManagedJDBCDriverConnectionPool"
            connection-driver="sun.jdbc.odbc.JdbcOdbcDriver"
            dialect="mssql"
            location="mssqldb"
            name="mssqldb"
            password="krishnan"
            url="jdbc:odbc:testnorth"
            username="Krishnan"
        />
    </data-sources>
```

In the above configuration, each <data-source> tag defines a connection to a separate database server.

The following table describes the connection attributes used in the <data-source> tag.

| Class	Name of the connection pool class. Always use **com.volante.component.server.jdbc.ManagedJDBCDriverConnectionPool**
name	Name of the data source.
location	Name of the server resource.
connection-driver	JDBC driver class.
url	URL to be passed to the driver.
username	User name to be used to get a connection to the database.
password	Password to connect to the database (for the specified username).

Resource Reference Mapping

By mapping the data source reference name used in a cartridge to one of the [data sources configured in Simulator](#), the user can connect to the database server identified by that data source configuration.

This section assumes that **invoicedb** is defined as the data source reference name using the 'Java/EJB Code Generation Settings' dialog as shown in the following picture.
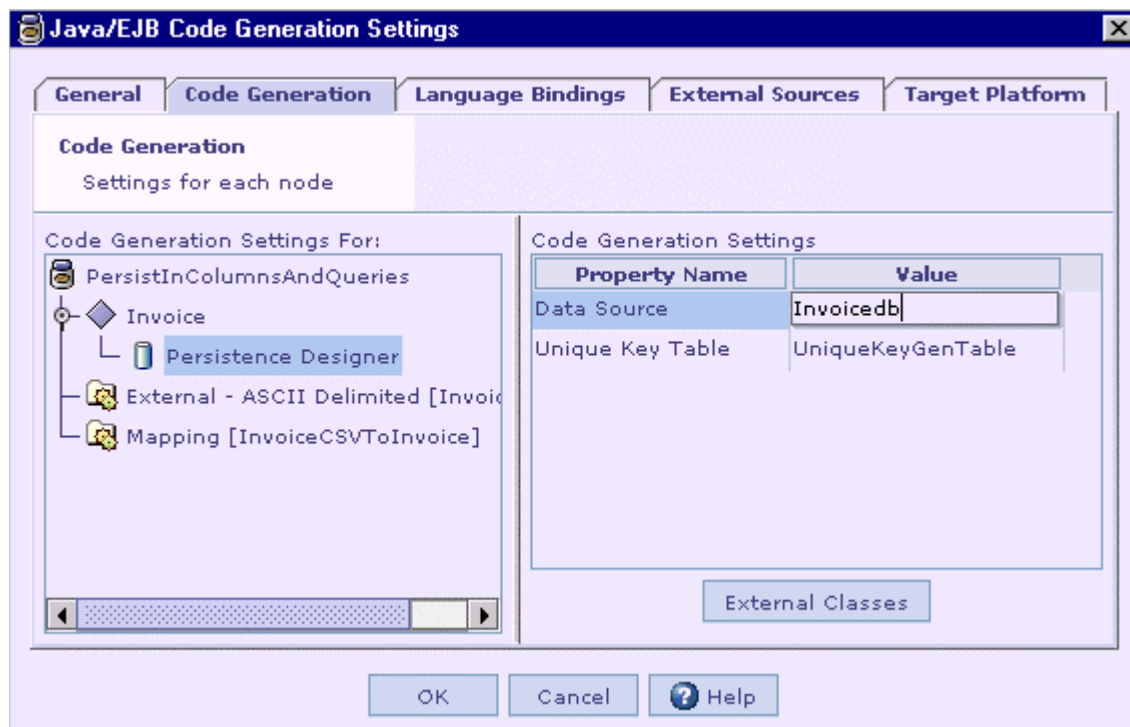

Follow the steps given below to map the data source reference to one of the Simulator data sources.

1. Select the **Tools** > **Resource Mapping** menu item from the Simulator menu bar. The 'Resource Reference Mapping' dialog is shown.

2. Click on the ⊞ icon to add a new row, if required.

3. Enter/edit the data source reference name used in the cartridge in the 'Data source' column.
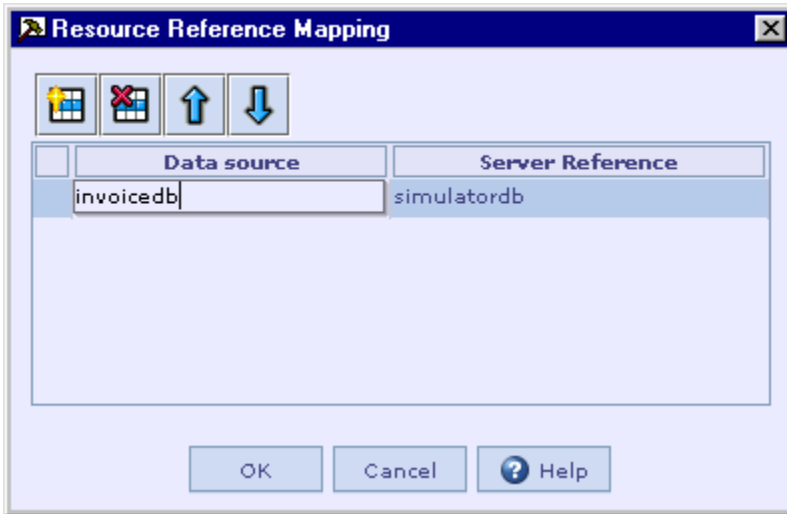
4. Now select the data source to be connected (for persistence and querying of internal messages) from the 'Server Reference' drop down list.
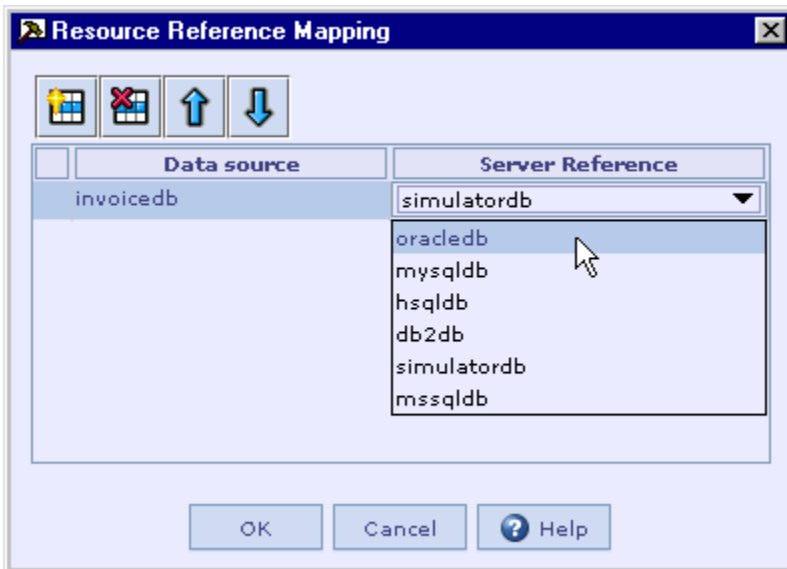
5. Click on the OK button to complete data source reference mapping.

6. Click OK to close the message box that prompts you to redeploy the cartridge.
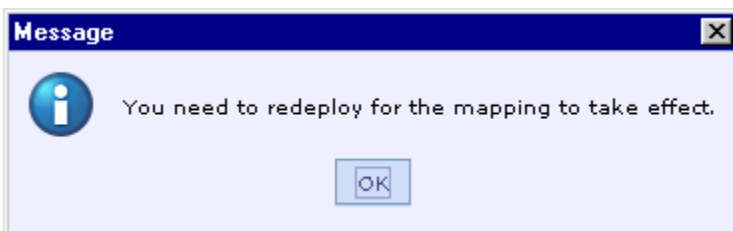
7. Click on the 'Redeploy Cartridge' icon ⬆ in the Simulator toolbar to start redeploying the cartridge.
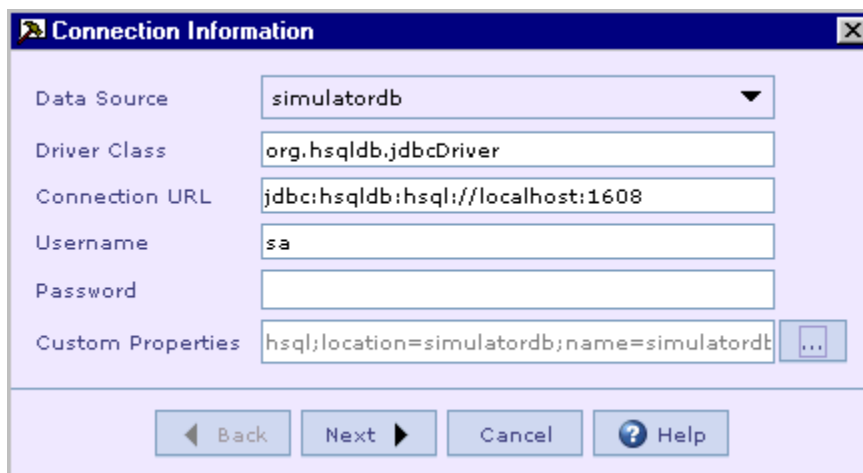
See Also:

Defining Data Sources
Testing a Cartridge with Persistence Support

Creating Schema

To persist internal messages, the schema should have been created. The XML and the SQL schema files are automatically generated during code generation under **<Cartridge directory>\java** directory, if the cartridge has a **Persistence Designer** node added. The schema should be created afresh whenever any change is made to the database table design.

Follow the steps given below to create schema based on the database table design of a Persistence Designer.

1. Select **Tools** > **Create Schema** menu item in Simulator.

2. In the 'Connection Information' dialog that appears, select the required data source.

3. Other fields required for making the database connection are populated from the Simulator data source configuration. Modify them as required and click the **Next** button.

4. In the 'Select SQL Schema' dialog that appears, select the schema file to be executed and click the **Finish** button. This dialog automatically populates the

schema file available under the cartridge's code generation directory, which is the required schema file under the normal circumstances.
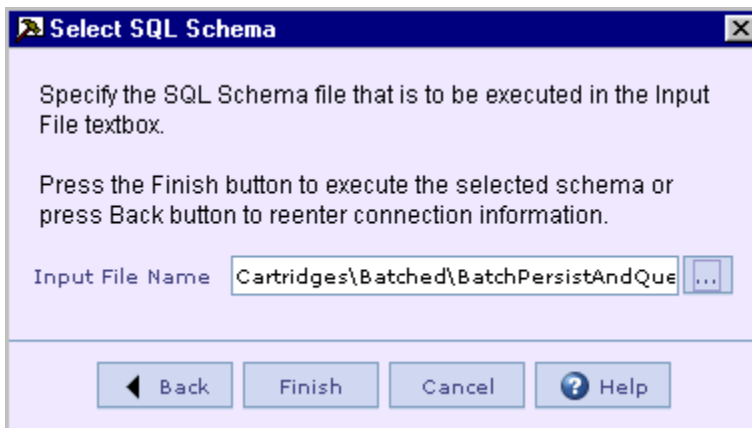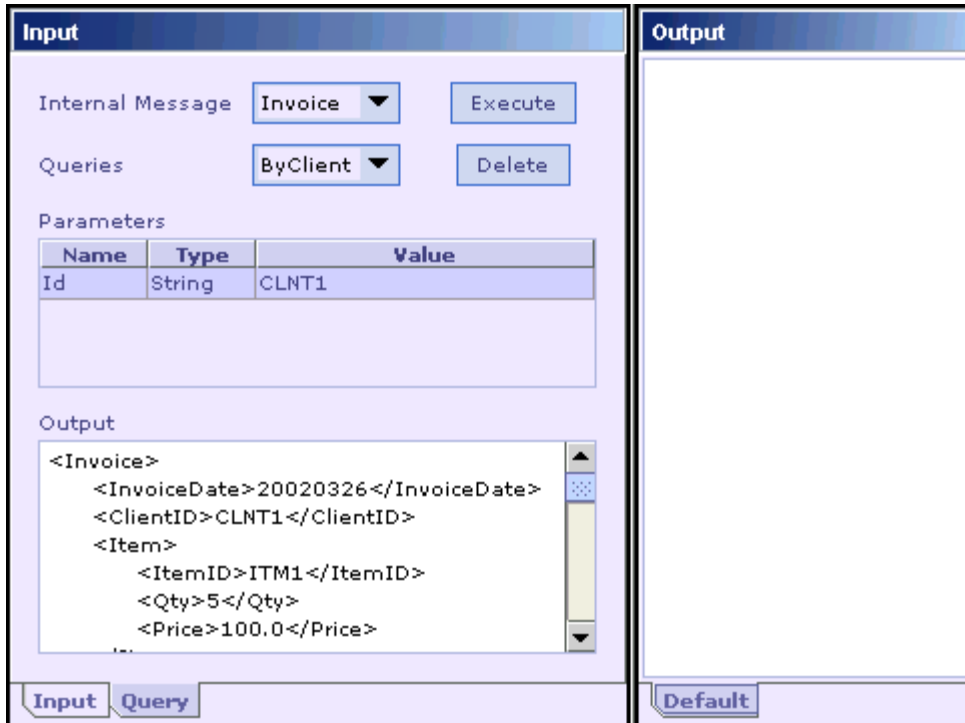

The schema file gets executed. The message window shows the SQL statements executed.

See Also:

Testing a Cartridge with Persistence Support

Invoking Queries

The **Query** tab in the Input pane of Simulator is used to invoke queries defined in the currently deployed cartridge.

Follow the steps given below to invoke a query:

1. Select the internal message available in the cartridge from the **Internal Message** combo box.

2. Select the query to be executed from the **Queries** combo box.

3. The combo lists the queries defined already using Designer. It also includes a query named 'All' that returns all internal message records.

4. In case of a query with parameters, specify parameter values to be passed to the query. In the above figure, the value *CLNT1* is provided for the parameter 'Id'.

5. Click on the **Execute** button to execute the query.

6. The query result is displayed in the **Output** text area.

Note:

Records matching a query can be deleted by clicking the **Delete** button.

See Also:

Testing a Cartridge with Persistence Support

Testing a Cartridge with Persistence Support

The following steps should be followed to test a cartridge having persistence support.

1. In the cartridge, add Persistence Designer and design the database tables.

2. Add queries if needed.

3. Specify the data source to be used for persistence in Java Code Generation **Settings** dialog.
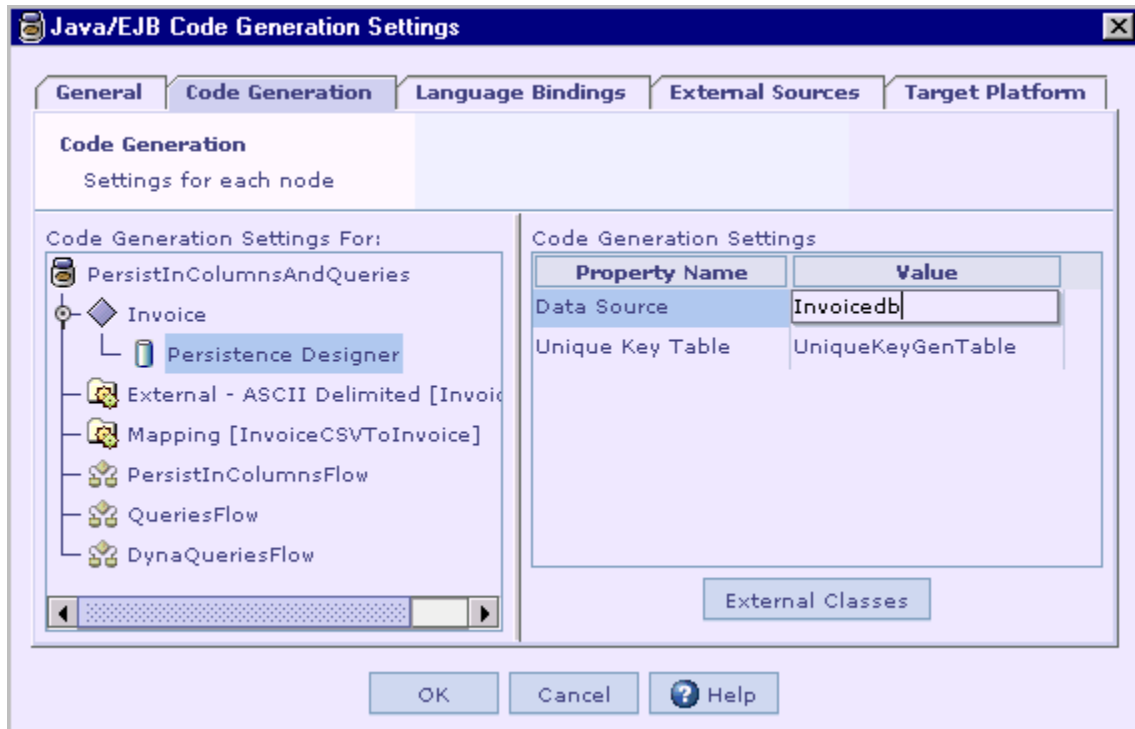
4. Generate code for the cartridge.

5. Launch Simulator or just click the **Redeploy Cartridge** button in Simulator toolbar if it is already launched.

   Map to the required Simulator data source.

6. If the data source mapped to is other than the default data source (simulatordb is started by default), make sure the database server is running.

7. Create the schema to persist the internal messages.

8. Select the message flow, message or message mapping to be executed.

9. In case of a message, select the processing that needs to be done. This can be one of 'Parse', 'Parse and Validate' and 'Parse and Write'.

10. Enter the data into the Input pane or open an input test file using the **File > Open Input from File** menu item.

    In case you have selected a message flow with multiple inputs, the Input pane is laid out as a form with text field and/or text area input elements based on the type of the corresponding input variable. For a simple type input variable, a text box is used to accept the input and for a 'Binary' type input variable, a text area is used to accept the input. To enter input into a text area by opening a data file, first select that input area and then open the input data file by using the **File > Open Input from File...** menu item.

11. Click the **Submit** button to start transformation.

12. The output of the transformation and/or errors generated is displayed in the **Output** and the **Message** pane respectively.

    To verify that the records corresponding to the internal message are inserted into the database, use the **Query** tab of the **Input** pane. Refer Invoking Queries to know about how to invoke queries in Simulator.

See Also:

Cartridge Simulator

Test Data Generation

The test data generation utility helps the user to generate test cases for various data formats (Currently only the internal formats of internal messages and external formats of SWIFT and XML are supported). This feature is now made available as part of Simulator.

The user can generate a single test case shown in the Input pane of Simulator. There is also an option to generate a set of test data files in the directory specified by the user.

In case of SWIFT, the generated test cases are based on the SWIFT format specification. The format specification for SWIFT has information about the type (character set) as well as the maximum or exact length of each field. So the generated test cases are reasonably useful.

Unlike SWIFT, the other formats (XML, Internal Format etc) are weakly specified. For instance, in case of XML format, a field of type String is unconstrained. To constrain the values generated for the field, the user is allowed to specify a pattern. The

values generated for the field will conform to this pattern. See the section [Defining Patterns](#) for further information. The pattern syntax is similar to the Regular expression syntax supported by JDK1.4. But instead of matching a string against the pattern, a random string is generated to conform to that pattern.

The first step in test data generation is defining the data generation specification (optional). For this the user is shown a dialog based on the selected data format. The dialog allows the user to define data generation specifications for the elements of the data format.

In case of simple fields, the specification is a pattern. See the section [Defining Patterns](#) for further information.

In case of sections, the specification is the minimum and maximum occurrences of the section (currently not implemented).

Normally, once [data generation specification](#) is provided, the user can proceed with the actual [data generation](#). It is possible to directly proceed to data generation without providing data generation specification. In this case, data generation is based on random values depending on the corresponding data type.

See Also:

[Defining Data Generation Specification](#)
[Generating a Test Case](#)
[Generating Test Data Set](#)
[Running Test Data Set](#)
[Entering Test Data for Internal Messages](#)
[Testing with Simulator](#)

Defining Data Generation Specification

Follow the steps given below to define data generation specification for the elements of the required data format.

1. Open and generate the cartridge that contains the data format for which you want to generate test cases.

2. Start Simulator.

3. Select the internal/external message for which you want to generate test cases.

4. If you select an internal message design element, you can generate test cases for the selected internal format. Similarly, if the selected design element is an external message, test cases can be generated for its external format. Currently

only the internal formats of internal messages and external formats of XML external messages are supported.

5. From the Simulator menu bar, select the **Test > Define DataGen Spec** menu item. The **Data Generation Specification** dialog is displayed as shown below.
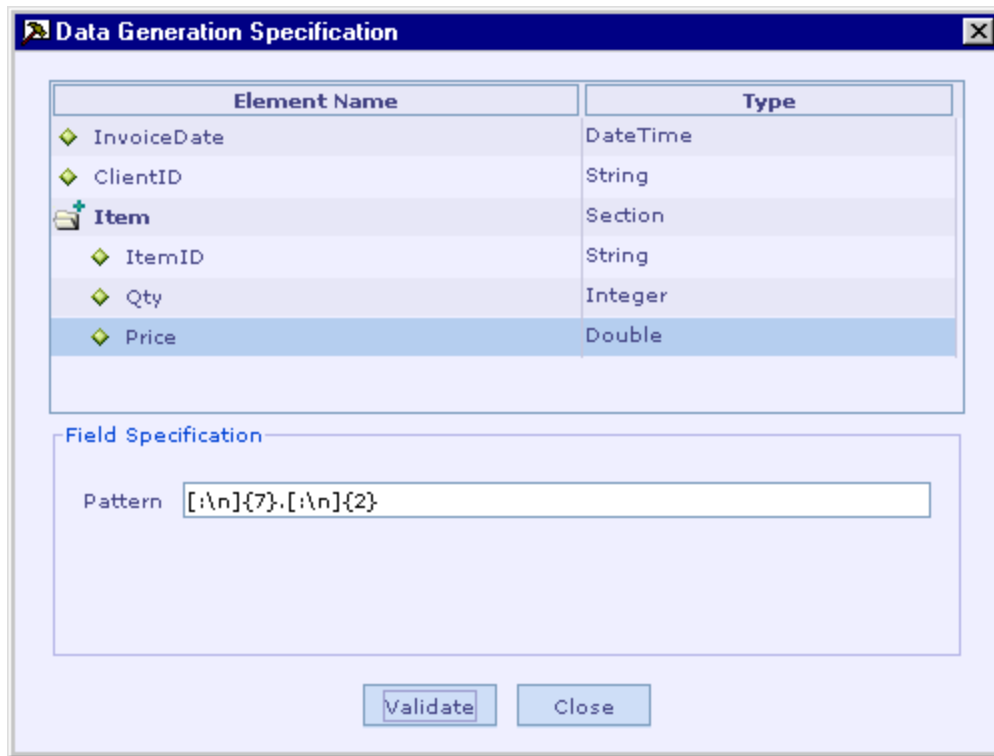
6. Select the field/section for which you want to specify data generation pattern.

7. Specify the data generation pattern for the currently selected field/section in the **Pattern** text field of the **Field Specification** section. See the section Defining Patterns for further information.

8. Once you have completed defining data generation specification, click on the **Validate** button to verify the validity of patterns specified.

9. Click on the **Close** button to close the dialog.

See Also:
Generating a Test Case
Generating Test Data Set
Running Test Data Set
Entering Test Data for Internal Messages
Testing with Simulator

Defining Patterns

In the **Pattern** text field of the **Data Generation Specification** dialog, the user can specify either a single pattern or a list of patterns. If the user specifies a list of patterns, one of them is randomly selected for data generation. In case of a list, a comma should separate each pattern.

The following points should be noted when defining a pattern.
    A pattern can be a value itself.
    A pattern can be a regular expression.
    Be careful when you use a regular expression so that it generates a value of the correct data type. Otherwise, it results in error during data generation rather than during specification. Refer to the [Pattern Grammar](#) section for further details.
    Except for String and Boolean data types, a pattern can also be a range.
    A pattern can also be empty. Two consecutive commas stand for an empty pattern. An empty pattern generates a null value.

The following is a list of valid pattern specifications:

String
SD199401, \:a{4}(//\:n{3})?, , \:x{7}
Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec

Integer
1970-1979, 2000, 2003

Long
75[\:n]{3}, 85000, 100000-150000

Double
123456.789-123456.799, 999999.999, [\:n]{7}.[123]{2}

Float
10000.50-20000.25, 35000.00, 50000.00, 100[\:n]{3}.[123]{2}

Character
a-z, 1-5, {, L, \:a

Date
1900121[0123], 1995120[\:n&&[^0]], 20011201-20011210

Note:

We have extended the Regex syntax to easily support SWIFT character set. Predefined character classes in the pattern syntax represent SWIFT character sets.

\:x refers to X char set
\:a refers to A char set
\:n refers to Numeric char set

Pattern Syntax

Character classes

[abc]   a, b, or c (simple class)

[^abc]   Any character except a, b, or c (negation)

[a-zA-Z]   a through z or A through Z, inclusive (range)

[a-d[m-p]]   a through d, or m through p: [a-dm-p] (union)

[a-z&&[def]]   d, e, or f (intersection)

[a-z&&[^bc]]   a through z, except for b and c: [ad-z] (subtraction)

[a-z&&[^m-p]] a through z, and not m through p: [a-lq-z](subtraction)

Predefined character classes

\d   A digit: [0-9]

\D   A non-digit: [^0-9]

\s   A whitespace character: [\t\n\x0B\f\r]

\S   A non-whitespace character: [^\s]

\w   A word character: [a-zA-Z_0-9]

\W   A non-word character: [^\w]

SWIFT character classes

\:n   SWIFT numeric character: [0-9]

\:a   SWIFT charset A

\:c   SWIFT charset C

\:x   SWIFT X charset

\:y   SWIFT Y charset

\:z   SWIFT Z charset

\:h   Hex char

Logical operators

XY   *X* followed by *Y*

X|Y Either *X* or *Y*

(*X*)   X, as a capturing group

Cardinality

X?   *X*, once or not at all

*X**   *X*, zero or more times

X+   *X*, one or more times

X{*n*}   *X*, exactly *n* times

X(*n*,}   *X*, at least *n* times

X{*n,m*}   *X*, at least *n* but not more than *m* times

Examples

Pattern	Generated Value
Abcd	abcd
a?	0 or 1 'a'
a*	0 or more a's
a+	1 or more a's
a{3}	aaa

a{3,5}	aaa or aaaa or aaaaa
a[bcd]	ab or ac or ad
\:n	One numeric character
\:a{3}	Three alpha characters
[abc[def]]	One character among 'abcdef '(union)
[abc&&[bcd]]	One character among 'bc' intersection
Ab{2}	abb (the cardinality applies to the last entity)
(ab){2}	abab (ab is grouped)
a|b	a or b
a|b{2}	a or bb
(a|b){2}	aa, bb, ab or ba
[\:n&&[^0]]	Numeric character except zero
\:x{16}	Equivalent to 16!x
\:x{0,16}	Equivalent to 16x
\:a{4}(//\:n{3})?	Equivalent to 4!a[//3!n]

See Also:

Pattern Grammar

Character   -> SimpleCharacter | CharacterClassProd

SimpleCharacter   -> any character (0-128)

CharacterClassProd   -> StartBrac [Hat] CharacterClass EndBrac

CharacterClass   -> PredefinedClass | UserdefinedClass | IntersectionClass |
                         UnionClass

IntersectionClass   -> CharacterClass && CharacterClassProd

UnionClass   -> CharacterClass CharacterClassProd

UserDefinedClass   -> SimpleCharacter*

PredefinedClass   -> Colon PredefinedClassChar

PredefinedClassChar -> a | n | x | y | z | h

CharacterGroup   -> (CharacterSequence)

CharacterOR   -> CharacterSequence | CharacterSequence

RepeatedCharSequence   -> CharacterSequence | CharacterSequence * |
                          CharacterSequence ? | CharacterSequence + |
                          CharacterSequence {n} | CharacterSequence {n,} |
                          CharacterSequence {n,m}

CharacterSequenceList   -> CharacterSequence CharacterSequence

CharacterSequence -> Character

CharacterSequence -> RepeatedCharSequence

CharacterSequence -> CharacterGroup

CharacterSequence -> CharacterOR

CharacterSequence -> CharacterSequenceList

See Also:

Generating a Test Case

Follow the steps given below to generate a single test case in the Input pane of Simulator.

1. Open and generate the cartridge that contains the data format for which you want to generate test cases, if you have not done so before.

2. Start Simulator, if you have not done so before.

3. Select the internal/external message for which you want to generate a test case.

4. Make sure that you have defined data generation specification for that design element. Otherwise, the generated test case is based on random values depending on the data type of the corresponding element. See the section Defining Data Generation Specification for further information.

5. From the Simulator menu bar, select the **Test > Generate Test Case** menu item to generate a test case.

See Also:

Defining Data Generation Specification
Generating Test Data Set
Running Test Data Set
Entering Test Data for Internal Messages
Testing with Simulator

Generating Test Data Set

Follow the steps given below to generate a set of test data files in the directory specified by the user.

1. Open and generate the cartridge that contains the data format for which you want to generate test cases, if you have not done so before.

2. Start Simulator, if you have not done so before.

3. Select the internal/external message for which you want to generate a test case.

4. Make sure that you have defined data generation specification for that design element. Otherwise, the generated test cases are based on random values depending on the data type of the corresponding element. See the section Defining Data Generation Specification for further information.

5. From the Simulator menu bar, select the **Test > Generate Test Data Set** menu item. The 'Generate Test Data Set' dialog is displayed.
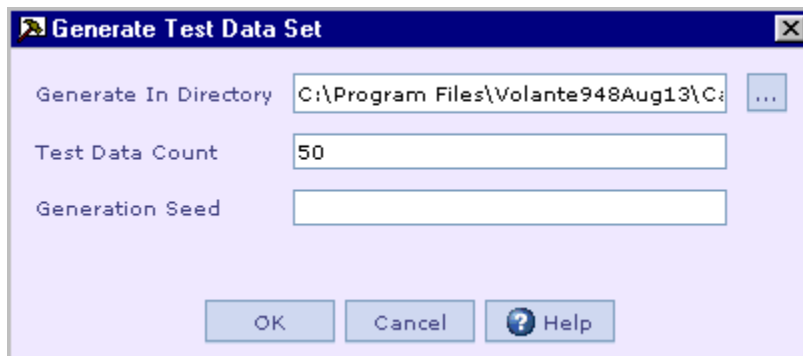
6. In the **Generate in Directory** text field, specify the directory where the generated test data files will be written.

7. In the **Test Data Count** text field, specify the number of data files to be generated.

8. In the **Generation Seed** text field, specify the random seed to be used in data generation. If not specified, the default value is used.

9. Click on the **OK** button to start generation.

See Also:

Defining Data Generation Specification
Generating a Test Case
Running Test Data Set
Entering Test Data for Internal Messages
Testing with Simulator

Entering Test Data for Internal Messages

The user can generate an internal message test case just by entering data for the fields of that internal message without worrying about XML syntax by using the 'Data Entry' tool provided in Simulator.

Follow the steps given below to generate an internal message by using the 'Data Entry' tool.

1. Select the internal message for which you want to generate a test case.

2. Select the **Tools** > **Data Entry** menu item.

3. The data entry dialog appears with internal message name as the title.

4. Enter values for the fields. Fields indicated by * are mandatory.

5. Nested fields within a section are grouped under the section name. In the above figure, fields 'ItemID', 'Qty' and 'Price' are grouped under section 'Item'.

   Add more elements of any repeating section, if required.

6. For repeating sections, when the cursor is placed inside the section group, a button panel appears. The button panel has buttons for adding an element of the repeating section, deleting an element and moving back and forth between the elements of the section.

7. Click OK after entering all the required values.

The generated test case is written to the **Input** pane of Simulator.

See Also:
Defining Data Generation Specification
Generating a Test Case
Generating Test Data Set
Running Test Data Set
Testing with Simulator

Frequently Asked Questions

1. When I press the **Submit** button, no output or error is produced?

Check whether you have added router or some other trigger to your internal message. Check if the message flow that is executed has an output variable defined in it or the flows sends the output to an output device.

2. I get a runtime error that the output device is not found similar to the one given below.

```
<TransformException>
    <Type>TransformException</Type>
    <Message>Unexpected error</Message>
    <Severity>1</Severity>
    <Cascadable>false</Cascadable>
    <Error-Phase>Internal Message</Error-Phase>
    <StackTrace>
    null
Type=TransformException, Message=Unable to lookup RMI output device
'device.GRNOutXML', ErrorCode=null, Severity=1, Cascadable=false
```

Your application attempts to write to a different output channel. You need to add an output channel in Simulator.

3. My application uses external jars (libraries). How do I test with Simulator?

Copy the external Jars to the directory in which jars were generated by the code generator. Simulator will automatically load the dependent jars referenced using manifest classpath. Remember to refer to these jars in the manifest classpath of the generated jars using **Java Code Generation Settings** dialog. A sample is shown below.
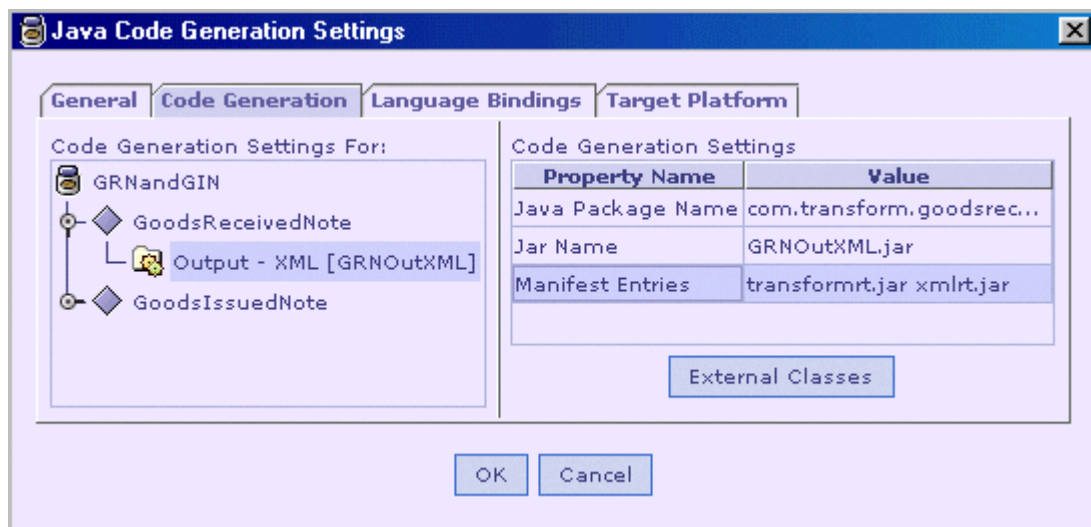
4. I get the error message "Error generating Unique key".

Verify that you have created **UniqueKeyGenTable** in the datasource referred in the cartridge. See the PersistenceDesigner.doc for further information.

5. I get the error message "Table not found".

    If the table mentioned in the error is other than UniqueKeyGenTable, verify that you have created the schema used in the current cartridge. Refer Creating Schema to know how to create the schema.

6. How do I use database other than Simulator database?

    Configure the datasource for Simulator.
    Add the database driver jars to *<installation dir>\lib\ext* directory.
    Create the schemas in the database you intend to use.
    See Testing a Cartridge with Persistence Support for more details.

7. I have added a Persistence Designer but nothing is saved in the database. No error is reported.

    Verify if you have added the Persistence Trigger.

8. I have made changes in the cartridge. But these changes are not reflected in Simulator.

    Make sure that you have generated code and redeployed the cartridge. Redeploy the cartridge using the **Redeploy Cartridge** button or the **Tools > Redeploy Cartridge** menu item in Simulator.

9. I get the error message "Input Parsing Error".

    Make sure that your input data corresponds to the message flow, message or message mapping selected for execution.

10. I have defined only the internal/external message in the cartridge. How do I test it?

    Select the internal/external message for execution.
    Select the 'Parse' from the Output Options drop down list.
    Click **Submit** button. The parsed structure is displayed in the Output pane.

    For external messages, two additional options are populated in the Output Options drop down list – 'Parse and Validate' and 'Parse and Write'. The former dumps an XML structure of the parsed input while the latter dumps the output in the plug-in format. In case of the 'Parse and Validate' option, the input message

is parsed and validations specified for that message are carried out. In case of the 'Parse and Write' option, apart from parsing and writing of the message, validations are carried out on the parsed input object and on the output object before writing the output.

See Also:

[Simulator](#)

Debugging

The process of locating and fixing errors in your application is known as debugging. It includes breakpoints, watch expressions, and the ability to step through code/model one statement or one procedure at a time and display the values of variables and properties

To understand how debugging is useful, consider the kinds of errors that can occur

Runtime errors   These occur while the application is running when a statement attempts an operation that is impossible to carry out. An example of this is division by zero.

Logic errors   These occur when an application doesn't perform the way it was intended. An application can have syntactically valid code, run without performing any invalid operations, and yet produce incorrect results. Only by testing the application and analyzing results can you verify that the application is performing correctly.

These kinds of error can be reduced using debugging.

Debugger requires some additional information to be available at runtime so that the execution state can be inspected. The Java Code generator has an option to generate this additional information. The 'Debug Info' check box present in the 'Build➔Code Generation Settings' dialog, should be enabled before generating code for the cartridge. For details on how to enable this option and to run a cartridge under the debugger refer to the section [Debugging from simulator](#).

See Also:

[Debug Window](#)
[Break Points](#)
[Step modes](#)
[Frames](#)
[Watches](#)
[Debugging From Simulator](#)

Debug Window

To view the 'Debug' window, click on the 'View Debug window' menu in 'Run menu'.
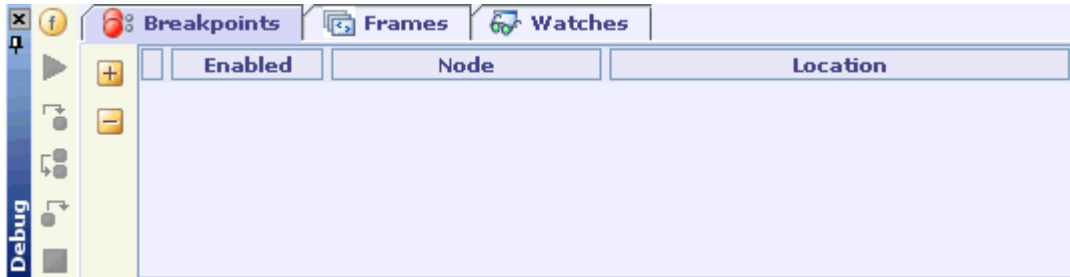
![Go button icon] 'Go' button when clicked, moves to the location of Breakpoint in Execution. If there is no Breakpoint set, the button when clicked completes Debugging.

![Show Null Fields button icon] 'Show Null Fields' button when clicked, hides the fields with null value i.e.,when this button is enabled ' ', fields without any value are not listed in the 'Frames' tab.

![Step Into button icon] 'Step Into' button when clicked, runs the next executable location (an activity in a message flow, line of code in functions, a validation rule or a mapping) and steps into procedures.

![Step Over button icon] 'Step Over' button when clicked, runs the next executable location (an activity in a message flow, line of code in functions, a validation rule or a mapping) without stepping into procedures.

![Step Out button icon] 'Step Out' button when clicked, runs the remainder of the current execution and breaks at the next line of execution.

![Stop button icon] 'Stop' button when clicked, terminates the debugging session.

In the 'Breakpoints' tab, the user can add/remove break points for a location (an activity in a message flow, line of code in functions, a validation rule or a mapping) in the cartridge.

In the 'Frames' tab, the user can view the values of all variables during debugging.

In the 'Watches' tab, the user can monitor the value of any variable, property, object, or expression when the code is executed.

When the Message window is open, it can be closed by clicking on the Close button present in its title bar.

The Auto Hide mode (Pinned mode) of the Message Window can be enabled or disabled, by toggling the Auto Hide button present in its title bar. When Auto Hide mode is enabled (indicated by the ⊨ icon), the Message Window hides itself when it loses focus. When Auto Hide mode is disabled (indicated by the ⊥ icon), the Message window remains visible even when it loses focus.

See Also:

[Break Points](#)
[Step modes](#)
[Frames](#)
[Watches](#)
[Debugging From Simulator](#)

Breakpoints

A breakpoint is a location (an activity in a message flow, line of code in functions, a validation rule or a mapping) at which Designer automatically suspends execution.

While running under the debugger, a breakpoint will stop your code on the line of code that will be executed next. Once execution has stopped, you can investigate your code/model and the variables in the enclosing context to get a better understanding of the execution context.

Note:

To navigate to the location for which Breakpoint is created, the user can also select a row in the Breakpoints tab, right click it and select 'Jump to Location' menu item.
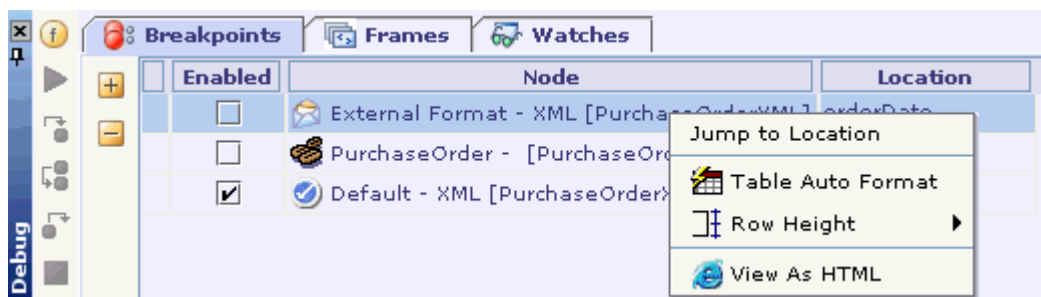
To view the Breakpoints in the HTML format right click inside the Breakpoints tab and select 'View As HTML' menu item. The search results are displayed in HTML form.

See Also:

[Adding Breakpoint](#)
[Deleting Breakpoint](#)

Adding Breakpoint

You can set Breakpoints in Designer for,

Message Flow
Mapping
Function Definitions
External Message
Internal Message
Validation

In the message flow UI select the location (an activity in a message flow, line of code in functions, a validation rule or a mapping) for which Breakpoint is to be added. In case of message flow, the location can be an activity. In case of mapping the location can be a 'row' in the mapping table (destination).

From the 'Run menu' in menu bar select 'Toggle Breakpoint'.
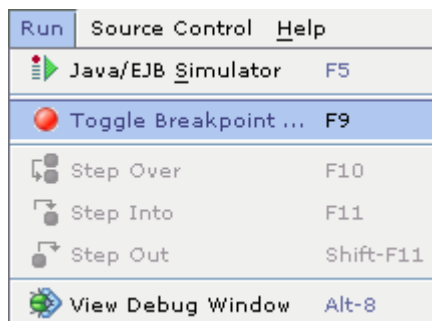
The selected activity appears with the breakpoint icon
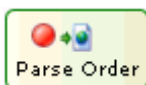
In Mapping, the selected row appears with the breakpoint icon as follows

The added breakpoints are also listed in the 'Breakpoints' tab of 'Debug' window.

Breakpoints can also be added by selecting the location (an activity in a message flow, line of code in functions, a validation rule or a mapping) and pressing F9 or by clicking 'Add Breakpoints' icon '⊞' in the Breakpoints tab of Debug window.

You can navigate to the breakpoint location by clicking on the hyperlink in the "Node" column.

Note:

When a cartridge is saved, the breakpoints added are also automatically saved.

See Also:

Break Points
Deleting Breakpoint
Enable/Disable Breakpoint

Deleting Breakpoint

Follow the steps given below to delete a Breakpoint using 'Toggle Breakpoint' menu.

1. Select the location with Breakpoint.

2. Press F9 or the 'Toggle Breakpoint' menu in 'Run menu'.

Follow the steps given below to delete a Breakpoint using 'Delete Selected Breakpoint(s)' icon.

1. In the 'BreakPoints' tab of Debug window, select the location for which Breakpoint is to be removed.

2. Click the 'Delete Selected Breakpoint(s)' icon⊟.

See Also:

Break Points
Adding Breakpoint
Enable/Disable Breakpoint

Enable/Disable Breakpoint

When you uncheck the 'Enabled' column in Breakpoints tab, the Designer would not suspend the execution even if the location has a breakpoint.
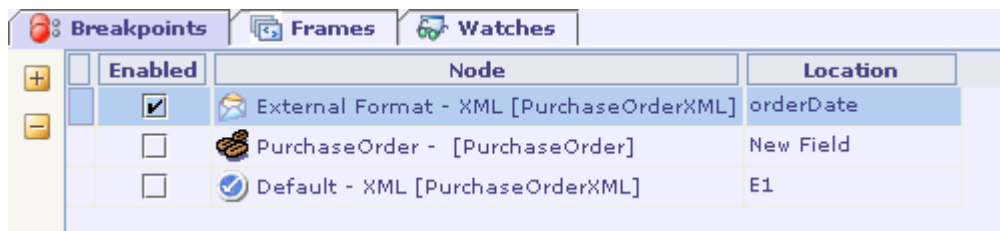
In the above figure breakpoint has been disabled for mapping of Records.Cost. So execution will not be suspended for the mapping and will proceed to the next mapping code to be executed.

Note:

If the Enabled column of a Breakpoint is unchecked, the Breakpoint icon in the corresponding Location appears as faded.

🔴　Breakpoint 'Enabled' state

🔴　Breakpoint 'Disabled' state

See Also:

Break Points
Adding Breakpoint
Deleting Breakpoint
Debug Window

Step modes

You can use the three step modes to start execution of a location (activity in case of flow, validation rule or mapping) or to continue execution after it has stopped at a breakpoint. The three available modes are [Step Into](#), [Step Over](#) and [Step Out](#). Using any of these modes to step through code does not take the code out of break mode.

Step Into

Executes till the next executable location (activity, validation rule or mapping). If the current location has child steps (for example mapping activity) control descends into it. When you use Step Into to step through code one statement at a time, Designer temporarily switches to run time, runs the current statement, and advances to the next statement.

Click the '🖥' button in the Debug window to step into the next location. You can also use the short key 'F11'.

See Also:

[Step Over](#)
[Step Out](#)
[Debug Window](#)

Step Over

Step Over is similar to [Step Into](#), except when the current location (activity, validation rule or mapping) to be executed is a compound location (function call, mapping activity etc). Unlike Step Into, which steps into the called procedure, Step Over runs it as a unit and then steps to the next statement in the current node.

For e.g. if a [function](#)that is being debugged contains a call to another function, and Step Over is done, then the called function is executed as a single unit, without stepping into it.

Click the '🐾' button in the Debug window to step over the next location. You can also use the short key 'F10'.

See Also:

[Step Into](#)
[Step Out](#)
[Debug Window](#)

Step Out

Step Out advances past the remainder of the code to be executed in the current node. The remaining code in the current location is executed as a single unit. It then advances to the statement immediately following the one that called the current location.

Click the '⬚' button in the Debug window to step out after the current location is executed. You can also use the short key 'Shift+F11'.

See Also:

Step Into
Step Over
Debug Window

Frames

This window provides a snapshot view of the values of all variables when debugging is being done. The Frames tab has a combo, which displays the call stack at the point where execution has paused. Each element in combo represents a stack frame; you switch to a frame by selecting it from the combo.

The variables (local and instance) that are associated with a stack frame are displayed as a tree.

Note:

Right clicking inside the Variables tree displays a popup menu with items, **Copy Name**, **Copy Qualified Name and Copy Value**. While **Copy Name** menu item copies the name of the selected variable, the **Copy Qualified Name** menu item copies the name with its qualified path. For e.g. for the variable 'OrderID', the name is "OrderID" when copied using 'Copy Name' menu item and the name is "messageObj.Data.Records.[0].OrderID" when copied using 'Copy Qualified Name' menu item.

Copy Value menu item copies the value of the selected variable.

If the variable is a 'Section or a 'Message variable', the Copy Value menu item copies the value in XML format.

See Also:
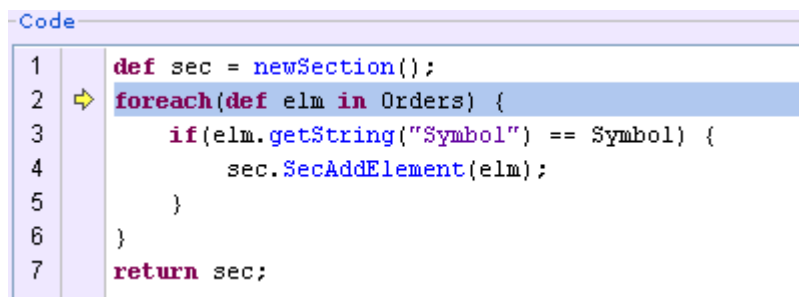
Frame Variables

The variables displayed in the window depend on the entity (External message, mapping, validation, message flow, function definition) that is being debugged.

Function Definition Variables

In case of Function Definition, the values of the input parameters to the flow are displayed as name value pairs. Any other local variable defined in the function definition is also displayed.

The combo box in the 'Frames' tab displays the line number of code currently executed in debugging.

The above function takes variables 'Orders' and 'Symbol'. A local variable 'sec' has been defined in it. Looping is done through each element of Orders and if the 'symbol' of an element matches the 'Symbol' value passed, it is added to the 'sec' variable.

In the above Function Definition code, the second line of code is being executed in Debugging. The combo box in the 'Frames' tab appears with the current location(Code in Function Definition) of debugging.
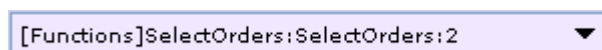
As mentioned above, the value in the combo box is of the form "[Functions]<Function Name>:<Function Name>:<Line Number In Formula>". For

e.g. in "[Functions]SelectOrders:SelectOrders:2", 'SelectOrders' is the Function definition name and '2' is the next line number of code to be executed.

The input arguments 'Orders' and 'Symbol' are also displayed in the 'Frames' tab. The 'sec' variable is currently empty. The corresponding Debug window appears as follows.

When debugging is completed for the above-mentioned code, the variable 'sec' is displayed along with its values.
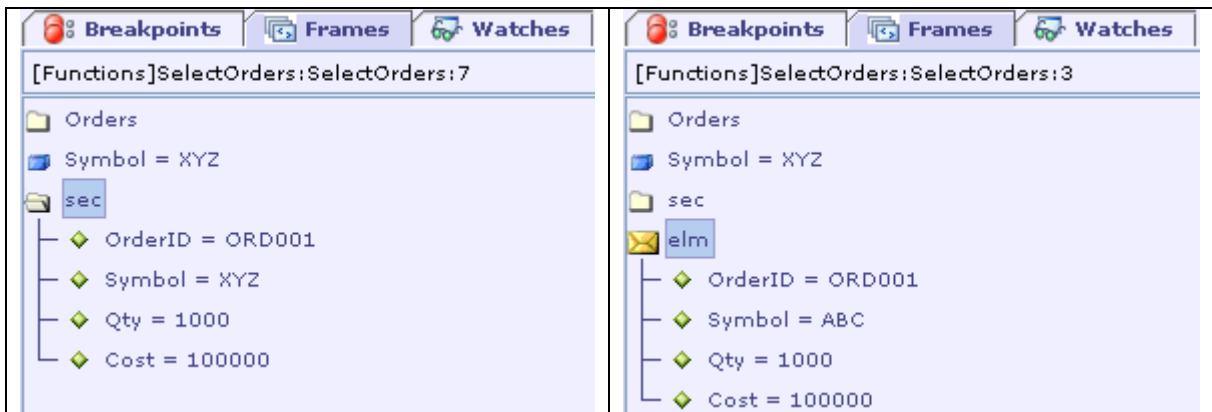
It can be seen that 'sec' has been populated from Orders. There is also a local variable 'elm' which represents a particular element in Orders. This variable is also displayed. While debugging if any exception occurs in the formula code, the debugging session is terminated and the corresponding action message is displayed in the 'Message' pane of Simulator.

Note:

The arrow mark ⇨ highlights the current location (activity in case of flow, validation rule or mapping) of debugging.

See Also:

Message Mapping Variables
External message Variables
Validation Rules Variables

Message Mapping Variables

When debugging a 'Mapping', the source mapping object and the destination-mapping object are displayed as variables 'Source' and 'Destination' in the 'Frames' tab. For example, consider the following mapping.

When the above mapping is debugged, the combo box in the 'Frames' tab appears with the current location (Mapping) of debugging.

The value in the combo box is of the form "[<Mapping Name>]map:<Mapping Field Name>". For e.g. in "[OrderToOrderCost]map:Records.OrderID", 'OrderToOrderCost' is the name of mapping and 'Records.OrderID' represents the field currently debugged. The source format and Destination format initially appears as follows.

The 'Source' variable displays the source format that is to be mapped. The fields in the source format are displayed as name=value pairs, where 'name' is the field name. The variable 'Destination' is initially empty.

'Source' format before Debugging
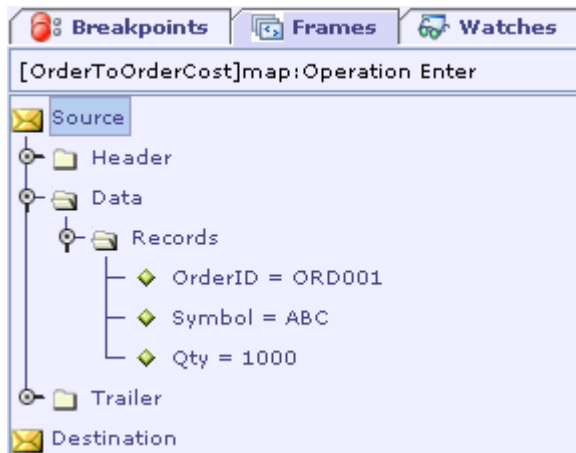
'Destination' format before Debugging

As the mapping proceeds, the Destination variable is listed with values that have been mapped. If the mapping code has any local variable defined in it, it is also displayed.

'Source' format after Debugging
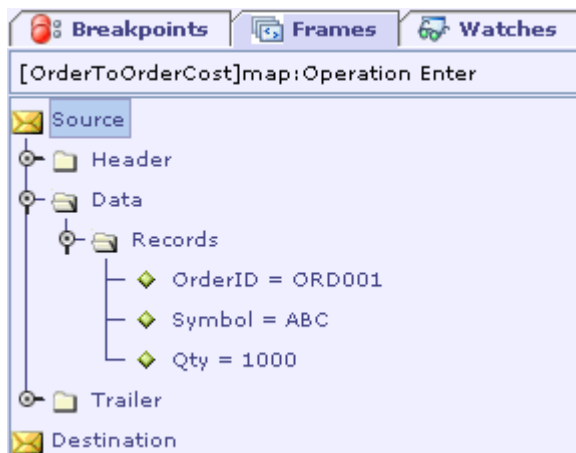
'Destination' format after Debugging
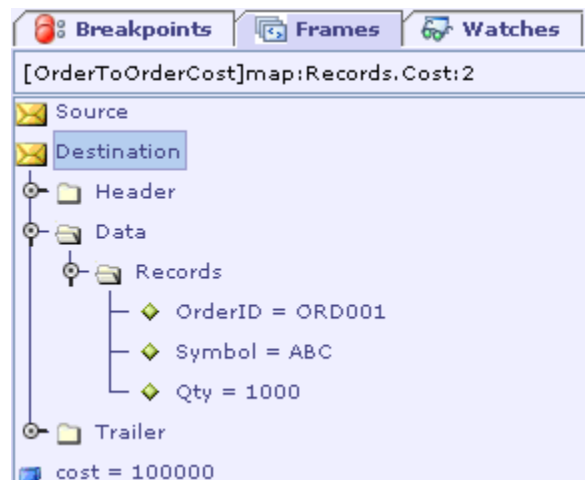
The 'Destination' format is filled with records after Debugging.

In the mapping for Records.Cost a local variable 'cost' has been defined. The value of this variable is calculated based on Record.Qty and set as the value for Records.Cost.

As can be seen from the above diagram, the local variable 'cost' is also listed during debugging. When debugging if any error occurs in mapping, the debugging session is terminated without any output and the corresponding action message is displayed in the 'Message' pane of Simulator.

See Also:

[Function Definition Variables](#)
[External message Variables](#)
[Validation Rules Variables](#)
[Message Flow Variables](#)
[Step modes](#)
[Debug Window](#)
[Debugging From Simulator](#)

External message Variables

When an external message is being debugged, a variable 'input' is displayed which contains the value of the raw input. If there are multiple records in the input they are displayed as a single string.

The 'Frames' tab appears with a combo box containing the value of current location(Message) of debugging.

```
[Order]parse:Operation Enter              ▼
```
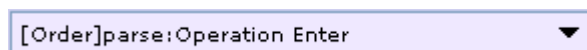
As can be seen from the above diagram, the value in the combo box is of the form "[<Message Name>]parse:Operation <Mode>". For e.g. in "[Order]parse: Operation

Enter", 'Order' is the External message name and 'Operation Enter' represents that debugging is about to start for the external message.

A variable 'messageObj' is displayed which is initially null. This variable is used to display the processed output.
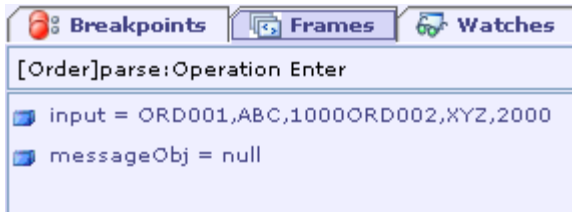
As the message is processed the 'messageObj' is populated with values of fields that have been processed. The values are displayed as name=value pairs, where 'name' is the field name.
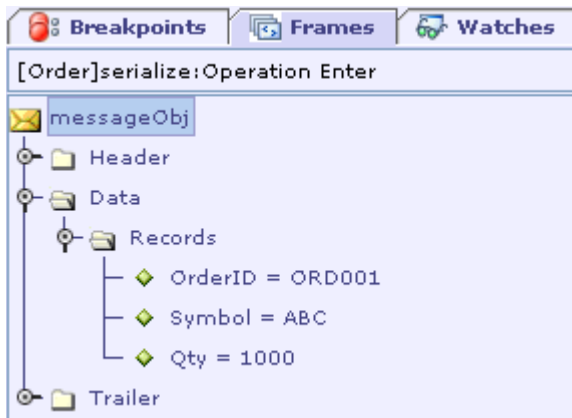
When debugging if parsing error occurs and the debugging session is terminated. The corresponding action message is displayed in the 'Message' pane of Simulator.

See Also:

[Function Definition Variables](#)
[Message Mapping Variables](#)
[Validation Rules Variables](#)
[Message Flow Variables](#)
[Step modes](#)
[Debug Window](#)
[Debugging From Simulator](#)

Validation Rules Variables

The message object variable that is being validated is displayed. If any validation has any variable defined in it, it is also displayed. For example consider the following Validation Rules.

The validation for the fields in the External message is defined. When debugging the above validation, in the frames window the currently debugged location (validation) is displayed in the combo box.
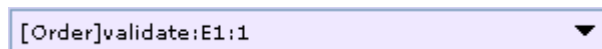
As can be seen from the above diagram, the value in the combo box is of the form "[<Message Name>]validate:<Validation Name>:<Line Number of Formula>". For e.g. in "[order]validate:E1:1", 'Order' is the message name, 'E1' is the name of validation and '1' represents the line number of code in validation 'E1' that is to be executed next.

A variable 'messageobj' appears in the 'Frames' tab which contains all the parsed input.
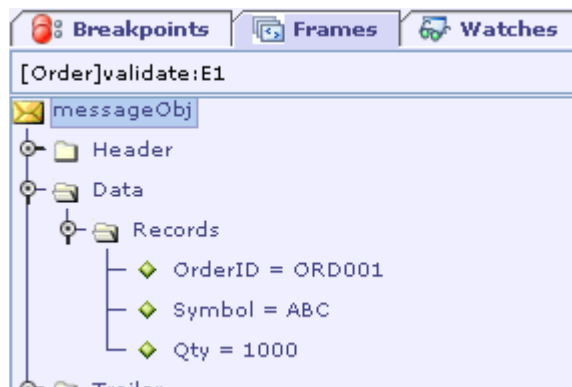
As debugging continues, the variables defined in the validation rules are also displayed in the 'Frames' tab along with the messageobj variable.

The variables 'symbol1' and 'symbol2' that was defined in the formula of validation E2 are also listed. When debugging, if the validation set for the field fails, debugging stops and the corresponding action message is displayed in the 'Message' pane of Simulator.

See Also:

Function Definition Variables
Message Mapping Variables
External message Variables
Message Flow Variables
Step modes
Debug Window
Debugging From Simulator

Message Flow Variables

When a Message flow is debugged, the variables defined in the message flow are listed in the 'Frames' tab. Initially these variables are assigned null values. As the debugging proceeds, these variables are filled with values. The values are displayed as name=value pairs, where 'name' is the field name. For example consider the following message flow.

When the above flow is debugged, the combo box in frames tab lists the location (activity in flow) that is to be debugged next.

```
[SelectOrdersFlow]run:Start1                    ▼
```
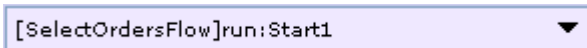
In the above diagram, the value in the combo box is of the form "[<MessageFlowName>]run:<ActivityName>". For e.g. "[SelectOrdersFlow]run:Start1" where 'SelectOrdersFlow' is the name of the message flow and 'Start1' is the name of the 'Start' activity.

Initially the variables defined in the message flow appear in the Debug window with their values initialized to null. As debugging proceeds the input values are assigned.

During debugging the variables listed in the 'Frames' tab depend on the activity that is being debugged in the message flow (Parse/Serialize/Mapping/Validation/Custom etc).

When an activity is being debugged an 'arrow' icon will appear in the activity. For e.g. when a 'Serialize' activity is being debugged it will appear as follows.
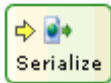
The arrow icon in the activity represents that the 'Serialize' activity is currently debugged.
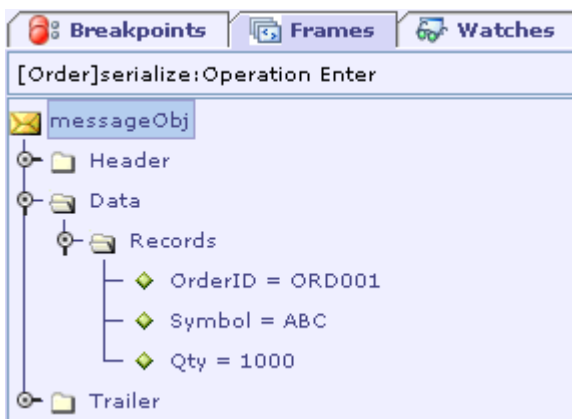
When debugging of the flow is completed, all the variables defined in the flow that were initially null will be listed again with their values.

When debugging if error occurs while processing any activity, the debugging session is terminated and the corresponding action message is displayed in the 'Message' pane of Simulator.
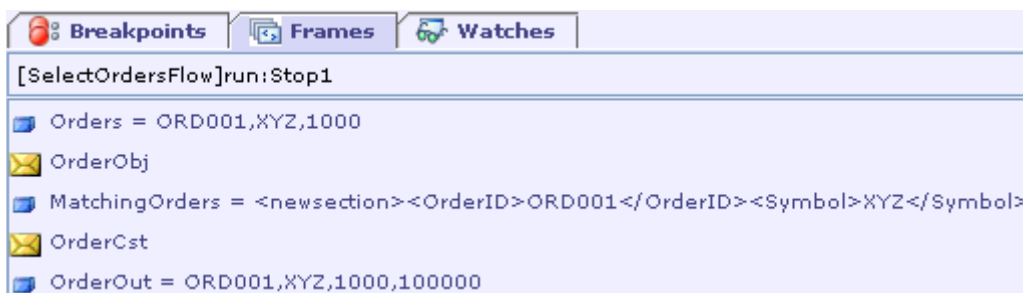
In the above figure it can be seen that the variables are listed along with their values.

See Also:

[Function Definition Variables](#)
[Message Mapping Variables](#)
[External message Variables](#)
[Validation Rules Variables](#)
[Debug Window](#)
[Debugging From Simulator](#)

Watches

Watch expressions let you monitor the value of any variable defined in a location (an activity in a message flow, line of code in functions, a validation rule or a mapping), during debugging.

Right clicking inside the 'Watches' tab displays a popup menu with items **Copy Name**, **Copy Qualified Name**, **Copy Value**, **New Watch**, **Remove Watch** and **Remove All Watches**.

While **Copy Name** menu item copies the name of the watch expression, the **Copy Qualified Name** menu item copies the name with its qualified path. For e.g. for the variable 'OrderID', the name is "OrderID" when copied using 'Copy Name' menu item and the name is "Destination.Data.Records.[0].OrderID" when copied using 'Copy Qualified Name' menu item.

Copy Value menu item copies the value of the expression.

If the expression is a 'Section' or a 'Message variable', the Copy Value menu item copies the value in XML format.

See Also:

[Adding Watches](#)
[Deleting Watches](#)
[Break Points](#)
[Frames](#)

Adding Watches

Watch expressions can be added in 'Watches' tab of Debug window by right clicking inside the 'Watches' tab and selecting **New Watch** menu item.

Select the **New Watch** menu item. The 'New Watch' dialog box appears. The name of the variable whose value needs to be monitored should be entered.

The expression is added in the Watches tab. Initially when the watch expression is created, value is not assigned. For e.g. consider the 'Destination' object created for a Mapping node and a local variable 'Cost' defined in the same Mapping. When expressions are added for these, they appear as shown below.

When debugging, initially the value of expressions in the Watches tab are assigned to null.

As debugging is proceeded for the mapping node, the expressions in the 'Watches' tab are assigned values when, values for the corresponding variables are assigned in the 'Frames' tab.
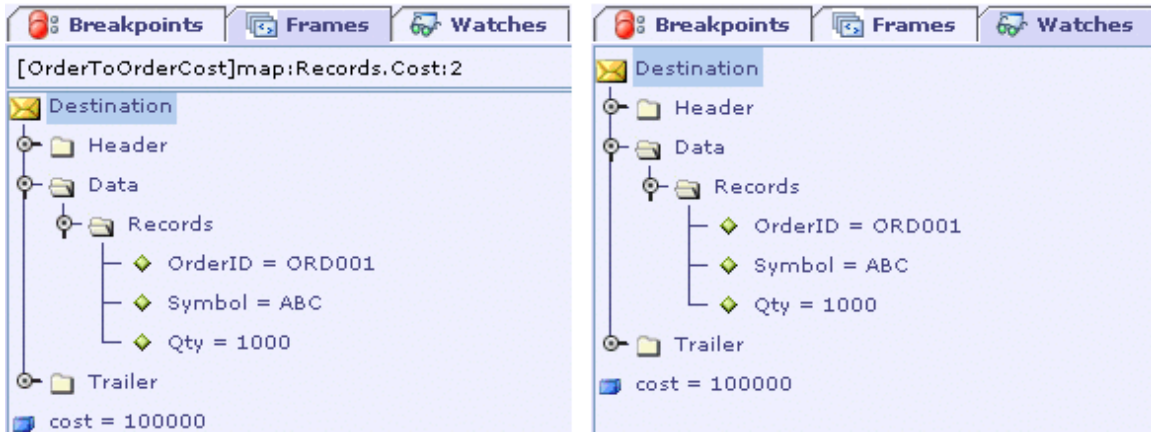
In the above figure it can be seen that the expression Destination has changed to a message variable. The expressions 'Destination' and 'cost' that were previously assigned null values are now assigned with values when the corresponding variables are assigned with values in the 'Frames' tab.

Note:

Please note that when an expression added in the watches tab a variable of the same name should have been defined in a location (an activity, line of code in functions, a validation rule or a mapping). Otherwise, the expression remains empty even after debugging.

For e.g. if a variable 'Cost' has been defined in a mapping and a expression named 'Cost1' has been added, while debugging the mapping the expression will not be populated with values. In this case an watch expression of name 'Cost1' needs to be created to monitor the value.

See Also:

Watches
Deleting Watches

Deleting Watches

Watch expressions can be removed by selecting the menu items **Remove Watch** and **Remove All Watches**.

While 'Remove Watch' menu item removes the selected expression, the 'Remove All Watches' menu item removes all the expressions in the 'Watches' tab.

See Also:

Watches
Adding Watches

Debugging From Simulator

The 'Debug Info' check box present in the 'Build➜Code Generation Settings' dialog, should be enabled before generating code for the cartridge.
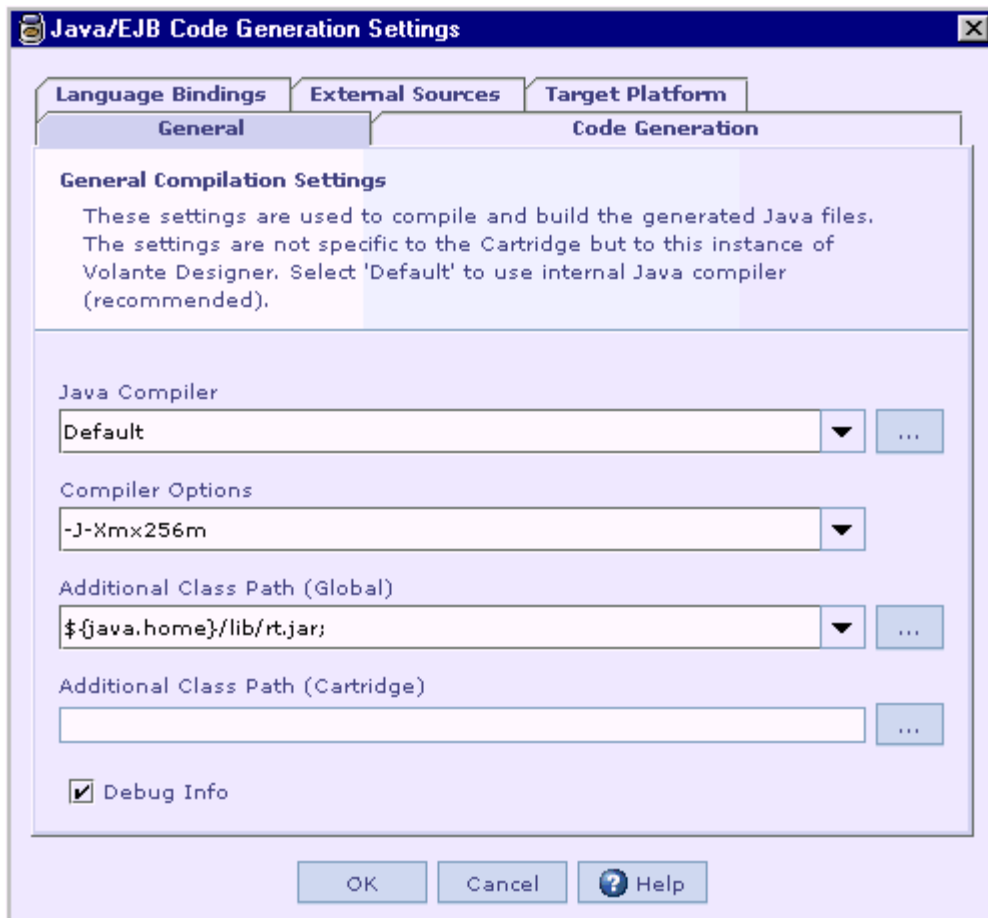
Please note that, enabling the 'Debug Info' check box is cartridge dependent. For each cartridge this option has to be separately selected.

1. Start simulator.

2. In the input combo box select the entity (external message/mapping/message flow) that needs to be debugged.

3. Enter input values in the 'Input' pane.

4. Select the 'Debug' button.

5. The focus is transferred to the '[Debug](#)' window in the cartridge

6. Click the '[Step Into](#)' button in the 'Debug' window to run the next executable location (activity in case of flow, validation rule or mapping).

7. Click the '[Step Over](#)' button, to run the next executable location without stepping into functions.

8. Click the '[Step Out](#)' button to step out after the execution of current location.

9. If a Breakpoint is set in a location, click the '[Go](#)' button to move to the location of '[Breakpoint](#)'.

   If there are multiple breakpoints, clicking the 'Go' button again takes you to the next location of Breakpoint. When there are no more breakpoints, and 'Go' button is clicked, debugging is completed and output is displayed in Simulator.

10. Click the '[Stop](#)' button to terminate the debugging session.

Note:

Output is generated in the 'Output' pane of simulator only if the debugging session is completed normally. If the debugging session is terminated using 'Stop' button or if the session terminates due to an exception, no output is generated and the corresponding action message is displayed in the 'Message' pane of simulator.

See Also:

[Debugging](#)
[Debug Window](#)
[Frames](#)
[Watches](#)

Executing Cartridge Entities

Utility to execute message flows, messages and message mappings defined in a cartridge. This utility is automatically created in the generated 'java' directory when code is generated for a cartridge in JAVA.

For e.g. if the cartridge node name is 'Employee' then an executable with name 'EmployeeTest.exe' will be generated. The generated executable can be used to test flows/messages/mappings defined in the cartridge.

Syntax

```
execute [options] name inputparam1 inputparam2 ....
```

Where

    options

-perf[:count]   Runs performance test (default count = 10000)
-threads[:count]   Runs multi thread test (default count = 5)
-nocascade   Stops the execution when an error occurs. Default is to run
                      in cascade mode

    For messages

-parse   - Parse the input file (-p)
-validate   - Validates parsed object (-v)
-write   - Writes the object (-w)

Default option while processing messages is -p -v –w

name is the name of the external message/message mapping/message flow that is to be executed.

inputparam1 inputparam2 .... is the list of input parameters.

Listing Cartridge Entities

Once code has been generated for a cartridge, you can view the entities that are present in the cartridge. Run 'execute.bat' without specifying any argument.

The list of messages/flows/mappings defined in the cartridge will be displayed.

```
Message Flows:
    execute BatchFlow rawIn(Binary) rawOut(RawMessage)
Messages:
    execute NonBatchASCII inputfile
    execute BatchASCII inputfile
Message Mappings:
    execute NonBatchASCIIToBatchASCII inputfile
```

See Also:

Executing Message Flows
Executing Messages
Executing Message Mappings
Performance measurement

Executing Message Flows

A message flow contains a sequence of processing steps that execute when an input message is received. When a message flow is executed through the execute utility, the steps defined in the flow are executed and the value of the output variable(s) defined in the message flow are dumped in the console.

For input variables of type 'Binary' and 'RawMessage' the input should be passed as a file that contains the data. For other types the input values can be passed directly.

The arguments passed to the flow should correspond to the input arguments defined in the flow.

For e.g. consider a flow that has two arguments, a binary value and an integer value. In this case, the arguments to the flow should be a file containing the data and an integer value.

```
execute <flowname> data1.txt 1000
```

The value for the integer input variable has been passed directly.

All the output variables that have been defined in the message flow are dumped after execution has been completed. The order in which the values are displayed corresponds to the order in which the output variables have been defined in the flow.

Consider a flow that has two output variables 'Id' and 'Name' defined in it. 'Id' has been defined first and then 'Name' has been defined. The output generated for the flow will contain the value for 'Id' first and then the value for 'Name'. An e.g. output for the above mentioned flow is given below.

```
1001
John
```

Note:

The name of the message flow to be executed is case sensitive. The name should be given as defined in the cartridge.

Please refer New File from Template to create a message flow client for a flow defined in the cartridge.

Specifying IN/OUT scope variables as arguments

Variables of scope 'IN/OUT' act both as input and output. While executing a flow that has variable of type RawMessage and scope 'IN/OUT', a file (file name) has to be passed as input for the variable. The output for the variable will be written to this file.

```
execute <flowname> data1.txt
```

The output will be written to the file 'data1.txt'.

It is not necessary the file data1.txt should be present while executing the flow. If the file is not present, it will be created and the output will be written to the file. If the file is present the output will be appended to the contents of the file.

See Also:
[Executing Messages](#)
[Executing Message Mappings](#)
[Performance measurement](#)
[Multi Thread testing](#)
[Executing multiple samples](#)

Executing Messages

In some cases the user may want to execute the message alone, to test it as a separate entity. Execute utility can be used to test only a message.

When a message is executed using execute utility, by default the data is parsed, validated, serialized and the serialized output is dumped.

The argument to the external message should be a file containing the input data.

```
execute <messagename> data1.txt
execute <messagename> data1.txt –p
execute <messagename> data1.txt –p -v
```

The options that can be specified while executing messages are

-parse   - Parse the input file (-p)
-validate   - Validates parsed object (-v)
-write   - Writes the object (-w)

If none of the options are specified, then the default is to parse, validate and write the input data.

Note:

The name of the message to be executed is case sensitive. The name should be given as defined in the cartridge.

See Also:

[Executing Message Flows](#)
[Executing Message Mappings](#)
[Performance measurement](#)
[Multi Thread testing](#)
[Executing multiple samples](#)

Executing Message Mappings

In some cases the user may want to execute the message mapping alone, to test it as a separate entity. Execute utility can be used to test only a message mapping.

When a message mapping is executed, the input data will be parsed, validated and mapped to the destination message. The destination message is then validated and serialized. The output dumped is the serialized destination message.

The argument to the message mapping should be a file containing the input data.

```
execute <messagemappingname> data1.txt
```

Note:

The name of the message mapping to be executed is case sensitive. The name should be given as defined in the cartridge.

See Also:

[Executing Message Flows](#)
[Executing Messages](#)
[Performance measurement](#)
[Multi Thread testing](#)
[Executing multiple samples](#)

Performance Measurement

By default the transformation is performed once when execute utility is run. The user would want to measure the transformation performance when multiple runs are executed. To measure the performance of a flow/message/mapping the '–perf' option should be used.

If no number is specified after the –perf option, by default the count will be 10000. The processing will be done once, and then the performance measurement will start. Only the output for the first run is dumped. The output generated during performance measurement is not dumped.

During performance testing, we need to measure performance after it has reached a stable state. So Hotspot compiler is warmed up before the actual performance testing is done.

```
    execute –perf:5000 <message/flow/mapping name> inputparam1….
```
The transformation is done 5000 times.

```
    execute –perf <message/flow/mapping name> inputparam1….
```
The transformation is done 10000 times.

The number of transformations per second and the time taken (in milli secs) to process each message will be dumped.

```
----- Performance Testing -------
Milli secs/message = 0.0875
Transformations/sec = 11428.57142857143
```

See Also:

[Executing Message Flows](#)
[Executing Messages](#)
[Executing Message Mappings](#)
[Multi Thread testing](#)
[Executing multiple samples](#)

Multi Thread Testing

The user may want to test the performance of flow/message/mapping in a multi-threading environment to check if the performance scales when multiple threads are executed.

Multithreaded testing can be done by using the '–threads' option. If the count is not specified after the '-threads' by default 5 threads will be executed. If a value is specified after the '-threads' option then that many number of threads will be executed.

By default for each thread the transformation will be performed 10000 times. If the '–perf' option is specified along with the –threads option, the transformation will be performed based on the -perf count.

```
   execute message/mapping/message flow –threads
```
5 threads will be executed, with transformation being done 10000 times for each thread.

```
execute message/mapping/message flow –threads:10
```
10 threads will be executed, with transformation being done 10000 times for each thread.

```
   execute message/mapping/message flow –threads:10 –perf:1000
```
10 threads will be executed, with transformation being done 1000 times for each thread.

The output dumped will contain the total time taken for processing, time taken to process a single message, transformations per thread per second and overall transformations per second.

```
----- Multi thread testing -------
Number of threads = 5
Number of iterations per thread = 10000
Total time taken = 2062ms
Milli secs/message = 0.2062
Overall Transformations/sec = 4849.66
Transformations/sec per thread= 969.93
```

See Also:

Executing Message Flows
Executing Messages
Executing Message Mappings
Performance measurement
Executing multiple samples

Executing Multiple Samples

During regression testing, you may want to test multiple samples in one shot. In such cases, to execute multiple samples, the parameters can be specified one after the as command line arguments.

```
   execute message/mapping/messageflow data1.txt data2.txt
```

Two files data1.txt and data2.txt will be executed.

Wildcard characters can also be given while specifying the filename. Files that match the pattern specified will be processed.

```
execute message/mapping/messageflow *.txt
```

All files with extension '.txt' will be executed. This support is present only in **execute** utility.

```
execute message/mapping/messageflow f*.txt
```

All files whose name start with 'f' and have extension '.txt' will be executed. This support is present only in execute utility.

When multiple files are executed, output for each file will be generated in the order they are specified. When each sample is executed a text 'Executing sample <count>' will be dumped.

```
…………..<output for first sample>
------- Executing sample 2 ----------------------
…………..<output for second sample>
```

Executing multiple samples for Flow with Multiple Arguments

Consider a flow that has two arguments a binary input and an integer value.
In this case the arguments to the flow should be a file containing the data and an integer value.

```
execute <flowname> data1.txt 1000
```

If multiple files need to be executed for this flow they should be specified as

```
execute <flowname> data1.txt 1000 data2.txt 2000
```

All the arguments for the first execution are specified and then the second input file has been specified.

See Also:

[Executing Message Flows](#)
[Executing Messages](#)
[Executing Message Mappings](#)
[Performance measurement](#)
[Multi Thread testing](#)

Working With Cartridge Designer

This section covers the features that make working with Designer more easier/interactive.

The following table describes the topics covered under this section.

Topic	Description
Tables	It explains how to manipulate tables in Designer.
Mount Directory	By allowing mounting of one or more file system directories, Designer supports management of external cartridge resources along with the cartridge.
Navigation Features	It explains how to quickly move between Designer elements.
Diff	It explains how to diff two design elements or two cartridges.
Cartridge Publisher	It explains how to generate HTML docs for the cartridge and its child design elements.
Cartridge Read-Only Mode	It explains how to view a cartridge in read only mode.
Verify Integrity	It explains how to verify the integrity of a message or messages in the cartridge.
Export/Import a Design Element	It explains how to export/import a design element in different formats such as XML, HTML, DTD.
Copy/Paste	It shows the easy way for creating copies of design elements and fields.
Validating Design Elements	It explains how to validate a design element or all design elements in a cartridge.
Delete Node	It explains how to remove a node from the cartridge.
Find Usage	It explains how the user can find the usages of a design element in a cartridge.
Find	It explains how the user can search for an item of interest in a design element based on the text contained in it.

[Incremental Search](#)	It explains how the user can quickly search for occurrences of text in text area elements, [tables](#) and [formula editor](#) that appear in [Designer](#) and [Simulator](#).
[Drag and Drop](#)	It shows the easy way for opening cartridge and data files.
[Recent File List](#)	It shows the easy way for accessing the recently opened cartridges.
[Exiting Designer](#)	It explains how to exit designer.
[Version Support](#)	It explains the version support provided by Designer.

See Also:

[Designer User Interface](#)
[Cartridge](#)
[Message](#)
[Message Mapping](#)
[Formula](#)
[Function Definition](#)
[Resources](#)
[Code Generation](#)
[Simulator](#)

Tables
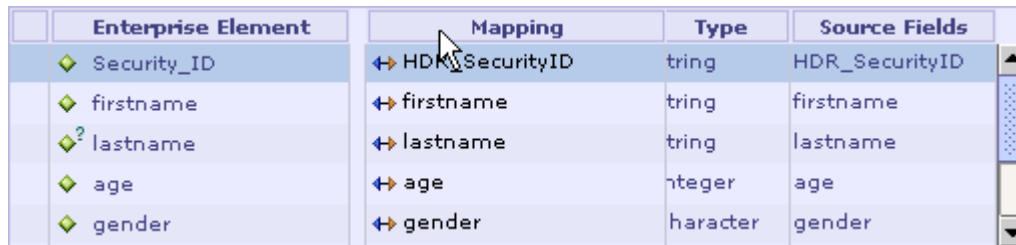
This section explains how to manipulate tables in Designer. The table contents of a message can also be viewed in tree structure.

See Also:

Rearranging Columns of a Table
Showing/Hiding Columns of a Table
Zebra Highlighting in Tables
Table Auto Formatting
Controlling Row Height in Tables
Tooltips for Table Elements
Viewing Table as HTML Page
Comment/Annotation Support in Tables
Expand/Collapse
Incremental Search

Rearranging Columns of a Table

The user can easily rearrange the columns of a table by dragging a column title and dropping it at the required location.

	Enterprise Element	Mapping	Type	Source Fields
	◆ Security_ID	↔ HDR_SecurityID	tring	HDR_SecurityID
	◆ firstname	↔ firstname	tring	firstname
	◆² lastname	↔ lastname	tring	lastname
	◆ age	↔ age	nteger	age
	◆ gender	↔ gender	haracter	gender

See Also:

Showing/Hiding Columns of a Table
Zebra Highlighting in Tables
Table Auto Formatting
Controlling Row Height in Tables
Tooltips for Table Elements
Viewing Table as HTML Page
Comment/Annotation Support in Tables
Incremental Search

Showing/Hiding Columns of a Table

Right-clicking a column title brings up a short-cut menu, which displays the actual columns of the table as menu items. Selecting the check box besides a menu item shows the corresponding column. Whereas deselecting the check box besides a menu item hides the corresponding column. As hiding a column is a common requirement, a menu item to hide the current column is also included in the short-cut menu.
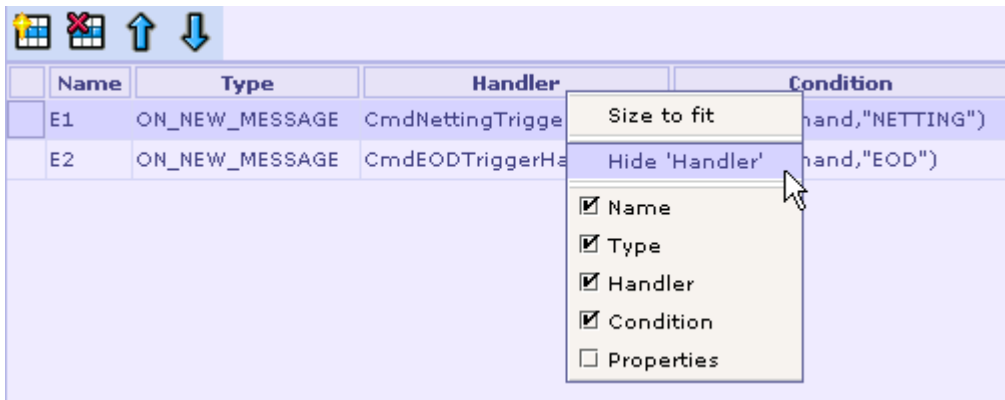
See Also:

Rearranging Columns of a Table
Zebra Highlighting in Tables
Table Auto Formatting
Controlling Row Height in Tables
Tooltips for Table Elements
Viewing Table as HTML Page
Comment/Annotation Support in Tables
Incremental Search

Zebra Highlighting in Tables

In the table, adjacent rows are colored differently to distinguish a row from its preceding and succeeding row as shown in the following picture.

See Also:

Table Auto Formatting

Right-click on any row in the table and select "Table Auto Format" from the context menu as shown in the following picture.
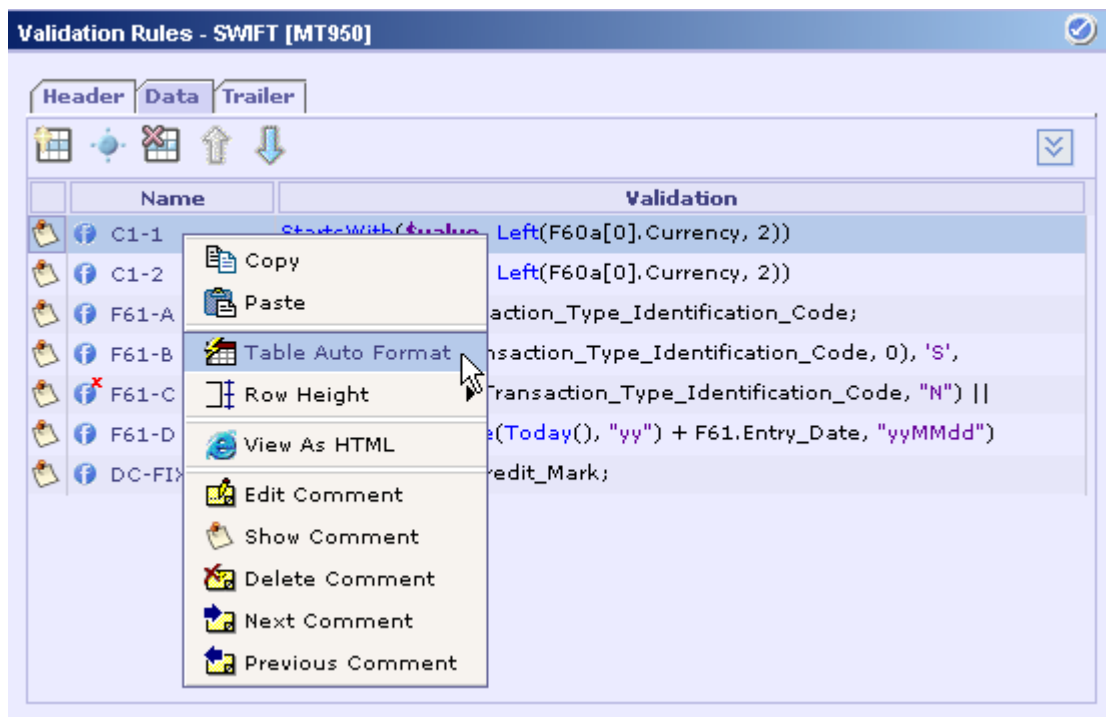
All the rows and columns in the table are resized (best fit) based on content as shown in the following picture.

See Also:

Controlling Row Height in Tables

Right-click on a row in the table and select the "Row Height" menu item from the context menu as shown in the following picture.
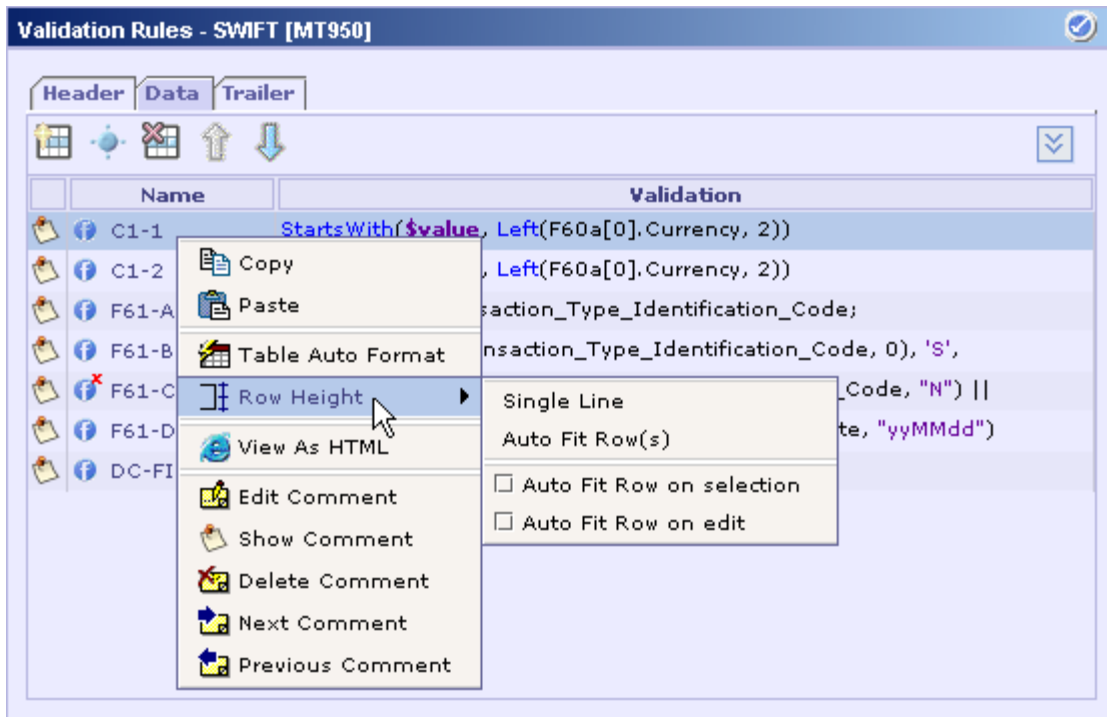
You can resize the selected row / all rows in the table using any of the four options (explained below) in the context menu, which appears when you select the "Row Height" menu item.

Single Line

If this option is selected, the height of the selected row/rows is set to a single line.

The following picture shows the rows before selecting the 'Single Line' menu item.
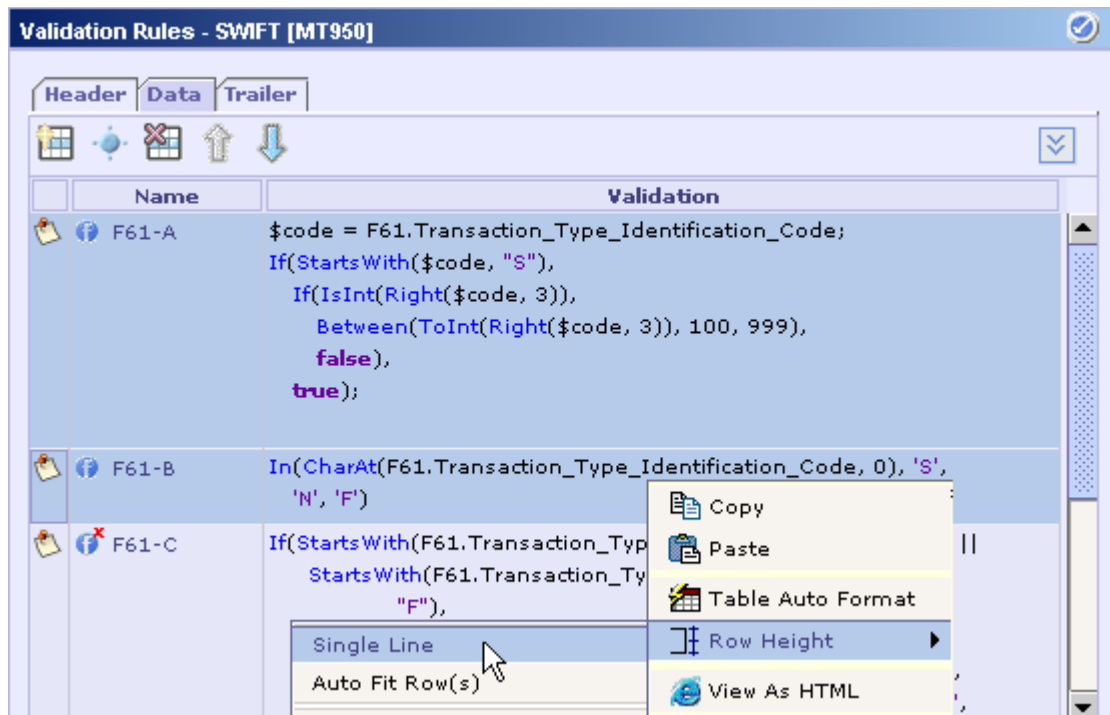
The following picture shows the rows after selecting the 'Single Line' menu item.
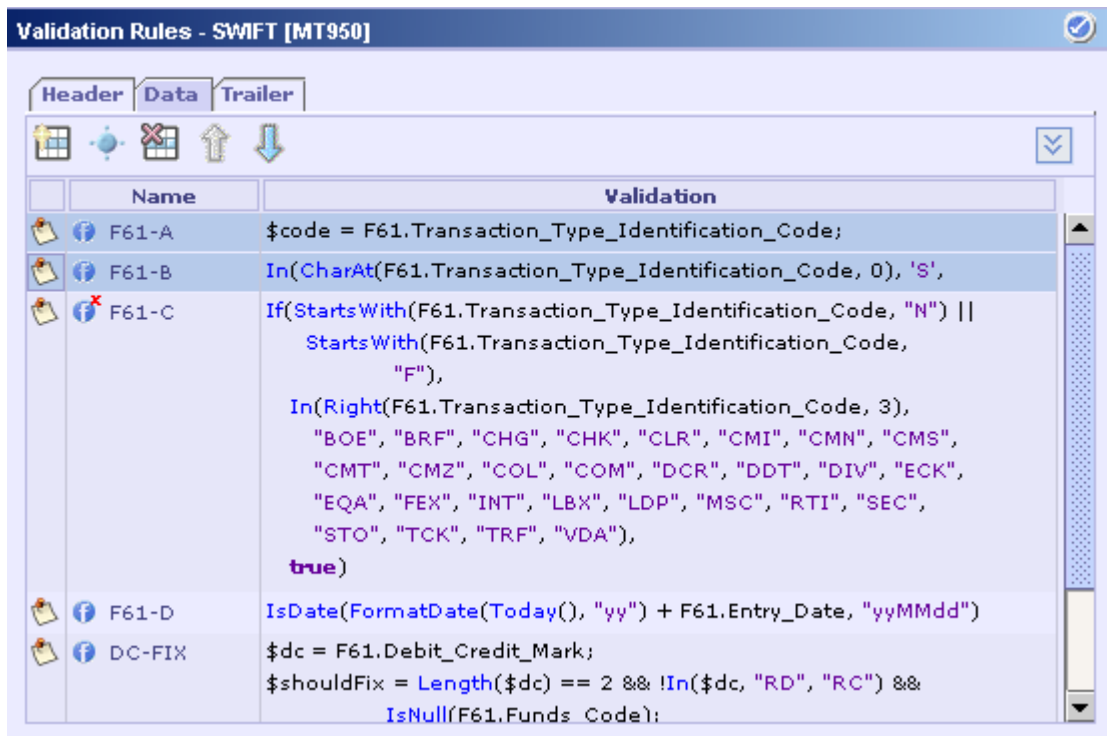
Auto Fit Row(s)

If this option is selected, the height of the selected row/s is set based on the height of the cell in that row which has the maximum height.

Auto Fit Row on selection

If this check box is checked, any row in the table is automatically resized on selection to appropriate height (multiple lines if required). When you move the selection to another row, the original height is restored. This is useful for viewing formula.

Auto Fit Row on edit

If this check box is checked, any row in the table is automatically resized to appropriate height when you start editing the same. Unlike "Auto Fit Row on selection", the original height is not restored when you navigate to some other row.

You can also set the required row height setting to all the rows and cells added in the table by right-clicking the Column Header and selecting "Row Height" from the context menu as shown in the following picture:

Auto Fit

If this option is selected, the height of all rows are automatically best fit based on the height of the cell in the corresponding row which has the maximum height. This is same as "Auto Fit Row(s)" seen above, but is applied to all rows instead of the selected row alone.

Name	Validation	Applie

ⓘ C16	If (F71F.$size > 0 \|\| F71G.$size > 0,
	    F33B.$size > 0,
	    true
	)	

ⓘ C17	If (In($value, "TELI", "PHOI"),
	    F56a.$size > 0,
	    true
	)
		23E.Instruction

ⓘ C18	If (In($value, "TELE", "PHON"),
	    F57a.$size > 0,
	    true
	)
		23E.Instruction

ⓘ C19	$value == F32A[0].Currency	71G.Currency

Auto Fit cells in column

If this option is selected for a column, say "Name", then the height of all the rows in the table is set based on the height of the corresponding cell, which appears under the column "Name".

Name	Validation
ⓘ C1-1	If ($value == F32A[0].Currency,
ⓘ C1-2	F33B.$size > 0
ⓘ C2	$mir = "";

See Also:

[Rearranging Columns of a Table](#)
[Showing/Hiding Columns of a Table](#)
[Zebra Highlighting in Tables](#)
[Table Auto Formatting](#)
[Tooltips for Table Elements](#)
[Viewing Table as HTML Page](#)
[Comment/Annotation Support in Tables](#)
[Incremental Search](#)

Expand/Collapse

Consider a table where the sections are collapsed

Enterprise Element	Type	Hidden	Description
◆ InvoiceDate	DateTime	☐	
◆ ClientID	String	☐	
🗀 item	Section	☐	

To expand sections right click the table and select 'All Sections' menu item under 'Expand' menu as shown below.

The table now appears as shown below

To collapse sections right click the table and select 'All Sections' menu item under 'Collapse' menu as shown below.

The table now appears as shown below

See Also:

[Rearranging Columns of a Table](#)
[Showing/Hiding Columns of a Table](#)

Tooltips for Table Elements

ToolTip for fields/sections added in the table show most of the properties set for the same as shown in the following picture.
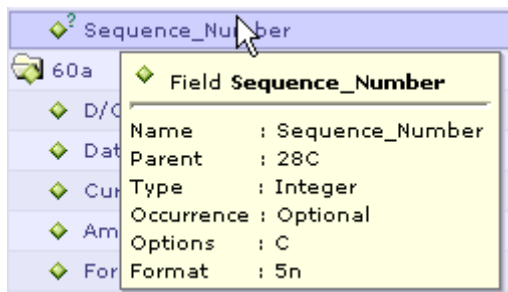
See Also:

Viewing Table as HTML Page

Selecting the **View As HTML** menu item from the short-cut menu that appears by right-clicking in the empty column of a table generates an HTML page containing the table information.

Enterprise Element	Data type	Mapping	Source Fields
◆ Security_ID	String	↔ HDR_SecurityID	HDR_SecurityID
△ firstname	String	↔ firstname	firstname
		↔ lastname	lastname
		↔ age	age
		↔ gender	gender
		↔ joindate	joindate
		↔ designation	designation
		↔ salary	salary

The HTML page is a sort of quick view of the current table information. The structure of the generated HTML page will be similar to the table. If you have hidden a column or reordered the columns, it will be reflected in the generated HTML page.

See Also:

Rearranging Columns of a Table
Showing/Hiding Columns of a Table
Zebra Highlighting in Tables
Table Auto Formatting
Controlling Row Height in Tables
Tooltips for Table Elements
Comment/Annotation Support in Tables
Incremental Search

Comment/Annotation Support in Tables

Mapping/validation formulas can easily be created, but very difficult to maintain/understand. Comments will make that easy.

Given below is the list of items for which the user can add comments. It also describes where it can be specified for the corresponding item.

Mapping – For mapping rules comments can be added against each target field in the left margin. For custom mapping, the comment can go into the window that pops up when the **Custom Mapping** button is clicked.

Validation – For both external and internal message validation rules, the comment can be specified in the left margin of the corresponding validation rule.

Enrichment – For processing of a field, a comment can be added in the left margin. For "Custom Processing", it is specified using the Comment dialog that pops up when clicking the **Add Comment** button.

See Also:
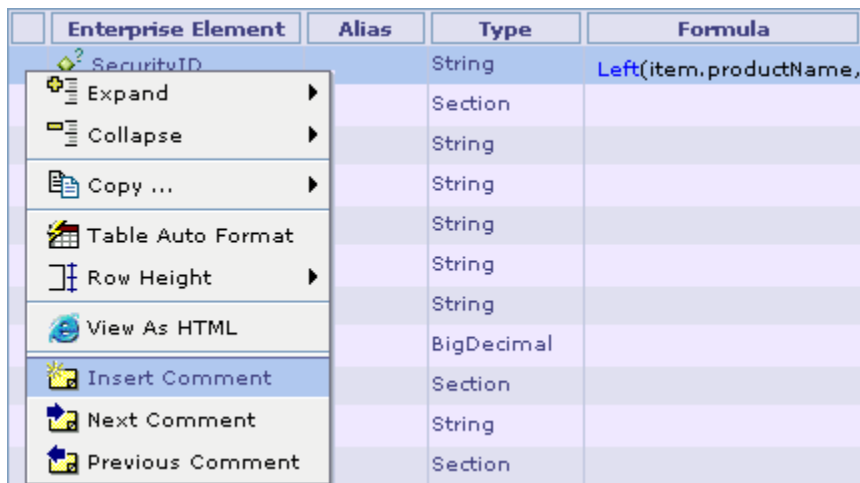
Adding a Comment

Follow the steps given below to add a comment for a table row.

1. In the table, right-click on the first column (empty column) of the row corresponding to the required field.

   This brings up the short-cut menu as shown in the following picture.

2. Now select the **Insert Comment** menu item from the short-cut menu.

   This brings up the comment/annotation pane as shown below.

3. Enter the comment in the annotation pane and click outside that window or just press the Esc button to close it to finish entering the comment.

See Also:

Editing a Comment
Removing a Comment
Viewing a Comment
Moving Between Comments in a Table

Editing a Comment

Follow the steps given below to edit an existing comment.

1. In the table, right-click on the first column (empty column) of the row corresponding to the field containing the comment to be edited.

    This brings up the short-cut menu as shown in the following picture.

2. Select the **Edit Comment** menu item from the short-cut menu.

This brings up the comment/annotation pane.

3. Modify the comment in the annotation pane and click outside that window or just press the Esc button to close it to finish editing the comment.

See Also:

Adding a Comment
Removing a Comment
Viewing a Comment
Moving Between Comments in a Table

Removing a Comment

Follow the steps given below to remove an existing comment.

1. In the table, right-click on the first column (empty column) of the row corresponding to the field containing the comment to be removed.

This brings up the short-cut menu as shown in the following picture.
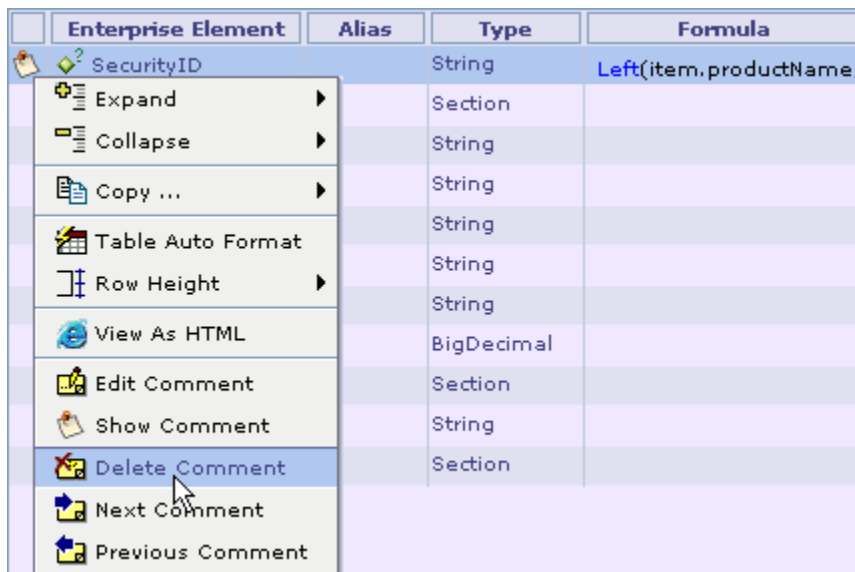
2. Select the **Delete Comment** menu item from the short-cut menu to remove the comment.

See Also:

Adding a Comment
Editing a Comment

Viewing a Comment

Moving the mouse cursor over the comment icon displayed in the empty column shows the comment. The comment disappears if you move away the mouse cursor. You can also display a comment by selecting the **Show Comment** menu item of the short-cut menu that appears when you right-click the empty column. But the comment pane should be closed by clicking outside the comment pane or by pressing the Esc button.

See Also:
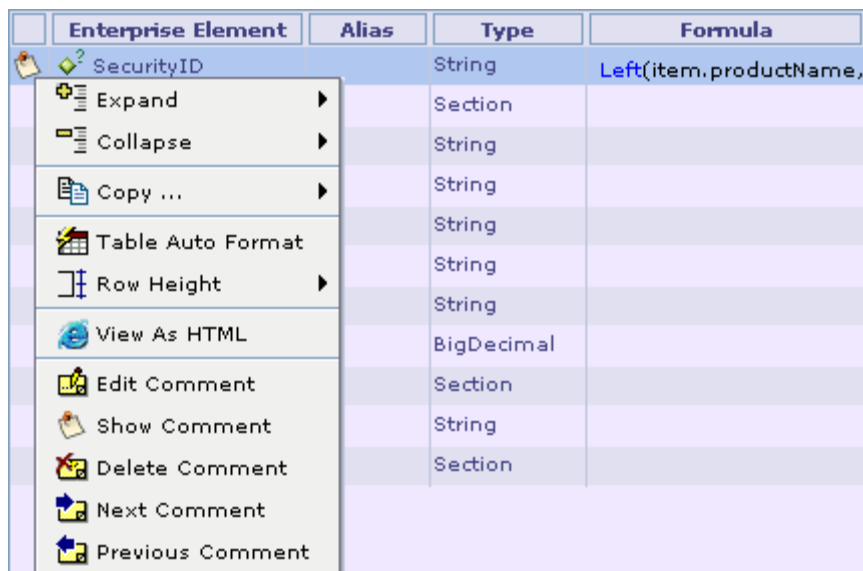
Adding a Comment
Editing a Comment
Removing a Comment
Moving Between Comments in a Table

Moving Between Comments in a Table

The user can easily move between the comments of a table by selecting the **Next Comment** and **Previous Comment** menu items of the short-cut menu that is displayed by right-clicking the empty column of a table.

	Enterprise Element	Alias	Type	Formula
	SecurityID		String	Left(item.productName,
	Expand ▸		Section	
	Collapse ▸		String	
	Copy … ▸		String	
			String	
	Table Auto Format		String	
	Row Height ▸		String	
	View As HTML		BigDecimal	
	Edit Comment		Section	
	Show Comment		String	
	Delete Comment		Section	
	Next Comment			
	Previous Comment			

See Also:

Adding a Comment
Editing a Comment
Removing a Comment

Mount Directory

Cartridge uses external resources such as source files, XML, DTD, etc. By allowing mounting of one or more file system directories, Designer supports management of these resources along with the cartridge. Designer also supports viewing, editing and creation of these resources

Usually you mount directories containing resources related to the current cartridge so that you can manage a cartridge and its resources from within Designer. It is recommended that the mount directories are present under the cartridge directory.

When you save a cartridge, the mounted directory locations are saved in it. When you reopen a cartridge, the mounted directories are shown in Explorer along with cartridge design elements.
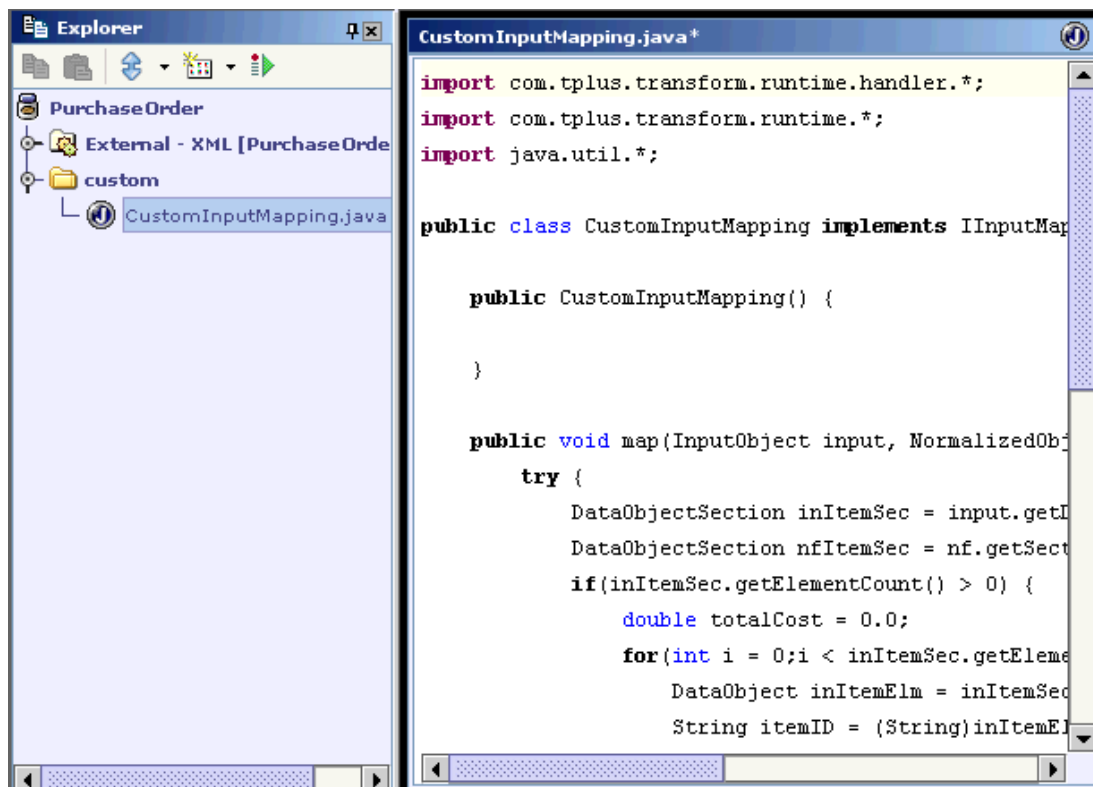
NOTE:

Saving a cartridge is not tied with saving the constituent files/directories of mounted directories. In fact changes made to a constituent file of a mounted directory is saved as soon as the focus is changed to some other item of cartridge.

Please also note that the source/resource files of a mounted directory are not included in the code generation process. For these files to be included in the code generation process, they have to be specified in the code generation settings of the Cartridge.

See Also:

[Mounting a Directory](#)
[Working with a Mounted Directory](#)
[Editing and Saving a File](#)
[Creating a Directory](#)
[Creating a File](#)
[Working With Cartridge Designer](#)
[New File from Template](#)

Mounting a Directory

Follow the steps given below to mount a directory:

1. Select 'Mount Directory' menu item from the 'New Item' drop down in Explorer toolbar.
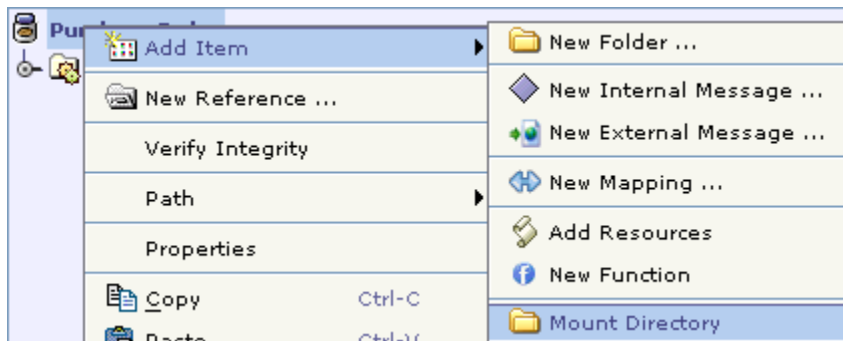
The 'Select Directory' file open dialog box is shown.

2. Select a directory and click on the Open button.

The selected directory along with its top-level children is shown in Explorer.

Save the cartridge, if you want the mounted directory location is saved with the cartridge so that when you reopen cartridge, the mounted directory is also shown in Explorer along with cartridge design elements.

See Also:
[Mount Directory](#)

Working with a Mounted Directory

When you mount a directory, the constituent files and sub-directories are displayed in Explorer in a tree view. Please note that not all the files under the mounted directory are shown; only files with some extension (file types commonly used with a Cartridge) are shown.

When you select a mounted directory or its sub-directory in Explorer, the top-level children of that directory are displayed in the detail pane. When you hover the mouse pointer over the name of a child item, it displays a hyperlink and clicking it will take you to that child item.

You can expand/collapse the mounted directory and its sub-directory nodes. In fact when you collapse and then expand a directory node, the content of that directory is refreshed. This means that if a file/directory is newly created under that directory (using another application), it will be included in the tree view.

See Also:

Mount Directory

Editing and Saving a File

When you select a file node in Explorer, the content of that file is shown in detail pane. The content is shown with syntax highlight for some file types such as JAVA, CPP, XML, etc. The status bar at the bottom of the detail pane shows the current line and column.

To edit a file, simply select it in Explorer and change its content shown in detail page. As soon as you change the focus to some other node in Explorer, the file gets saved.

Please note that saving the file is not tied with saving a cartridge. The file is saved when the file node loses focus irrespective of whether the cartridge is saved or not.

See Also:

Mount Directory

Creating a Directory

You can create a new directory under the mounted directory or one of its constituent directories by following the steps given below. The new directory location is relative to the selected directory.

1. Select the directory node in Explorer under which you want to create a new directory.

2. Select the 'New Folder' menu item from its shortcut menu.
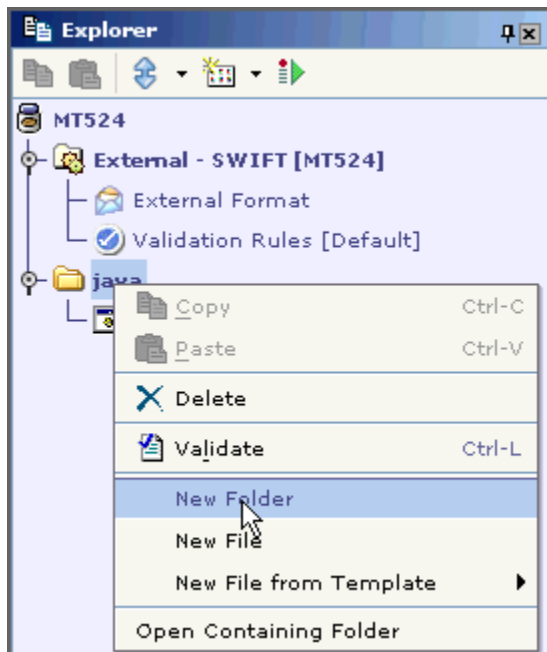
The 'New Folder' message box is shown.

3. Enter the name of the folder to be created and click OK.

The folder with the specified name is created directly under the selected directory.

See Also:

Mount Directory

Creating a File

You can create a new file under the mounted directory or one of its constituent directories by following the steps given below. The new file location is relative to the selected directory.

1. Select the directory node in Explorer under which you want to create a new file.

2. Select the 'New File' menu item from its shortcut menu.

   The 'New File' message box is shown.

3. Enter the name of the file to be created with its extension and click OK.

The file with the specified name is created directly under the selected directory.

See Also:

[Mount Directory](#)
[New File from Template](#)

New File from Template

One of the things users occasionally would want is how to write client, custom extension classes etc. The 'New File From Template' feature allows the user to create source files from template. Source files can typically be created for handler classes, clients etc from predefined templates. The files can be created for Java. This feature is available only when a directory has been mounted in the cartridge.

The templates are stored under 'config\filetemplates' directory. The templates for Java are under 'config\filetemplates\Java' and. They are customizable. The templates are shown as menus when the user selects a 'Mount Directory' folder and selects 'New File from Template' menu.

See Also:

[Creating a New File from Template](#)
[Mount Directory](#)
[External Sources Tab (Java/EJB)](#)

Creating a New File from Template

1. Right click the 'Mount directory' folder under which you need to create a file and select 'New File from Template' menu item. Select Java menu item depending on the platform in which the template is to be generated.
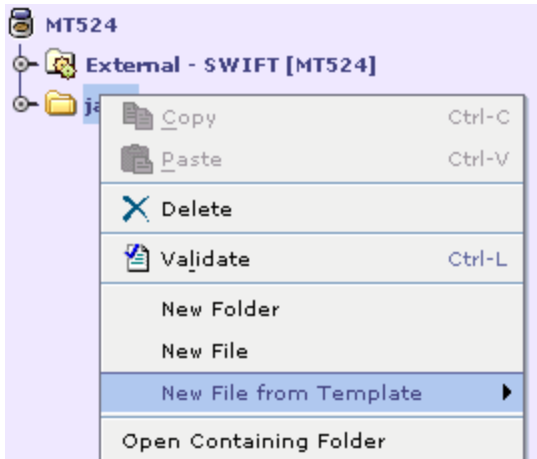
The types of template that are supported will be displayed. Select the template for which you need to create a template file. Based on the selected template a dialog will appear which will prompt you to specify parameters for the template file to be generated. This dialog will vary depending on the template that is selected.

2. If 'FieldValidation' menu item is selected the following dialog will be displayed.
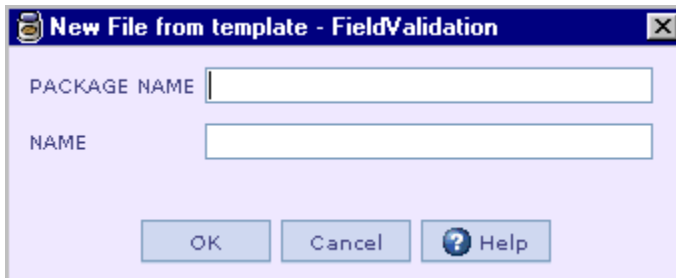
3. Specify the package in which the client is to be generated.

4. Specify the name of the client.

A client file 'PurchaseClient.java' will be created under the mount directory.

Navigation Features

These features help in quick movement between elements of a cartridge which otherwise require the Explorer window.

Moving Between Recently Visited Elements

The **Back** menu item of the **Edit** menu takes to the last design element the user has visited before selecting the current design element.

Whereas, the **Next** menu item of the **Edit** menu takes to the design element that the user has visited before selecting the **Back** action.

Moving from a Field to its Validation Rule

The user can quickly move from a field definition in the Format UI to its validation rules. This can be done by selecting the **Find > Validation** menu item from the short-cut menu that appears when you right-click in the row corresponding to the required field.
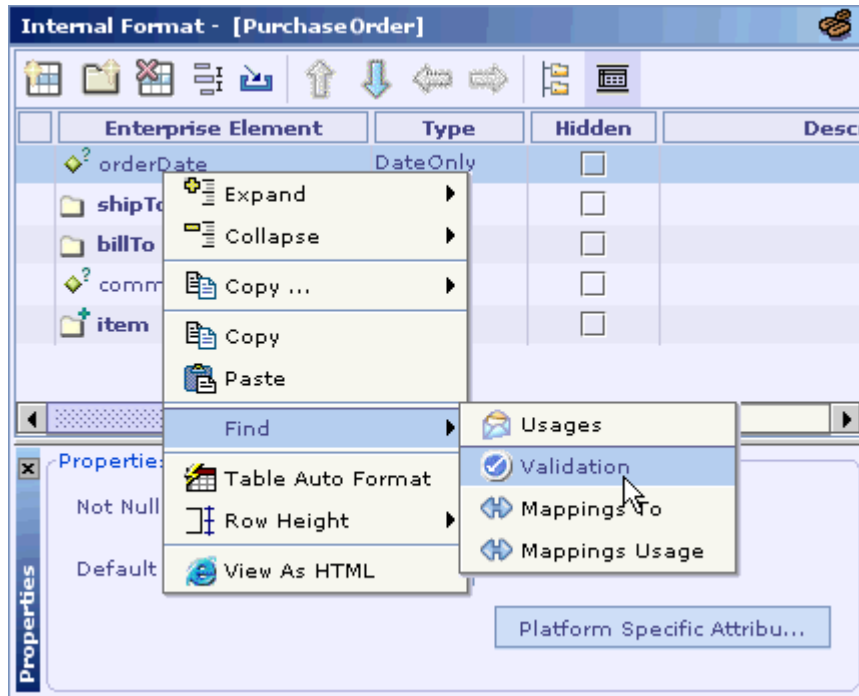
See Also:

Moving from a Field to its Mapping Rule

The user can quickly move from a field definition in the Format UI to its mapping rule, i.e. to the mapping rule in which this field is the target field. This can be done by selecting the **Find > Mappings To** menu item from the short-cut menu that appears when you right-click in the row corresponding to the required field.
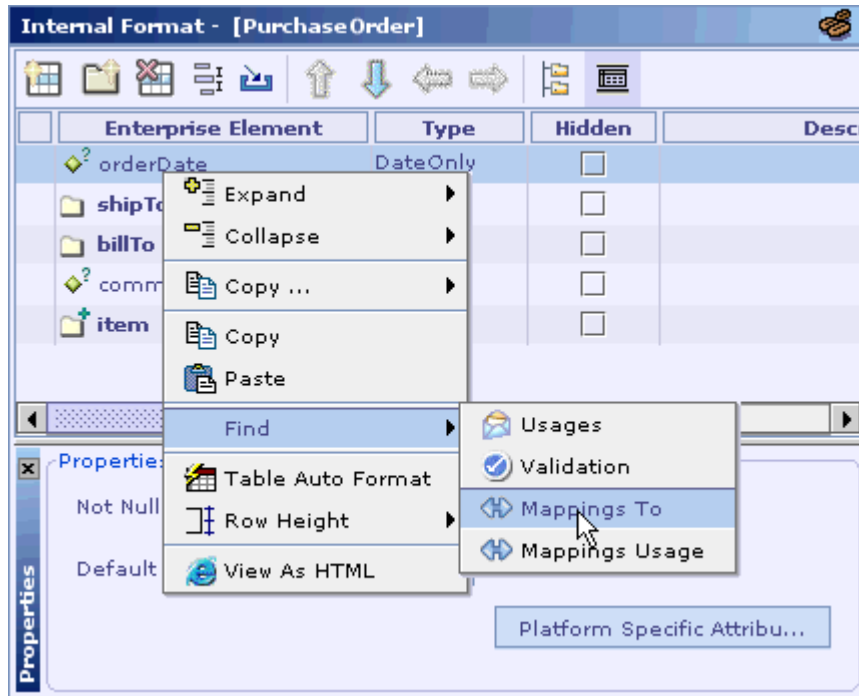
See Also:

Moving from a Field to its Mapping Usage

The user can quickly move from a field definition in the Format UI to its mapping usage, i.e. to the mapping rule in which this field is used as the source field. This can be done by selecting the **Find > Mappings Usage** menu item from the short-cut menu that appears when you right-click in the row corresponding to the required field.

See Also:

Moving from a Field to its Usage Items

The user can quickly move from a field definition in the Format UI to its usage items such as validation rules, mapping rules and processing rules. This can be done in two steps. First select the **Find > Usages** menu item from the short-cut menu that appears when you right-click in the row corresponding to the required field.
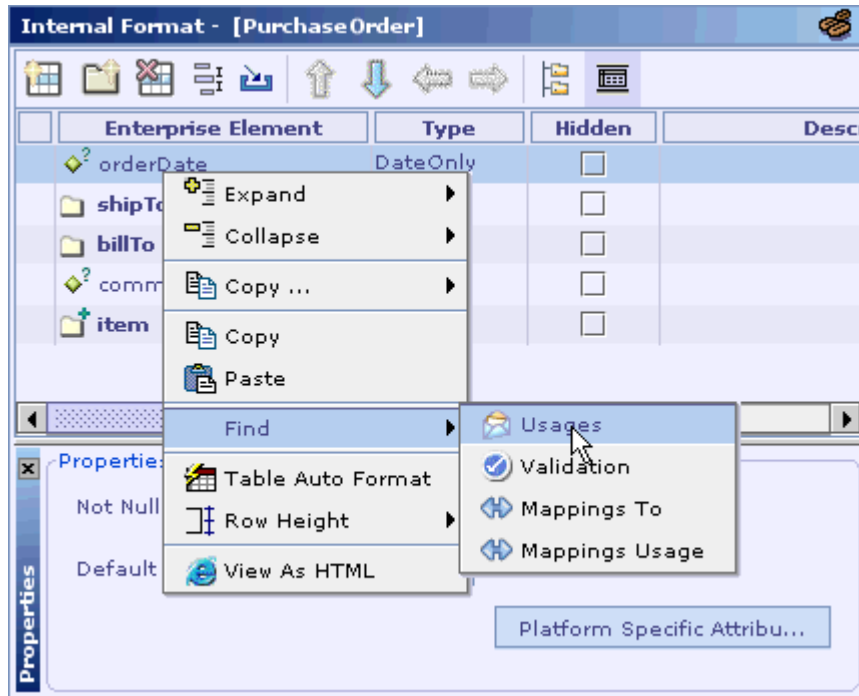
Next move to the required item by clicking the hyperlink displayed when you hover the mouse pointer over the items shown in the Search Results pane.

See Also:

Moving Between Recently Visited Elements
Moving from a Field to its Validation Rule
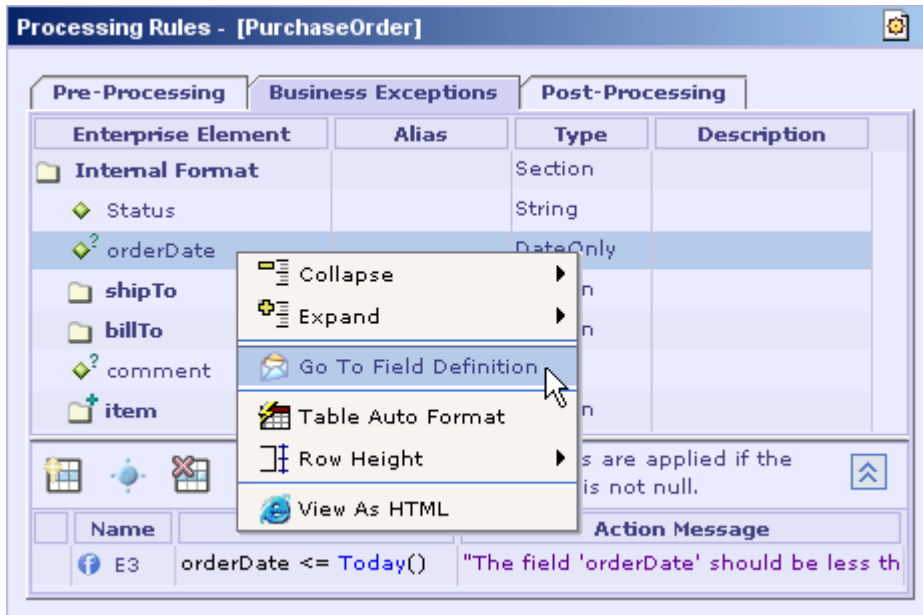Moving from a Field to its Mapping Rule
Moving from a Field to its Mapping Usage
Moving Back to a Field Definition
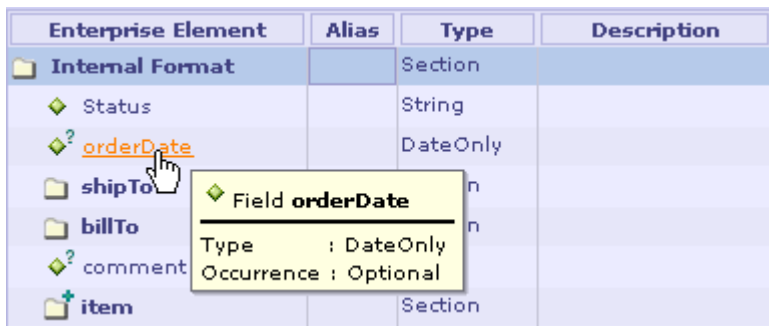Moving Between Source and Destination Fields in Mapping Rules UI
Navigation Features

Moving Back to a Field Definition

The user can quickly move back to a field definition in the Format UI from its usage items such as validation rules, mapping rules and processing rules. This can be done by selecting the **Go To Field Definition** menu item from the short-cut menu that appears when you right-click in the row corresponding to the required field.

The same can also done by moving the mouse pointer over the required field when the CTRL key is depressed and clicking the hyperlink displayed as shown in the following picture.
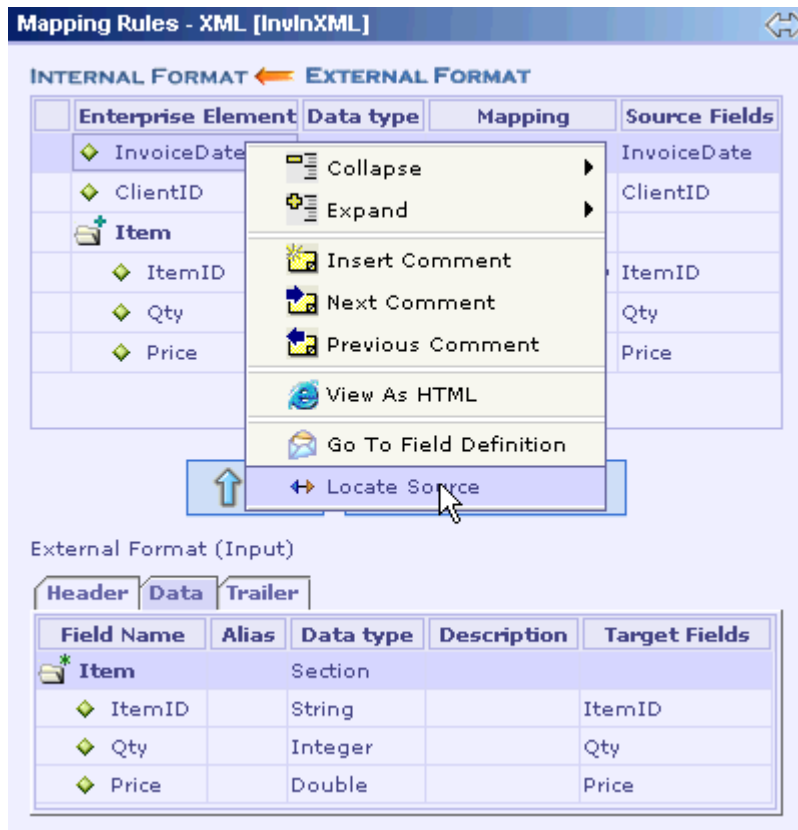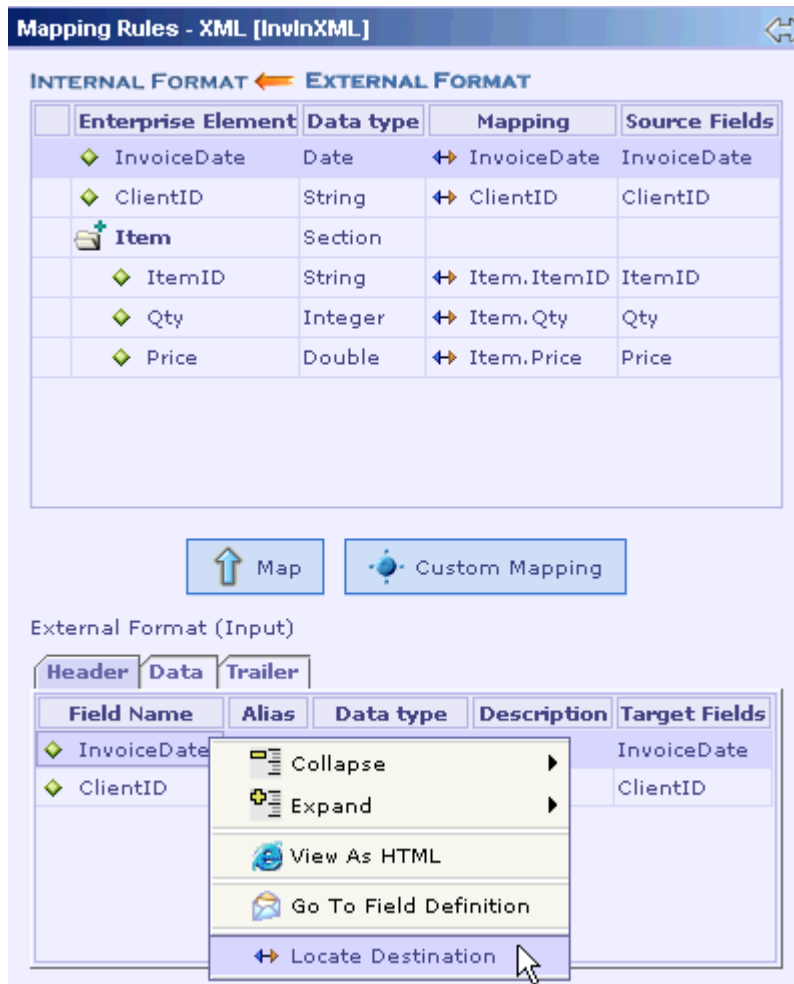

See Also:

Moving Between Recently Visited Elements
Moving from a Field to its Validation Rule
Moving from a Field to its Mapping Rule
Moving from a Field to its Mapping Usage
Moving from a Field to its Usage Items
Moving Between Source and Destination Fields in Mapping Rules UI
Navigation Features

Moving Between Source and Destination Fields in Mapping Rules UI

To move to the source field corresponding to a target field, select the **Locate Source** menu item from the short-cut menu that appears when you right-click on the target field row as shown in the picture given below.

Likewise to move to the target field corresponding to a source field, select the **Locate Destination** menu item from the short-cut menu that appears when you right-click on the source field row as shown in the picture given below.

See Also:

Diff

This section explains how to diff two design elements or two cartridges. Designer supports the following diff operations.

[Comparing Two Nodes in the Cartridge](#)
[Comparing Two Cartridges](#)

See Also:

[Differencing View](#)
[Differences Pane](#)
[Exporting as HTML](#)
[Cartridge Publisher](#)
[Cartridge Read-Only Mode](#)
[Verify Integrity](#)

Comparing Two Nodes in the Cartridge

Designer provides support for comparing two nodes in the cartridge. The only restriction is that the two nodes should be of the same type. For instance it does not make sense to compare FIX message's field structure with that of SWIFT.

Select two nodes in the explorer (exactly two).

Right click and select 'Diff' from the context menu.

The first selected node and its children are compared with the second and the Diff view is displayed as a separate frame.

See Also:

[Comparing Two Cartridges](#)
[Differencing View](#)
[Differences Pane](#)
[Exporting as HTML](#)
[Diff](#)

Comparing Two Cartridges

Designer also provides support for comparing two cartridges. The active (currently opened) cartridge can be one of the two cartridges that are compared.

From the Tools menu select 'Diff Cartridges' menu item.

Select the first and the second cartridge files. You can also choose to use the active cartridge (the one that is open or selected in the explorer) as either the first or the second cartridge.

Click OK. The two cartridges are compared and the diff view is displayed in a separate frame
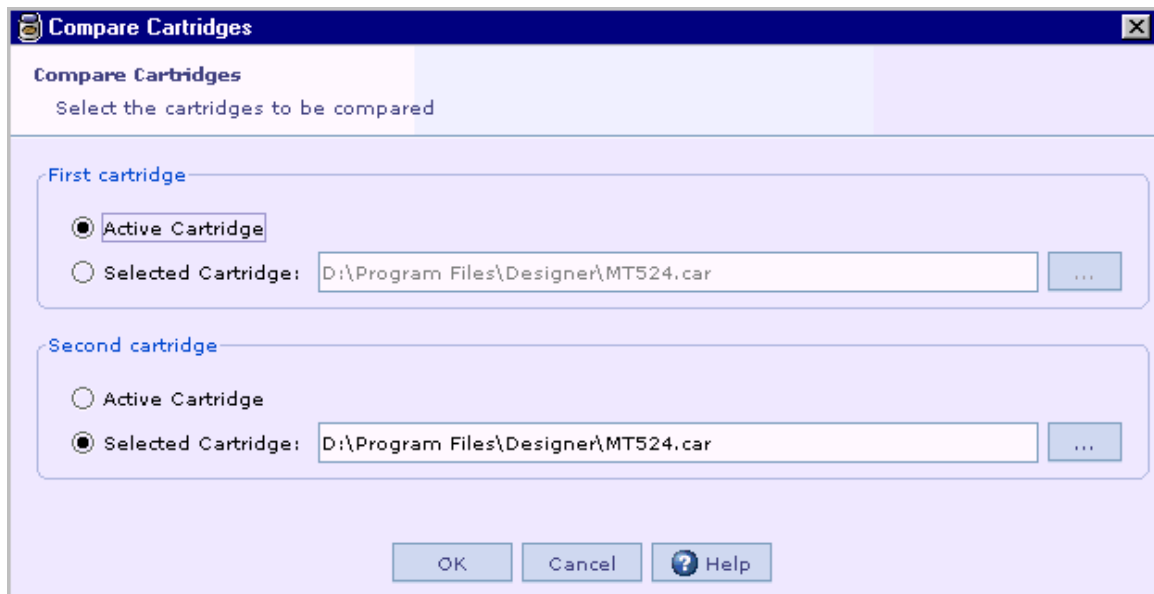
An interesting use of Differencing is in upgrading messages in the cartridge. For example, if a cartridge uses FIX4.1 message and the user wants to upgrade it to the next version (4.2), this process can be made easier by comparing them and pointing out the differences (field deleted etc). This will be particularly useful in case of internal message upgrades (when EIM changes).

See Also:

Comparing Two Nodes in the Cartridge
Differencing View
Differences Pane
Exporting as HTML
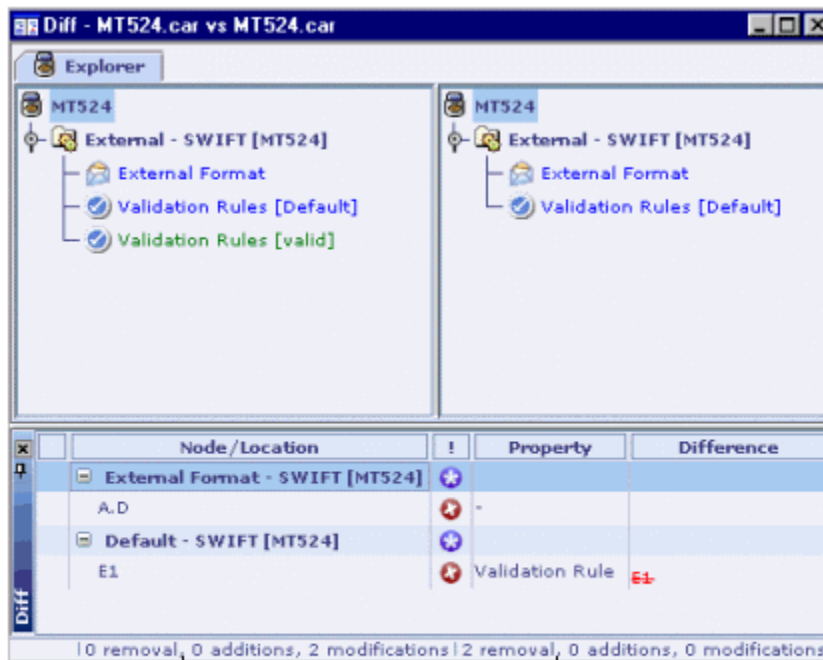Diff

Differencing View

The diff view is a tri-pane with the bottom pane displaying the list of differences.

The top left pane represents the first node/cartridge and the top right pane represents the second.

Initially the Explorer View of the first and second entities is displayed. Double clicking a node opens a new tab, with the detail view of the selected node displayed (next to each other). Number of such detail views can be opened in tabs.

To close a detail view tab, right click on the tab and select close (or Close All). This does not work for explorer tab (for obvious reasons).

Clicking on a node in one of the explorer view will result in synchronizing the other explorer view (the corresponding node is selected). This synchronization also works for fields in the Detail views.

See Also:

Comparing Two Nodes in the Cartridge
Comparing Two Cartridges

Differences Pane

It is the bottom pane that displays the list of differences.

When you are viewing the explorer, the differences are grouped based on the node as shown as tree. Use the context menu to expand / collapse the tree.

Node/Location	!	Property	Difference
⊟ **External Format - SWIFT [MT524]**	✱		
A.D	✱	-	
⊟ **Default - SWIFT [MT524]**	✱		
E1	✱	Validation Rule	E1

| 0 removal, 0 additions, 2 modifications | 2 removal, 0 additions, 0 modifications

Clicking on a node/location (hyper link) will take you to that location. If required it may open a 'Detail View comparison' in a new tab.

The property column indicates the property that has changed. If the node or the field itself has been deleted or added this column will be empty.

See Also:

[Comparing Two Nodes in the Cartridge](#)
[Comparing Two Cartridges](#)
[Differencing View](#)
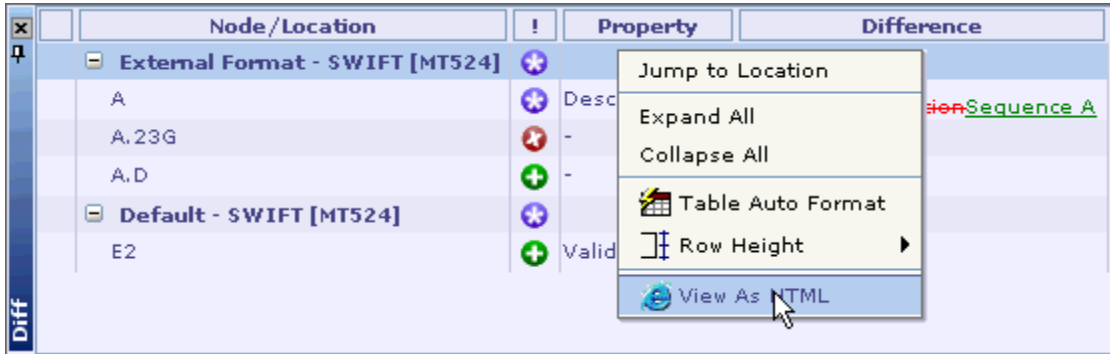[Exporting as HTML](#)
[Diff](#)

Exporting as HTML

The 'Export As HTML' feature of the Diff List Table (bottom pane) can be used to generate HTML reports.

Follow the steps given below to generate HTML report:

Expand all the nodes in the Diff list by selecting 'Expand All' from the context menu.

Right click and select 'View As HTML'.

The generated HTML report appears as follows.

Colors and Icon

Color	Icon	Explanation
Red		Node or field deleted from the first
Green		Node or field added to the second
Blue		Property of a node or field has been modified

Tool Tip

Taking the mouse over the 'Diff Icon' at a location will popup a tool tip that shows the differences at the location. This tool tip is also displayed in 'Difference' column of the Diff List (bottom pane).
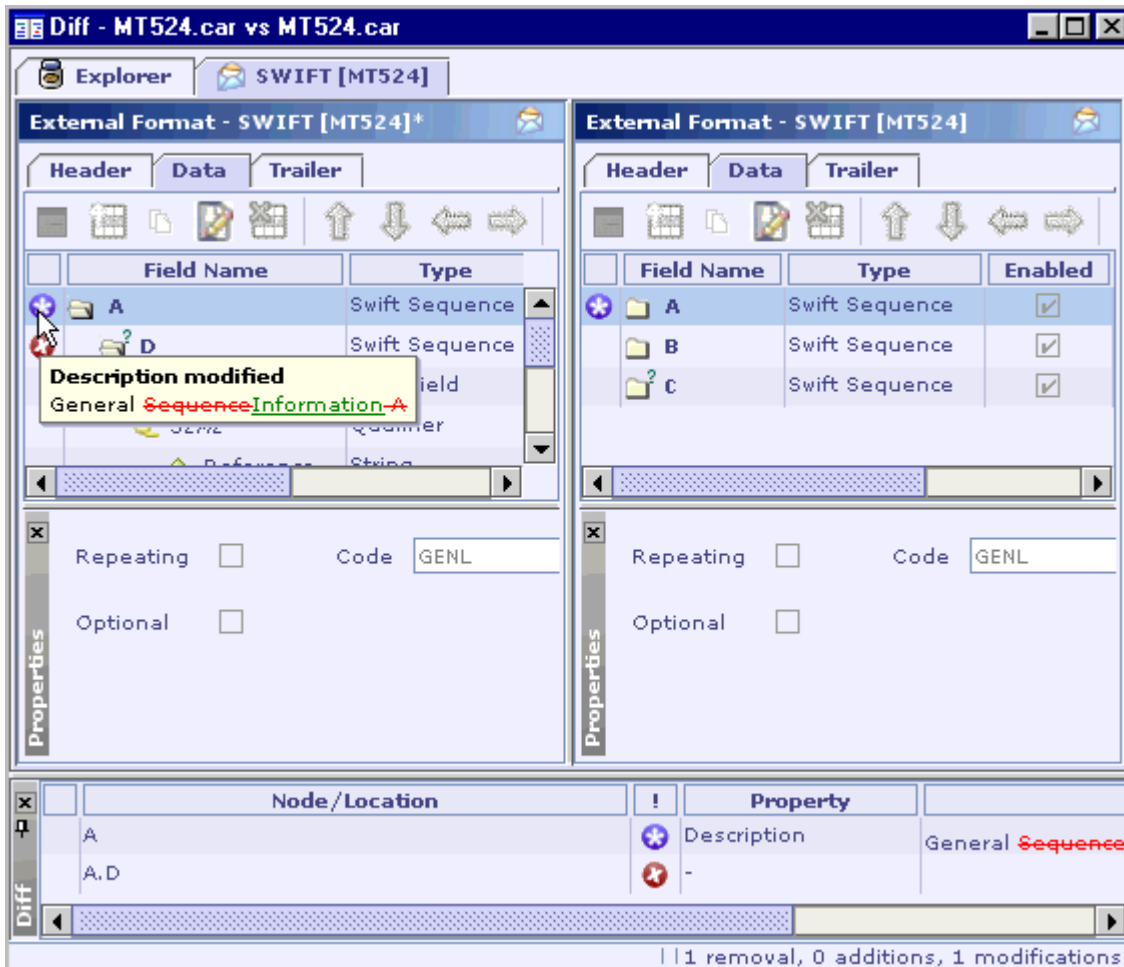
We have used Microsoft Word's 'Compare Documents' and 'Track changes' convention to display the differences. The old value is struck out (in red) and the new value is underlined (in green).

See Also:

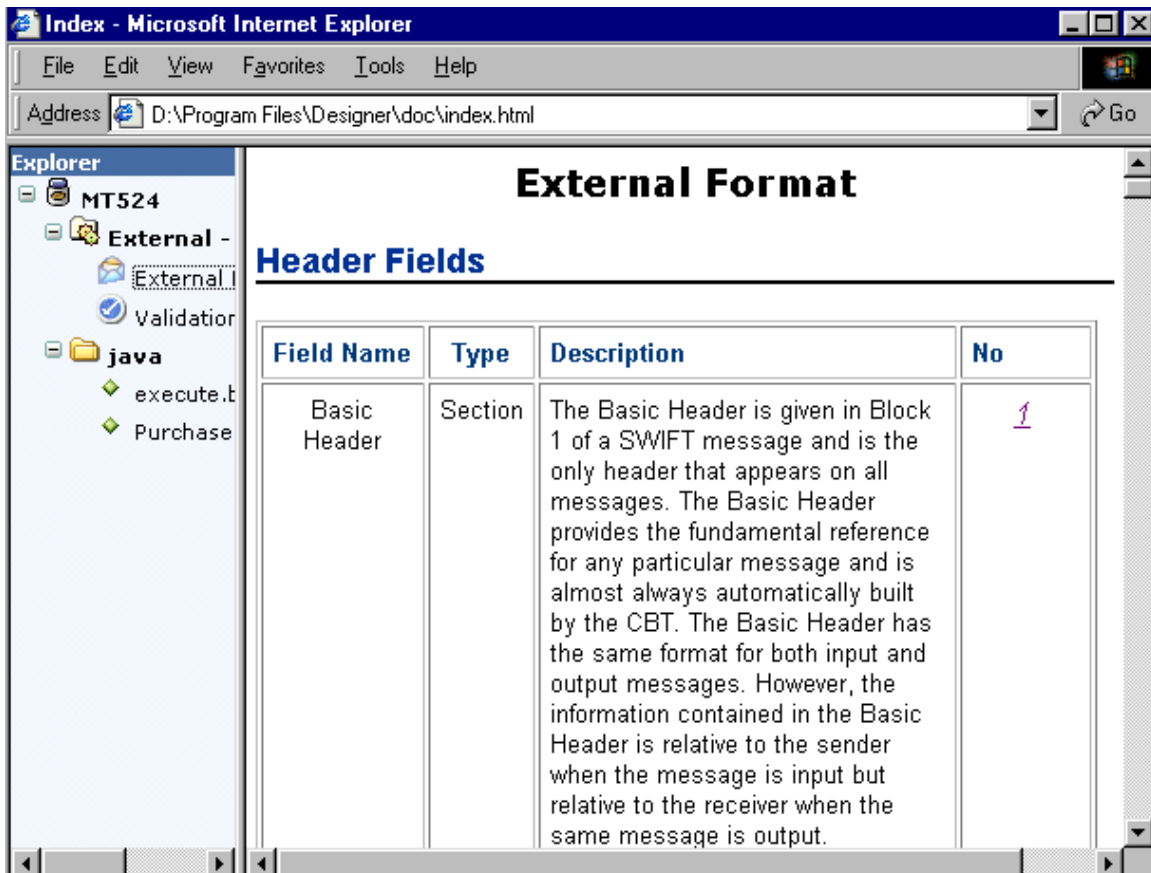Comparing Two Nodes in the Cartridge
Comparing Two Cartridges
Differencing View
Differences Pane
Diff

Cartridge Publisher

The 'Cartridge Publisher' tool is used to generate HTML docs for the cartridge and its child design elements into the specified directory.

The docs are generated as multiple files with an index.html and an explorer as shown below.

The explorer displays the contents of a cartridge in the form of a tree structure just like the Explorer window in Designer. The ⊞ icon represents an expandable node with child items and you can click on it to expand the node. Likewise, the ⊟ icon represents an already expanded node and you can click on it to collapse the node. Clicking on the hyperlink of a node loads the corresponding details page on the left window.

In case of design elements with simple properties, the details page displays those properties as shown below.

External Message : Order	
Name :	Order
Description :	This message is sent by a client for ordering items.
Standard :	XML
Version :	1.0
Standard Name :	Order
Standard Detailed Name	Purchase Order

In case of design elements for which a list of items needs to be displayed, the details page displays a summary table listing all the items involved and separate details table for each of those items. Each item displayed in the summary table is provided with a link and the user can click on it to view the details table of the item. It should be noted that the item properties displayed in these tables are configurable.

A sample summary table of an 'Internal Format' details page is shown below. The fields and sections of the internal format are listed in this table. The user can click on the hyperlink provided in the right-most column or the 'Name' column to move to the details page of the corresponding field. The user can also click on the folder icon displayed next to the section name to expand/collapse it.

	Name	Type	Description
1	◆ orderDate	DateOnly	
2	📁 item	Section	
3	◆ partNum	String	
4	◆ quantity	BigInteger	

A sample details table of an 'Internal Format' details page is shown below. It is the details table corresponding to the 'quantity' field. It lists the field properties that have been chosen for the HTML report. It also contains links to parent section of the field and the validation rule applied for that field. The user can click on these links to view the details table of those items. In general hyperlinks are provided wherever it makes sense.

See Also:

Generating HTML Reports
Cartridge Publisher Settings
Working With Cartridge Designer

Generating HTML Reports

Follow the steps given below to generate HTML docs for the current cartridge:

1. Change the 'Cartridge Publisher' settings, if required.

2. From the Tool's menu, select 'Cartridge Publisher' menu item.

3. The 'Cartridge Publisher' dialog will be displayed.
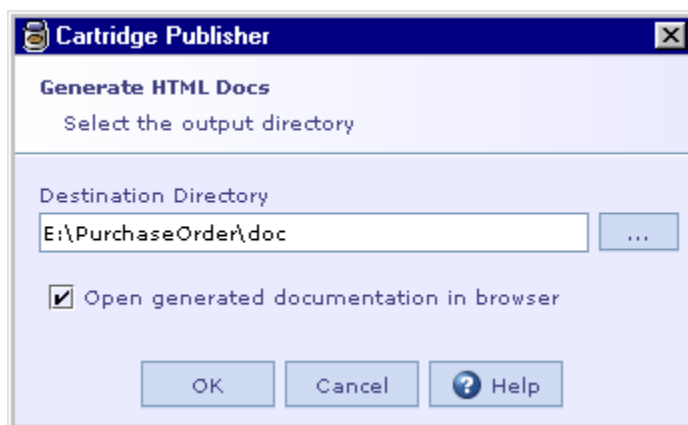
4. In the 'Destination Directory' text box, enter the directory in which the docs (HTML files) should be generated.

5. Alternatively you can click on the ellipsis button next to the text box to bring up the 'File Open' dialog and select the directory.

6. Select the 'Open generated documentation in browser' check box, to immediately open HTML docs after generation.

7. Click OK to start generation of HTML docs.

See Also:

Cartridge Publisher Settings

Cartridge Publisher Settings

The 'Configure HTML Reports' dialog displayed when selecting the Tools > 'Cartridge Publisher Settings' menu item allows the user to configure HTML docs generation. In particular, it allows the user to select the design element properties to be included in the generated HTML docs. The 'Configure HTML Reports' dialog is shown below.
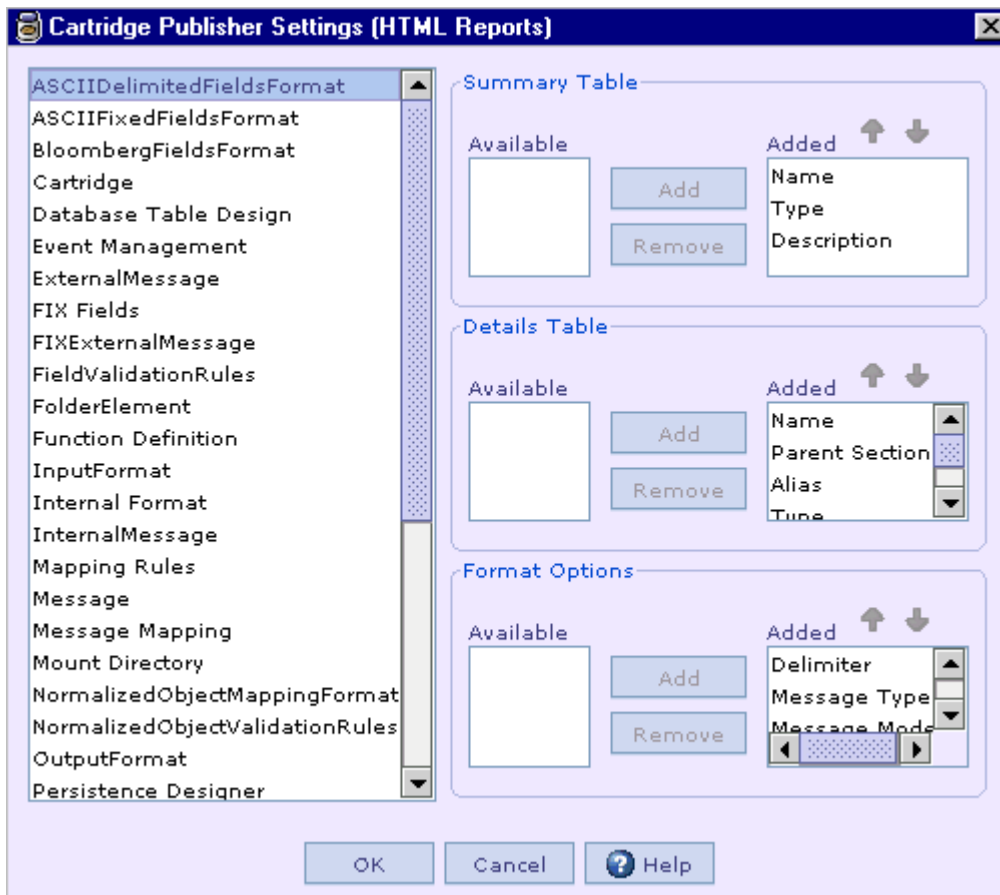
The list box on the right side of the dialog box lists the different types of design element items that can be configured. On selecting a design element item from this list, the current configuration for the selected design element appears on the left side of the dialog box.

In case of design elements with simple properties, the corresponding page displays those properties in the details table of that design element. Examples of this type of design elements include Cartridge, Internal/External Message, and so on. Configuration of these design elements involves properties to be included for the details table as shown in the following picture.
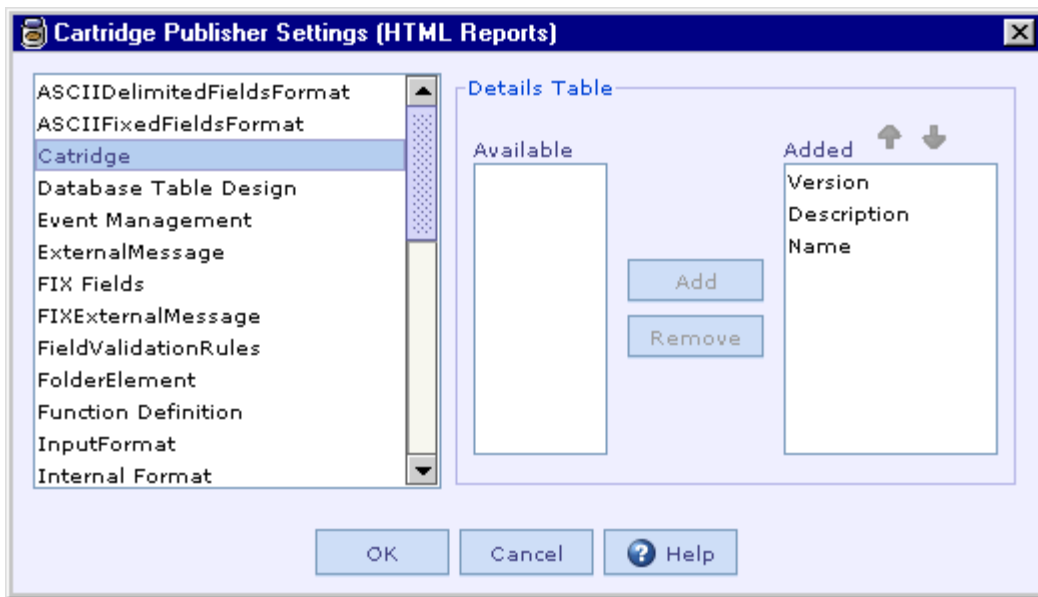
In case of design elements for which a list of items needs to be displayed, the corresponding page displays a summary table listing all the items involved and separate details table for each of those items. Examples of this type of design elements include 'Internal Format' of internal messages, 'External Format' of external messages of all type, and so on. Configuration of these design elements involves properties to be included for the summary table and details table as shown in the following picture.
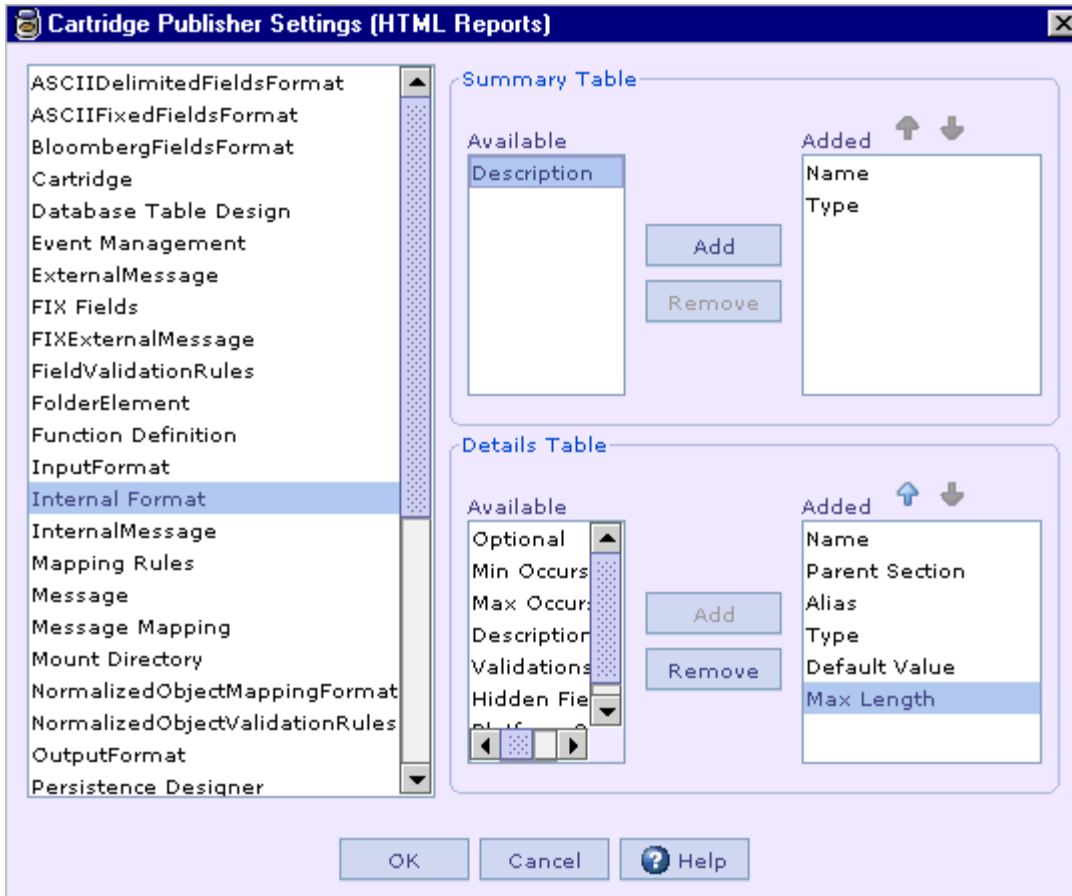
Adding Properties to the Generated Report

The user can specify a design element property to appear in the generated report by following the steps given below.

1. In the 'Available' list of the summary/details table, select the properties that should appear in the generated report.

    Use SHIFT-click to select a set of continuous properties and CTRL-click to select any non-continuous properties without affecting the current selection.

2. Click on the 'Add' button.

    The selected properties will be moved from the 'Available' list of the summary/details table to the 'Added' list and these properties will be included in the generated report.

Removing Properties from Generated Report

The user can specify a design element property not to appear in the generated report by following the steps given below.

1. In the 'Added' list of the summary/details table, select the properties that should not appear in the generated report.

   Use SHIFT-click to select a set of continuous properties and CTRL-click to select any non-continuous properties without affecting the current selection.

2. Click on the 'Remove' button.

   The selected properties will be moved from the 'Added' list of the summary/details table to the 'Available' list and they will be excluded in the generated report.

Arranging Properties of the Generated Report

The user can specify the position of the design element properties in the generated report by following the steps given below.

1. In the 'Added' list of the summary/details table, select the properties to be repositioned.

   Use SHIFT-click to select a set of continuous properties and CTRL-click to select any non-continuous properties without affecting the current selection.

2. Click on the **Up** icon ⬆ to move the selected items up by one position or the **Down** icon ⬇ to move the selected items down by one position.

   In the generated report, the design element properties will be shown in the specified order.

See Also:

Generating HTML Reports

Cartridge Read-Only Mode

Designer provides support for viewing cartridge in read-only mode. When a cartridge is opened in the read-only mode, in all the UI (detail view), the fields are made non-editable, so that the user is not allowed to make changes to it.
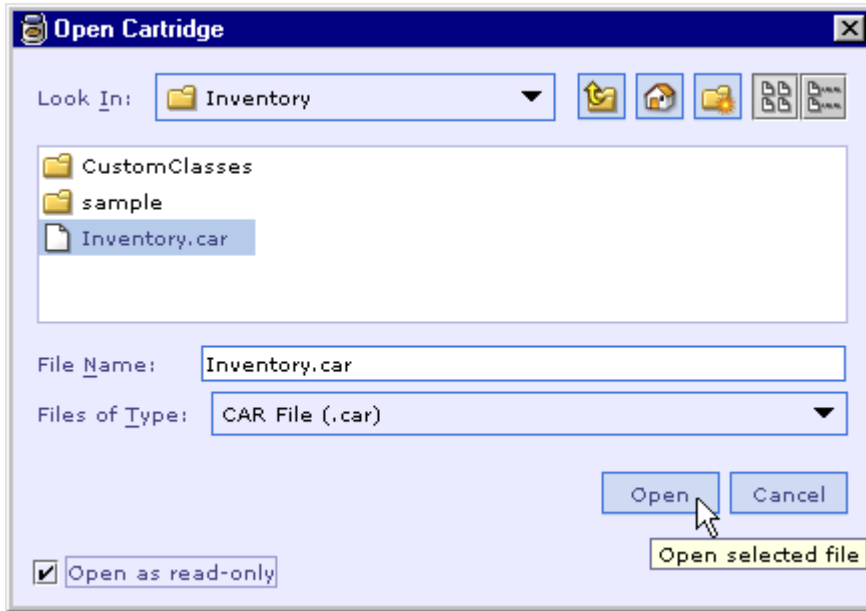
As can be seen in the following picture, the read only nodes are shown with a lock icon overlaid on top of the node icons. The lock indicates that it cannot be modified.
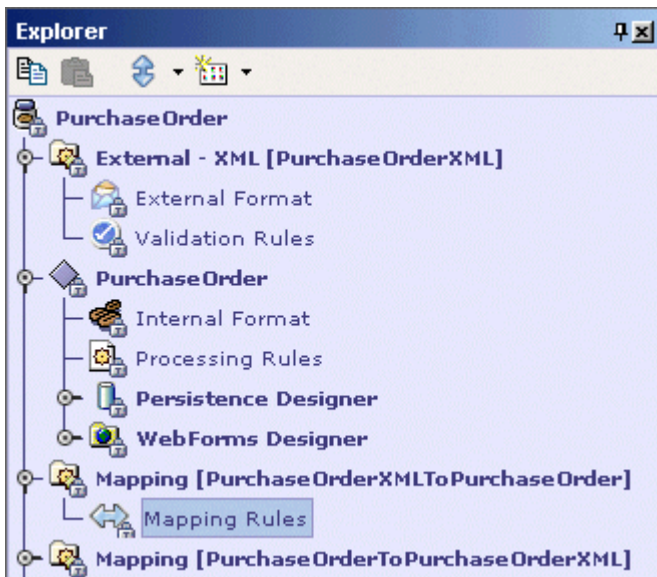
See Also:

Working With Cartridge Designer

Verify Integrity

This feature (is implemented based on) leverages the following new features.

1. Tracing messages in cartridge to standard

2. [Differencing](#)

To verify the integrity of a message or messages in the cartridge,

1. Right click the cartridge node or a message node (internal/external) in the explorer.

2. From the context menu select 'Verify Integrity'. The integrity of all the messages under the selected node is verified and the integrity violations are displayed in the diff view (separate frame).

How it works?

1. For each internal/external message node under the selected node it tries to load the corresponding standard message from the message catalog. If the message node is not based on a standard or if the standard message could not be loaded, a violation is reported (to make sure that the user knows about this). Note that in case of internal messages, it does at present pick up the standard (EIM) from the Repository.

2. The message in the cartridge is compared with the standard message (using the differencing feature). The list of differences is filtered and changes that are allowed are removed from the list (for instance, in case of FIX it is OK to add user defined fields, as long as the tag is greater than 5000).

3. A virtual tree of all the standard messages used is constructed. In the diff view this tree is the reference (left view) against which the messages in the cartridge (right view) are compared.

4. The integrity violations are displayed in the bottom pane.

See Also:

[Diff](#)
[Working With Cartridge Designer](#)

Export/Import a Design Element

Designer provides for exporting/importing a design element in different formats such as XML, HTML, DTD, etc. depending on its type. While the XML and HTML formats are supported for all types of design elements, the other formats are supported only for specific types of design elements. Please note that the HTML format is supported only to export design elements.

Unlike the validation operation, an export/import operation invoked on a design element automatically includes its child elements. For example, the export operation

invoked on an input format not only exports the input format design element but also its External Format, Validation Rules and Mapping Rules child elements.

See Also:

Exporting a Design Element
Importing a Design Element
Copy/Paste
Cartridge Publisher

Exporting a Design Element

The export operation saves the definition of the selected design element in the specified format.

Follow the steps given below to export the definition of a design element into a file.

1. In the **Explorer** pane, select the design element to be exported.

2. Invoke the **Export** command either from the **Edit** menu or from the popup menu of that design element. The shortcut key **Ctrl+Shift+E** can also be used to invoke the **Export** command.

    The **Export** dialog box is shown.
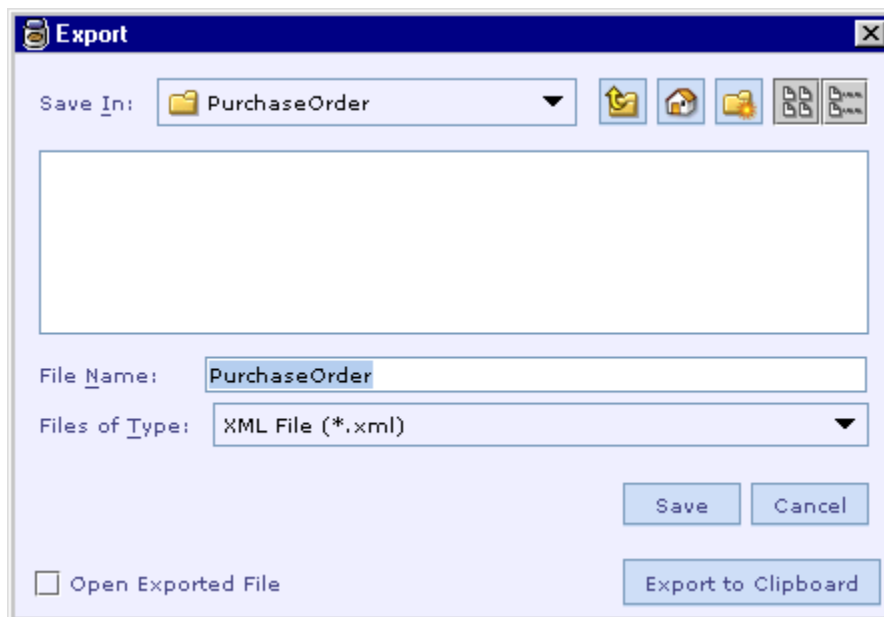
3. Select the destination directory and type-in the target file name in the **File Name** text box.

4. Select the required format from the **Files of Type** drop-down list box.

5. If the exported file needs be opened immediately after its creation, select the **Open Exported File** check box.

6. Click on the **Save** button to start the export operation.

Alternatively you can select the required format and click on the **Export to Clipboard** button to export the design element to the clipboard, ready for pasting elsewhere.

See Also:

Importing a Design Element
Export/Import a Design Element
Copy/Paste

Importing a Design Element

The import operation recreates a design element based on its definition found in the import file. Note that the import operation replaces the definition of the current design element with the imported design element. This means that the current definition of the design element is lost when the import operation is completed.

Follow the steps given below to import the design element definition found in a file (usually a previously exported file).

1. In the **Explorer** pane, select the design element into which the design element definition needs to be imported. Note that the type of the selected design element should match the type of design element definition found in the file.

2. Select the **Import** command either from the **Edit** menu or from the popup menu of the selected design element. The shortcut key **Ctrl+Shift+I** can also be used for invoking the import operation.

   The **Import** dialog box is shown.
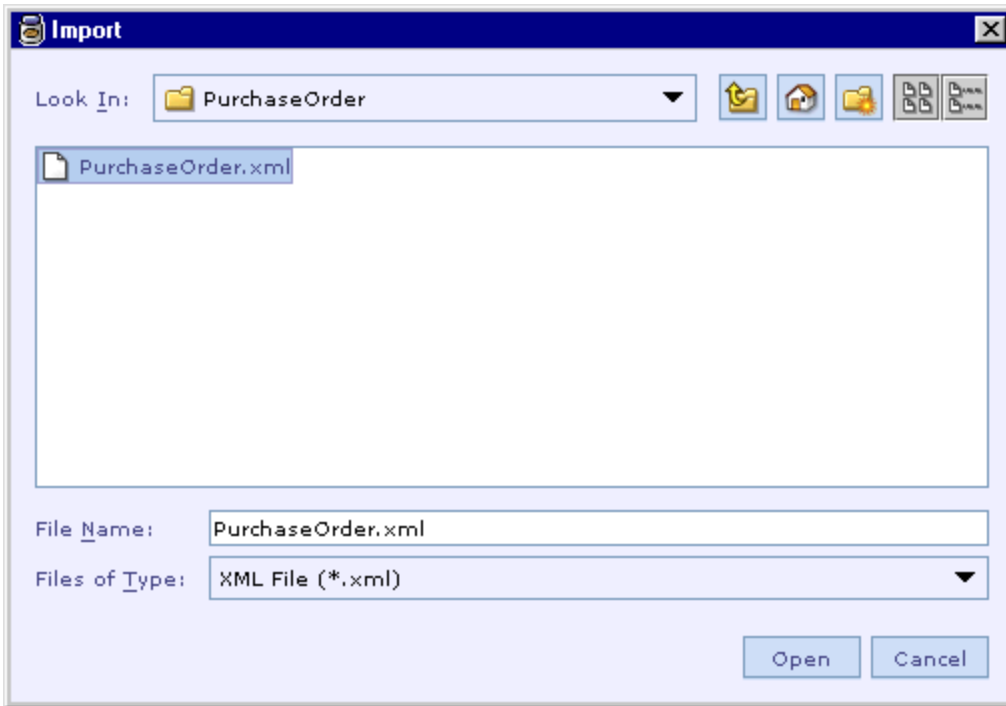
3. Select the type of file to be imported from the **Files of Type** drop down list box.

4. Navigate to the directory that contains the file to be imported and select the required file. Now the file name appears in the **File Name** text box.

5. Click on the **Open** button to start the import operation.

Note

The user needs to save the cartridge to permanently save the imported design element definition.

See Also:

[Exporting a Design Element](#)
[Export/Import a Design Element](#)
[Copy/Paste](#)

Copy/Paste

Designer provides an easy way for creating copies of design elements and fields through the platform independent copy/paste feature. When the copy operation is invoked on a design element or on a field, its definition is exported in XML format and copied into the clipboard as pure text. This makes the copy/paste operation platform independent.

Copy/Paste a Design Element

Designer allows copying a design element and pasting it into another design element. When the paste operation is invoked on a design element other than cartridge and Folder nodes, it replaces the definition of the current design element with the imported design element. When the paste operation is invoked on a cartridge or Folder design element, it creates a new copy of the design element.

Copy a Design Element

Given below are the steps for creating a copy of a design element.

1. Select the design element in the Explorer window.

2. Select the **Edit > Copy** (Ctrl-C) menu item.

This copies the definition of the design element including its child design elements to the clipboard, ready for pasting elsewhere.

Paste a Design Element

When the paste operation is invoked on a design element other than cartridge and Folder nodes, the definition of the copied design element in XML format is imported into it. Note that the paste operation replaces the definition of the current design element with the imported design element. This means that the current definition of the design element is lost when the paste operation is completed.

When the paste operation is invoked on a cartridge or Folder design element, it creates a new copy of the design element.

Given below are the steps for pasting a design element whose definition in the form of XML format is already copied into the clipboard. See the sections [Copy a Design Element](#) and [Exporting a Design Element](#) for copying a design element definition to the clipboard.

1. In the Explorer pane, select the cartridge design element or the internal message design element into which the copied element needs to be pasted. Make sure that the selected design element is the same type as that of the copied design element. Otherwise, a **Paste Error** dialog will be shown to indicate the error.

2. Select the **Edit > Paste** (Ctrl-V) menu item to complete the paste operation.
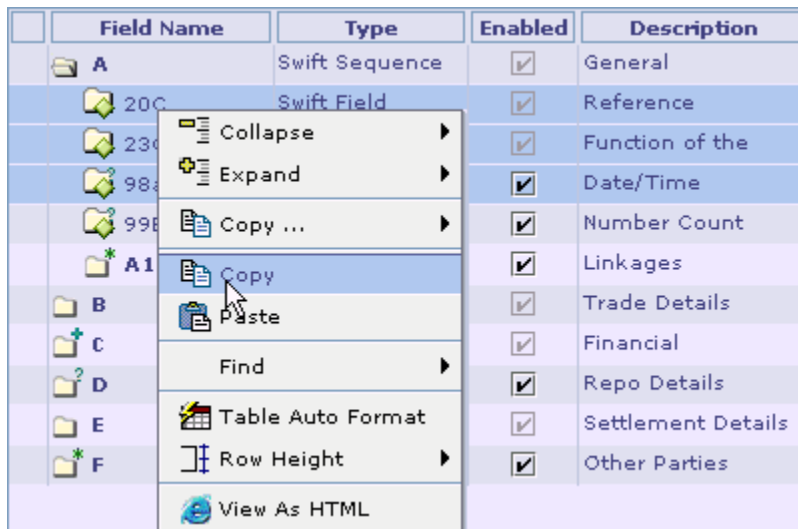
See Also:

Copy/Paste Fields

Designer allows copy/paste of fields between message formats of the same type as well as between message formats of different types. When you copy/paste fields between message formats of same type, all of the field properties are copied properly. On the other hand when you copy/paste fields between message formats of different type, only the information that is common to all formats is copied.

Copy Fields

Given below are the steps for creating a copy of fields. It should be noted that copying a section includes its constituent fields.

1. Select the internal/external message design element in the Explorer window.

2. In the message format UI, select the field(s) to be copied.

3. Select the **Copy** (Ctrl-C) menu item from the short-cut menu that is displayed when you right-click the mouse.

Paste Fields

Designer allows pasting of fields in the form of XML and CSV.

Paste Fields in the Form of XML

Given below are the steps for pasting the fields, which were copied as given in the Copy Fields section.

1. Select the internal/external message design element in the Explorer window.

2. In the message format UI, select the field after which you want to paste the copied fields.

3. Select the **Paste** (Ctrl-V) menu item from the short-cut menu to complete the operation.

Paste Fields in the Form of CSV

To paste fields in the form of CSV, the fields must be specified as shown in the following example.

```
InvoiceDate,DateTime
ClientID,String
ItemID,String
Qty,Integer
Price,Double
```

The following points should be noted in the above CSV:

Here, each line represents a field.

On each line, the first value represents the name of the field and the second value represents the Designer type of that field.

The field name and field type values are separated by a comma

Once you prepare the fields in the form of CSV as shown above, you can copy it and paste it in the internal/external format UI to create fields.

See Also:

Copy/Paste a Design Element
Copy/Paste
Export/Import a Design Element

Copy Name/Qualified Name

Consider the following table.

	Enterprise Element	Type	Hidden	Description
◆	InvoiceDate	DateTime	☐	
◆	ClientID	String	☐	
📁	**Item**	Section	☐	
◆	ItemID	String	☐	
◆	Qty	Integer	☐	

Right clicking from a field or section displays the sub menu items **Name** and **Qualified Name** under the menu 'Copy'.
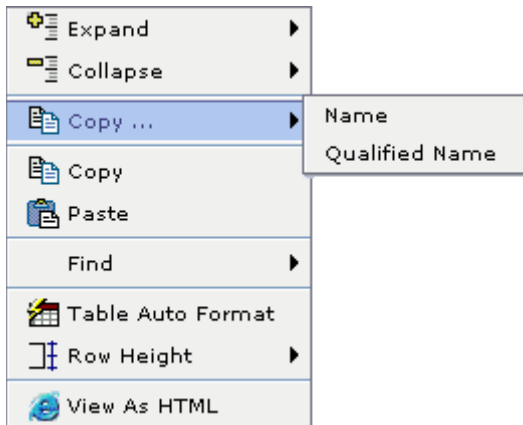
While **Name** menu item copies the name of the selected field/section, the **Qualified Name** menu item copies the name with its qualified path. For e.g. for the sub field 'Qty', the copied name is "Qty" when copied using 'Name' menu item and the copied name is "Item.Qty" when copied using 'Qualified Name' menu item.

See Also:

Copy/Paste a Design Element
Copy/Paste Fields

Validating Design Elements

The **Validate** command 🗎 can be invoked on a design element either using the **Build > Validate** menu item or from its popup menu. The validation operation invoked on a design element validates only that design element; it does not validate its child elements, if any. For example, the validation operation invoked on an external message design element does not validate its **External Format**, and **Validation Rules** child elements.

Note

Selecting the **Build > Validate All** menu item validates the entire cartridge. It invokes the validation operation on all design elements of the cartridge.

The following table lists the validations carried out on each type of design element.

Design Element	Validations
Cartridge	Verifies whether the names of design elements defined in a cartridge are unique.
Internal Message	Verifies the following: 1. whether its transformation name conforms to the identifier rules such as the identifier not using the space character, not starting with a numeric character and so on. 2. whether the internal message name is unique.
Internal Format	Verifies whether the default values specified for normalized fields are valid.
Processing Rules	The following validations are carried out for processing rules: 1. whether the data type of the specified formula matches with the data type of the field. 2. whether the fields used in the processing formula are accessible. 3. whether only one of a processing class or a processing formula is specified for an internal message field. Validations specified for the Validation Rules element are also carried out.
External Message	Verifies the following: 1. whether its transformation name conforms to the identifier rules such as the identifier not using the space character, not starting with a numeric character and so

	on.
	2. whether the message name is unique.
External Format	Validations are specific to the message standard used.
	The fields of the external format are validated.
	A data field validation includes the following:
	1. whether the specified default value is valid (default value data type conforms to the field type).
	A section validation includes the following:
	1. whether the min occurs value is not larger than the max occurs value.
	2. whether the constituent field names are unique.
Validation Rules	Verifies the following:
	1. whether rule names are specified for all validation rules.
	2. whether a custom class reference is provided for a Custom Class validation rule.
	3. whether a formula is specified for a Formula validation rule.
	4. whether the specified 'Applies To' field is a valid field found in the corresponding data format.
	5. whether the fields used in a formula are accessible.
	6. whether the action message is a string value.

| Mapping Rules	Verifies whether the mapping between a source field and a target field is acceptable. For example, one-to-one mapping between a top-level field and a nested field in a repeating section is unacceptable.
Resources	Verifies the following: 1. Whether constituent Resource names are unique. 1. Whether the Resource name is valid. For example, Resource names having '.' Character is invalid. 3. Whether **Value** specified for all Resources.
Function Definition	Verifies the following: 1. whether function name confirms to identifier rules (no spaces, should not start with numeric values, etc.) 2. whether code specified for function, when function is implemented using 'formula code' 3. whether formula code contains error 4. whether there is any return type mismatch 5. whether 'Native Function' reference specified when function is implemented using 'Platform Specific Code'.

See Also:

Find Usage

The user can use this feature to find the usages of a design element defined in a cartridge. For e.g. if an external format has been defined in cartridge he can use this feature to find if this message has been used elsewhere in the cartridge (i.e.) in mapping, message flows etc.

1. Right click the design element for which you want to find the usage. Select 'Find Usage' menu item.
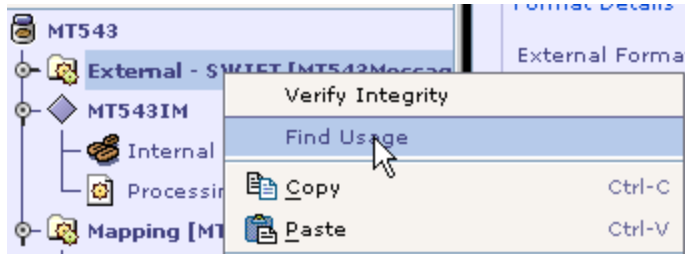
In the above figure 'Find Usage' is done for external message 'MT543Message'.

2. If there are any usages for the element they will be listed in the search results window.


△ Node	Location	Property
MT543MessageToMT543IM		Source Messsage
MT543Flow	MT543Obj	Variable
MT543Flow	Parse	Name

The external message has been used in mapping and message flow. This is listed in the search results windows. Clicking on the link in the window will navigate to the item where the external message has been used.

See Also:

Search Results Window

Find

The user can search for an item of interest in a design element based on the text contained in it. Designer provides the search feature through its **Find** , **Find Next** and **Find in Path** commands. For example, within an internal message format, the user can search for fields/sections containing the text 'Time' as part of its name or description. This search operation will find fields with names such as 'DateofBirth', 'JoinDate', 'SalaryDate' and so on.

See Also:

Find in UI
Find in Path
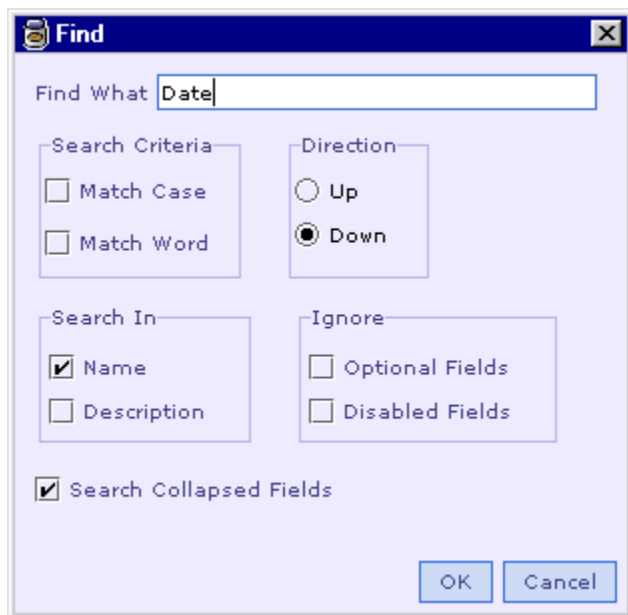Incremental Search
Navigation Features

Find in UI

Follow the steps given below to search for fields/sections of an internal message format whose names contain the text 'Date'.

1. Select the **Internal Format** design element in the explorer pane.

2. Select the **Search > Find** (Ctrl-F) menu item.
   The **Find** dialog box is shown.

3. In the **Find What** text box type the search text 'Date'.

4. Specify other search parameters as required.

   Select the **Match Case** check box in the **Search Criteria** section to find only the text that has the same pattern of upper and lower case as the text specified in the **Find What** text box.

   Select the **Match Word** check box in the **Search Criteria** section to search for occurrences that are whole words and not part of a larger word.

   Specify the direction to search starting from the current field/section by selecting either the **Up** or **Down** radio button in the **Direction** section.

   Select the **Name** check box in the **Search In** section to search the names of fields/sections.

Select the **Description** check box in the **Search In** section to search the descriptions of fields/sections.

Select the **Optional Fields** check box in the **Ignore** section to ignore optional fields/sections from the search operation.

Select the **Disabled Fields** check box in the **Ignore** section to ignore disabled fields/sections form the search operation.

Select the **Search Collapsed Fields** check box so that the collapsed fields are also taken into account in the search operation.

5. Click on the **OK** button to start the search operation.

Find Next

Use the **Search** > **Find Next** (F3) menu item to resume the search operation. The search text that was last entered is again searched for. The search settings that were specified earlier are again used in the search process.
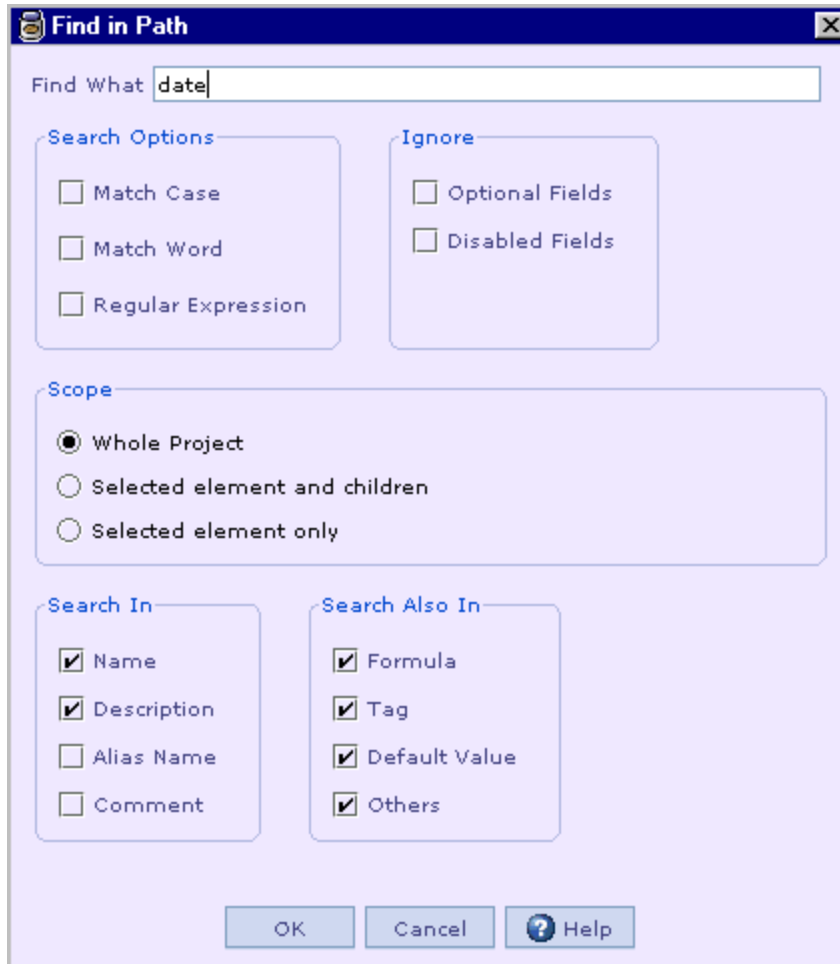
See Also:

[Find in Path](#)
[Find](#)

Find in Path

While the **Find** and **Find Next** commands allow the user to search for matching items one by one in a particular node, the **Find in Path** command can be used to list matching items of all nodes in the **Find** auxiliary window that appears at the bottom of Designer.

Follow the steps given below to search in the entire cartridge for items whose name or description contains the text 'date'.

1. Select the **Search** > **Find in Path** menu item.
   The **Find** dialog box is shown.

Find in Path

Find What: date

Search Options
- ☐ Match Case
- ☐ Match Word
- ☐ Regular Expression

Ignore
- ☐ Optional Fields
- ☐ Disabled Fields

Scope
- ◉ Whole Project
- ○ Selected element and children
- ○ Selected element only

Search In
- ☑ Name
- ☑ Description
- ☐ Alias Name
- ☐ Comment

Search Also In
- ☑ Formula
- ☑ Tag
- ☑ Default Value
- ☑ Others

OK   Cancel   ? Help

2. In the **Find What** text box type the search text 'date'.

3. Select the 'Whole Project' option in the **Scope** section to search for the given text in all nodes of the cartridge.

   The 'Selected element and children' option searches for the given text in the currently selected node and its children. For example, if an input format node is currently selected, the search is carried out in the output format node and its child nodes such as external format node, validation rules node and mapping rules node.

   The 'Selected element only' option searches for the given text only in the currently selected node. For example, if an input format node is currently selected, the search is carried out only in the output format node. Its child nodes are not included in the search.

4. Specify other search parameters as required.

Select the **Match Case** check box in the **Search Options** section to find only the text that has the same pattern of upper and lower case as the text specified in the **Find What** text box.

Select the **Match Word** check box in the **Search Options** section to search for occurrences that are whole words and not part of a larger word.

Select the **Regular Expression** check box in the **Search Options** section to search for occurrences based on the regular expression specified in the 'Find What' text box.

Select the **Optional Fields** check box in the **Ignore** section to ignore optional fields/sections from the search operation.

Select the **Disabled Fields** check box in the **Ignore** section to ignore disabled fields/sections form the search operation.

Select the **Name** check box in the **Search In** section to search the names of items.

Select the **Description** check box in the **Search In** section to search the descriptions of items.

Select the **Alias Name** check box in the **Search In** section to search the alias names of fields/sections.

Select the **Comment** check box in the **Search In** section to search the comments entered for items.

Select the **Formula** check box in the **Search Also In** section to search within formula entered for mapping rules, validation rules, processing rules.

Select the **Tag** check box in the **Search Also In** section to search within field tag values in formats such as FIX.
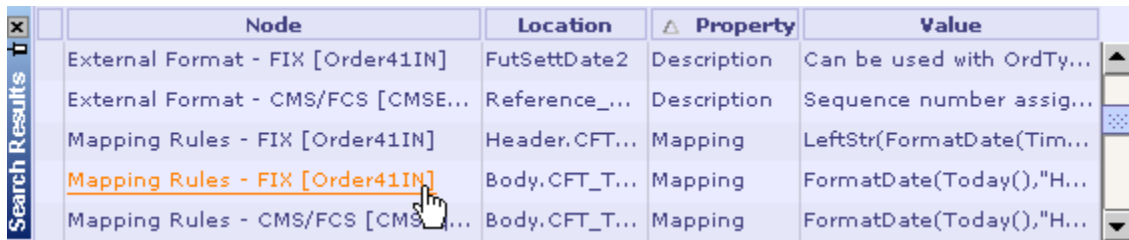
Select the **Default Value** check box in the **Search Also In** section to search the default value entered for fields.

The **Others** option in the **Search Also In** section searches for other attributes of an item that depends on the current node.

For example, in case of the events management node, the search is done in the event handler class name and event handler properties.

5. Click on the **OK** button to start the search operation.

The items that match the criteria are listed in the Search Results auxiliary window shown below.

Here, the **Node** column displays the name of the design element node.

The **Location** column displays the name of the matched item.

The **Property** column displays the matched attribute of the item.

The **Value** column displays the matched text.

As can be seen in the above picture moving the mouse pointer over the node name displays an hyperlink that takes the user to the corresponding item.
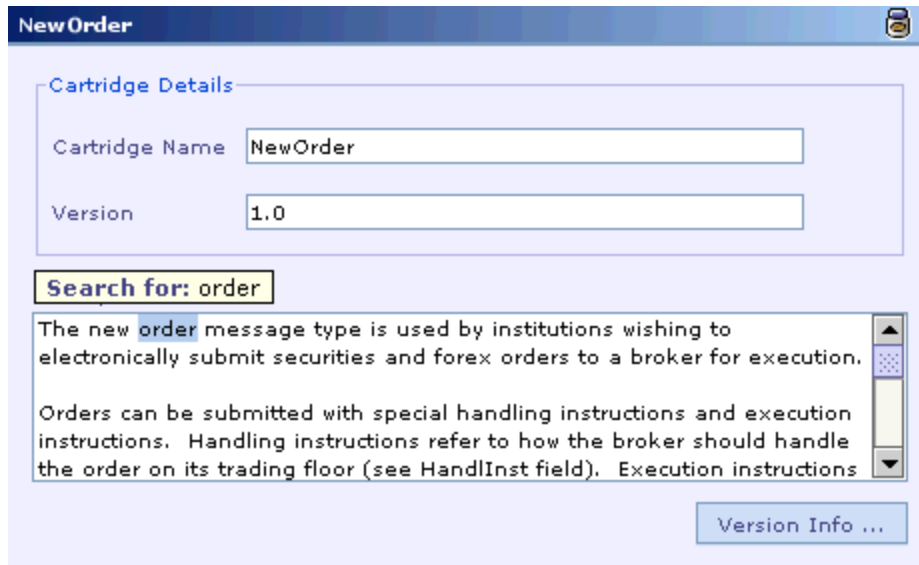
See Also:

Find in UI
Find

Incremental Search

In Designer and Simulator, the user can quickly search for occurrences of text in items like text area elements, tables and formula editor by following the steps given below.

1. Select the item to be searched by clicking in it.

   In case of a table, select the column to be searched by clicking a value in that column.

   Select the **Search** > **Incremental Search** menu item or press the shortcut key CTRL+I (press CTRL+Shift+I for case sensitive search).

2. Type in the search text in the 'Search for:' popup as shown in the following picture.

3. The matching text will be highlighted. If there is no match, the search text will turn red as in **Search for: pin** .

4. Press the Up and Down arrow keys to move to other matching occurrences of text, if any.

See Also:

Find in UI
Find in Path
Working With Cartridge Designer

Drag and Drop

Designer provides an easy way for opening cartridge and data files by supporting the drag and drop feature.

Drag and Drop a Cartridge File

The user can open a cartridge by dragging and dropping a cartridge file (with the .car extension) into the title bar of the main window of Designer. If the dropped file is not a cartridge file, it displays the message dialog indicating the error.

Drag and Drop a Data File

The user can open a data file by dragging and dropping it into the title bar of Simulator or the text area of the input pane.

See Also:

Working With Cartridge Designer

Version Support

Designer provides version support through its design elements. Both core design elements (such as cartridge) and design elements supported by plug-ins (such as external message node) contain version information. When a cartridge is saved, the design elements of that cartridge are stored along with their version information. To successfully open a cartridge file, the version of Designer and its plug-ins should not be earlier than their corresponding design elements stored in that file.
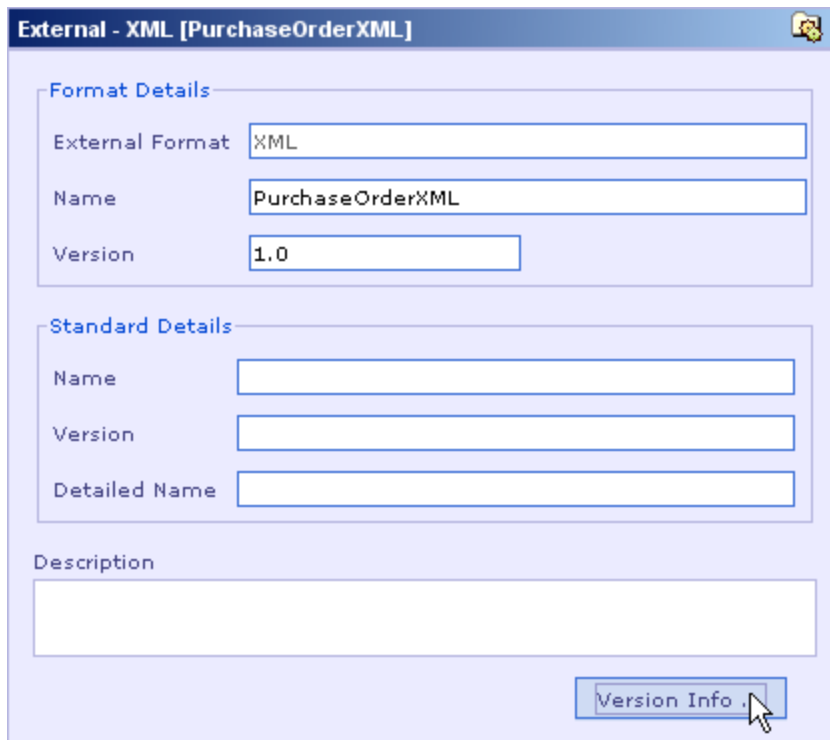
To find out the version of Designer and its plug-ins, select the **Help > Version Info** menu item. It displays the **Version Info** dialog, which contains the version information.

To find out the version of a design element of a cartridge, just click on the **Version Info** button in the design element UI of the corresponding design element.

Follow the steps given below to find out the version of an external message.

1. Select the external message node in the Explorer pane.
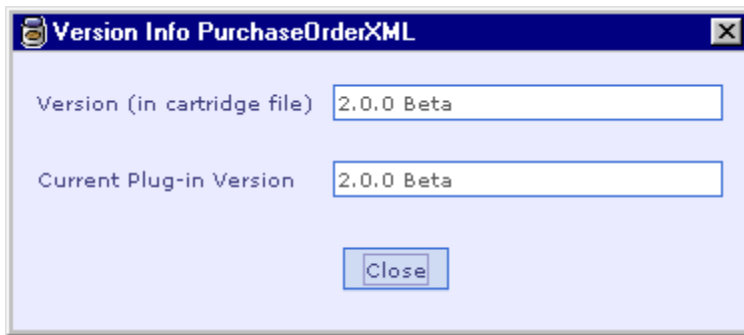
   The External Message UI appears in the Design Element UI pane as shown below.

2. Click on the Version Info button at the bottom of the External Message UI.

The **Version Info** dialog for the selected input format is displayed as shown below.

Here, the version of the input format design element stored in the cartridge file is 1.0.0 and the version of the XML Format Plug-in installed with Designer is 1.6.0.

Note

In the above example, the version of the XML Format Plug-in is later than the version of the input format design element stored in the cartridge file. If the user chooses to save this cartridge, the XML input format currently opened in Designer will be saved with the latest XML Format Plug-in. So the version of XML input format design element would also be changed from 1.0.0 to 1.6.0.

See Also:

Working With Cartridge Designer