Oracle® Cloud

Using the Oracle Mapper with Oracle Integration 3

F45538-06
October 2023

Oracle Cloud Using the Oracle Mapper with Oracle Integration 3,

F45538-06

Primary Author: Oracle Corporation

Contents

4 Mapper Use Cases

5 Troubleshoot the Mapper

Preface

Using the Oracle Mapper with Oracle Integration 3 describes how to use the mapper to map source data structures to target data structures.

> **Note:**
>
> The information in this guide applies to all of your Oracle Integration instances. It doesn't matter which edition you're using, what features you have, or who manages your cloud environment. You'll find what you need here, including notes about any differences between the various flavors of Oracle Integration when necessary.

Topics:

Audience

Using the Oracle Mapper with Oracle Integration 3 is intended for users who want to use the mapper to map source data structures to target data structures.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and

partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Resources

See these Oracle resources:

- Oracle Cloud at `http://cloud.oracle.com`
- *Using Integrations in Oracle Integration 3*
- Adapter documentation in the Oracle Cloud Library on the Oracle Help Center

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Get Started with the Mapper

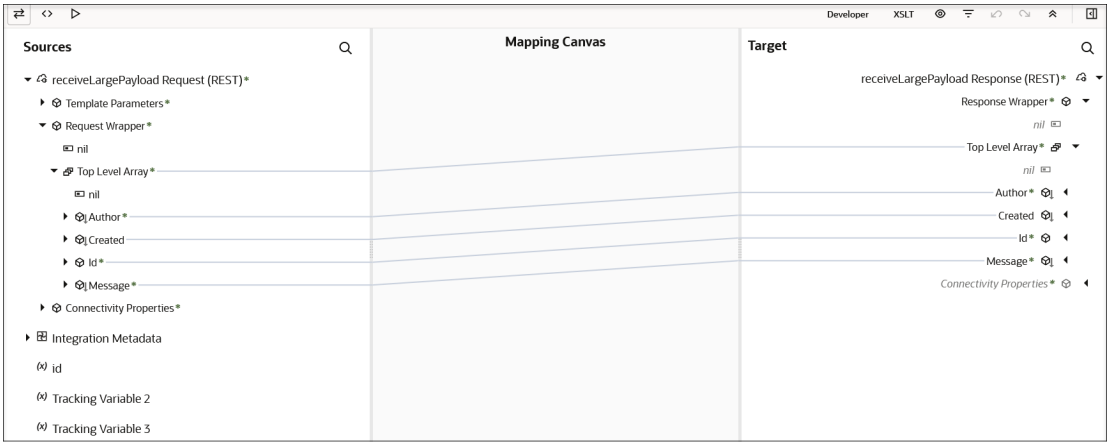
Review the following topics for an overview of how to use the mapper to map source data structures to target data structures.

Topics

About Mappings

One of the key tasks to any integration is defining how data is transferred, or *mapped*, between two applications.

In most cases, the messages you want to transfer between the applications in an integration have different data structures. A visual mapper enables you to map element nodes between applications by dragging source element nodes onto target element nodes. When you open the mapper for a request or response message in an integration, the data structures are automatically populated with the information pulled from the source and target connections. You can expand and load data structure levels on demand to display additional levels. There is no limit on the levels of display.



The maps you create are called transformation maps, and use the eXtensible Stylesheet Language (XSL) to describe the data mappings, which let you perform complex data manipulation and transformation. A standard set of XSLT constructs are provided (for

example, `xsl:if`, `xsl:for-each`, and others). A specialized function is also provided for you to reference lookups directly from the mapper.

> **Note:**
>
> The mapper supports XSL version 2.0. Version 1.0 is not supported.

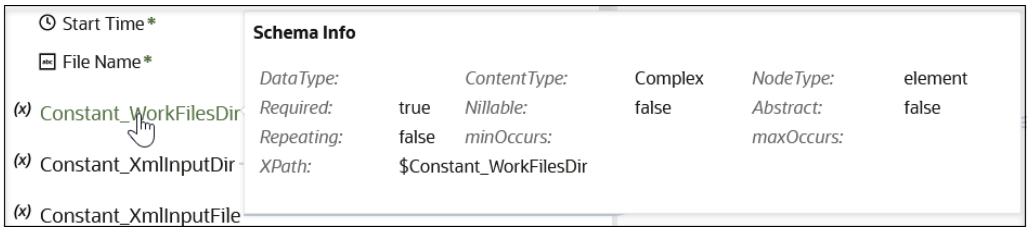
The mapper supports both qualified and unqualified schemas (that is, schemas without `elementFormDefault="qualified"`). Elements and attributes with and without namespace prefixes are also supported.

Substitution groups in schemas are supported. You can see all the substitutable elements in a base element in the mapper, and select the one to use.

Extended data types are also supported.

Elements and attributes for which mapping is required are identified by a blue asterisk (*) to the left of their names. To display only required fields, click the **Filter** icon in the mapper toolbar, select **Required Fields**, and click **Apply**.

You can also right-click elements and attributes and select **Node Info** to show specific schema details such as the data type, if mapping is required, and so on.



Additional custom annotations can also be displayed. These annotations are currently only available with the Oracle Engagement Cloud Adapter. The Oracle Engagement Cloud Adapter obtains this information from the applications and annotates it in the integration WSDL. This information is then read and made visible as annotations in the mapper (for example, title and description). This information can help you better understand what data is being mapped.

The mapper toolbar provides the following functionality.

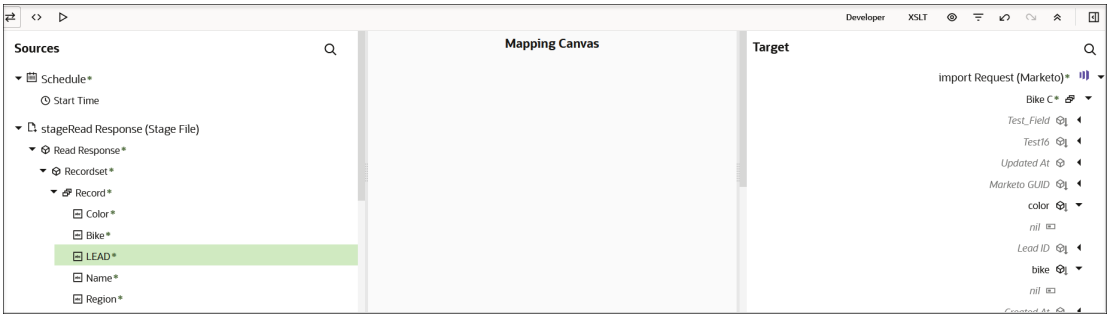
Element	Description
	Click to return to the mapping canvas when you are inside the Code or Test page.
	You can view the XSLT code being created as you design your mappings.
	Once you complete designing your mappings, you can test them by entering sample content of the message to process in the mapping tester.
Developer	Click to disable user-friendly, source and target element names in the mapper. By default, user-friendly element names are shown.
XSLT	Click to show the XSLT functions.

Element	Description
	You can select the following options: • Select to show the namespace prefixes on source and target element nodes. • Select to show the types (prefixes and data types) on source and target element nodes.
	You can filter the display of element nodes, error messages, and warnings in the source or target data structures.
	You can select to undo the previous action performed in the mapper. For example, if you perform a mapping, then press this button, the mapping is removed. The link is disabled when all actions have been undone.
	You can redo the action that was undone.
	You can maximize the size of the mapper. This is useful when working with large schemas.
	You can add functions, operators, and XSLT expressions to your mappings.

About Mapping Data Between Applications

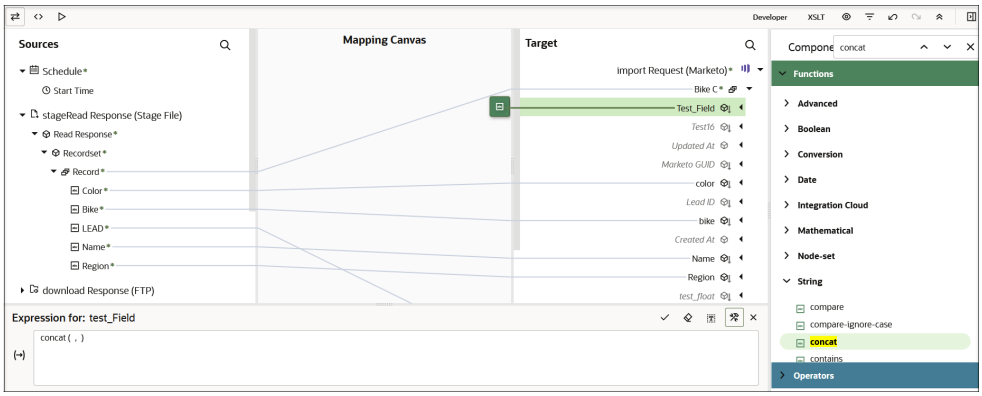
Once you create an integration and have the source and target connections in place, you can define how data is mapped between the element nodes in the two data structures.

The mapper appears with the element nodes of the source data structure on the left and the target data structure on the right.



1. To map fields directly, click a source element nodes and drag it to the corresponding field in the target element node.

   A line connects the two nodes. An Expression Builder below the mapper is displayed to show the XPath expression.



2. To use functions, operators, or XSLT statements in your mapping, see Work with Functions, Operators, and XSLT Statements.
3. When you are done mapping data, click **<**, then click **Apply** to save your changes when prompted. You can also click **Validate** to save your changes.

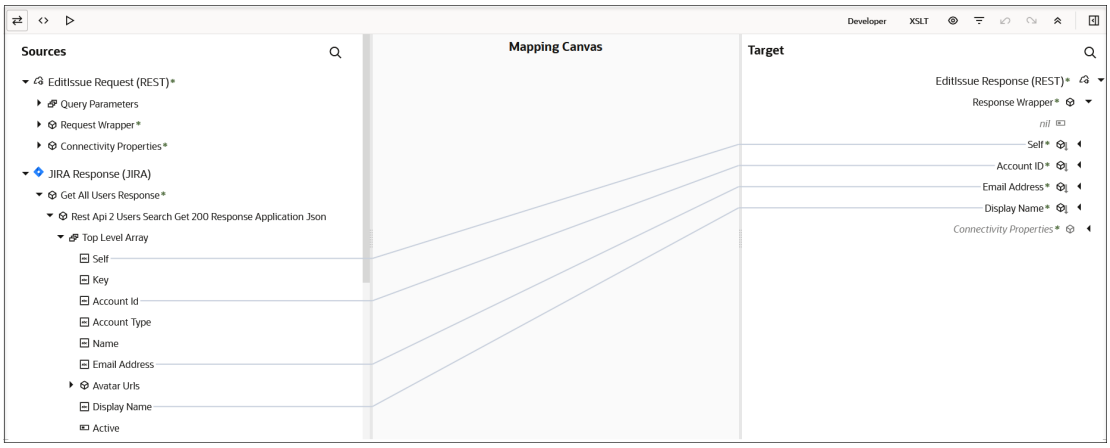
View User-Friendly Element Names

You can view user-friendly display names instead of technical names for source and target elements in the mapper tree and for expressions in the Expression Builder. This eliminates the need to try and understand the technical, often cryptic, names that are difficult to correlate to the user-friendly display names you see in the endpoint application's user interface. User-friendly names are displayed by default, but you can also toggle to the technical names.

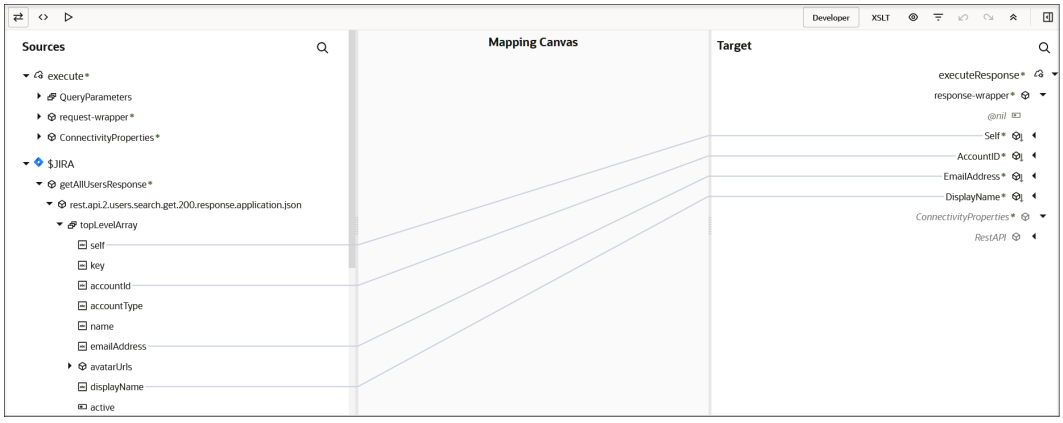
- Toggle Between User-Friendly Names and Technical Names
- Adapter Names
- Root Elements in Source and Target Trees
- Child Elements in Source and Target Trees
- Search For Data in the Source and Target Trees
- User-Friendly Expression for Mapping
- Expression Builder
- Other Sections of the User Interface

Toggle Between User-Friendly Names and Technical Names

By default, user-friendly names are displayed in the source and target mapper trees when you open the mapper. Name display is controlled by the **Developer** button at the top of the mapper.



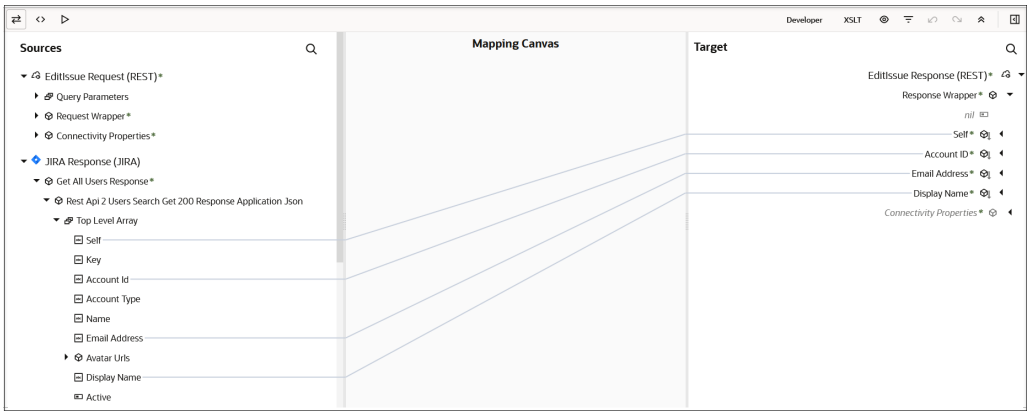
1. Click **Developer** to switch to technical names.
   An outline appears around the button, and technical names for the source and target elements are displayed.



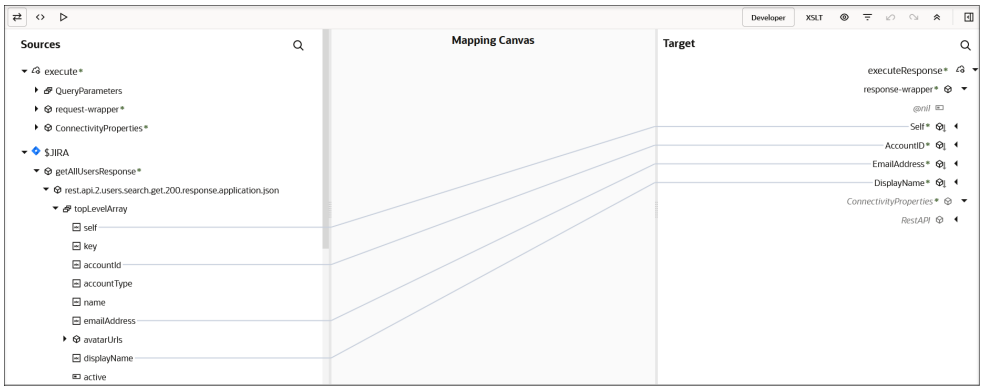
2. Click **Developer** to switch back to user-friendly names.

Adapter Names

In user-friendly mode, adapter names are displayed along with the adapter's associated icon and the type of payload (request/response). For this example, the source REST Adapter and Jira Adapter and the target REST Adapter are displayed.



1. Click **Developer** to switch to technical names. The adapter names are removed.



Root Elements in Source and Target Trees

User-friendly names for the root elements of the different payloads enable you to easily correlate them with the associated invoke/trigger connection, the adapter used, and the type of payload (request/response). The icon of the root element corresponds to the associated adapter.

The format of user-friendly names for the root elements differs based on the variable type or the associated adapter. The following table lists the format of user-friendly names for the root elements for different variable types.

Adapter/Variable Type	Format of User-Friendly Name	Example
Application Adapter	*trigger/invoke_action_name payload_type (request/response) (Associated_Adapter_Name)*	SendInventoryAdjustments Request (SOAP)
System Adapter	See the Example column.	Schedule - Schedule $self (for technical mode) or Integration Metadata (for user-friendly mode)

Adapter/Variable Type	Format of User-Friendly Name	Example
Tracking Variables	If a user-friendly name is entered for the tracking variable in the Business Identifiers For Tracking page, that becomes the user-friendly name for the variable in the mapper. If the **Tracking Name** field in the Business Identifiers For Tracking page is not populated for the variable, the system constructs the user-friendly name for the tracking variable in the format of Tracking Variable 1/2/3.	My Business Identifier Tracking Variable 1 Tracking Variable 2 Tracking Variable 3
Other Variables	For all other variables (that is, simple variables and the root element of the complex variables), the user-friendly name is automatically constructed using the name with which the variable was created (without the `$` prefix).	counter studentName

Child Elements in Source and Target Trees

The user-friendly names for the child elements in the source and target trees are derived from the associated schema files. If the schema files are generated with user-friendly names for the elements, the elements get rendered with those names in user-friendly mode in the mapper.

If the schema files do not contain user-friendly names for the elements defined, the child elements are displayed with the technical name in both user-friendly mode and technical mode.

The attributes of the schema elements are rendered with the `@` prefix followed by the attribute name in the mapper. With user-friendly names, the `@` prefix is not appended to the front of the name or in user-friendly mode. In technical mode, the attributes are shown appended with the `@` prefix.

User-friendly names do not include the namespace prefix. The option to view element names with the prefix **Show prefixes** available in the **View** menu of the mapper is disabled when the mapper is in user-friendly mode. The option is enabled once you switch to technical mode.

Search For Data in the Source and Target Trees

The source and target trees can be searched with the element name in either user-friendly mode or technical mode.

For example, assume the mapper is in user-friendly mode and an element exists whose user-friendly name is `BEG: Beginning Segment for Purchase Order` and technical name is `BegSegPO`. If `SegPO` is the search string used to search for the element, the search highlights the element irrespective of your current mode.

User-Friendly Expression for Mapping

Just as the source and target element technical names are simplified by their user-friendly names, the mapping expression created is represented in a simplified form.

This is a user interface-only entity. That is, the user friendly expression for a mapping is displayed in the mapper. However, it does not get saved in the XSL file. Click the **Code** tab of the mapper after creating a mapping. The **Code** tab shows the XSL file that is generated behind the scenes. Note that the file contains only the technical mapping, and not the user-friendly expression. The mappings work as they always have at runtime. At design time, the mapper displays the mappings as user-friendly expressions in user-friendly mode and as technical expressions in technical mode.

The user-friendly expression for a mapping is created when a mapping is constructed in the mapper. The user-friendly expression is created based on the user-friendly name for the components in the mapping.

Consider the following mapping:

```
concat($EDI-Translate/nsmpr0:executeResponse/ns31:TranslateOutput/
ns31:translation-status,
$EDI-Translate/nsmpr0:executeResponse/ns31:TranslateOutput/
ns31:tracking-info)
```

This mapping refers to a `concat` function whose parameters are two elements from the payload. The user-friendly expression for this mapping is as follows:

```
concat( translation-status, tracking-info)
```

where:

- `translation-status` is the user-friendly name of the element `$EDI-Translate/nsmpr0:executeResponse/ns31:TranslateOutput/ns31:translation-status`
- `tracking-info` is the user-friendly name of the element `$EDI-Translate/nsmpr0:executeResponse/ns31:TranslateOutput/ns31:tracking-info`

Expression Builder

When you navigate to the mapper, the Expression Builder launches in user-friendly mode by default when you select a target element.

The Expression Builder shows the mapping for the target element selected. As with the mapper, the Expression Builder also has two modes. User-friendly mode shows the mapping as a user-friendly expression.

Figure 1-1 User-Friendly Names in Expression Builder

To toggle the Expression Builder between the two modes, click  on the right side. You can manually edit the existing mapping in the Expression Builder.

Figure 1-2 Technical Names in the Expression Builder

Other Sections of the User Interface

Other sections of the mapper in which the source and target elements are displayed all show the names in synchronization with the mode that is selected for the mapper (user-friendly or technical). For example:

- The **Test** button (where the root elements of each source are displayed as the headers of the tabs)
- The **Filter** button (where one of the options to filter the tree data is by source name, which shows the root elements of the different sources)

This means that if the mapper is in user-friendly mode, these sections of the user interface also show the user-friendly names of the elements. If the mapper is in technical mode, these sections show the technical names of the elements.

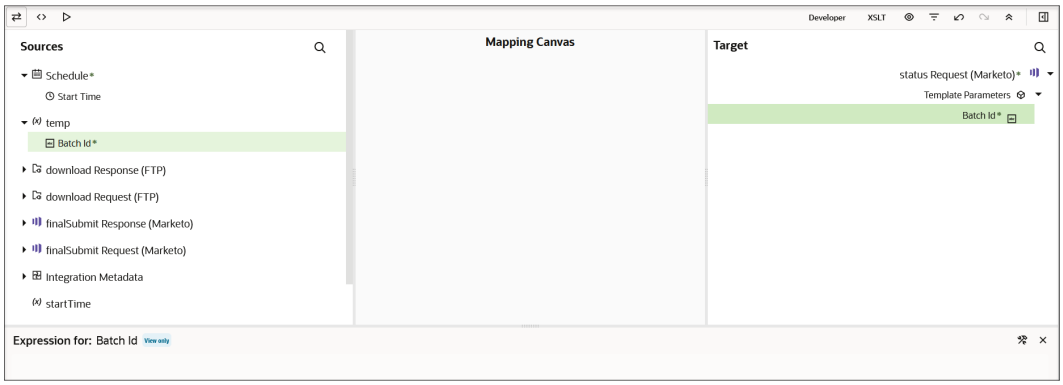
About the Expression Builder

Use the Expression Builder to view and edit your XPath expressions. This section provides an overview of the Expression Builder.

Displaying the Expression Builder

1. Click a target element node.

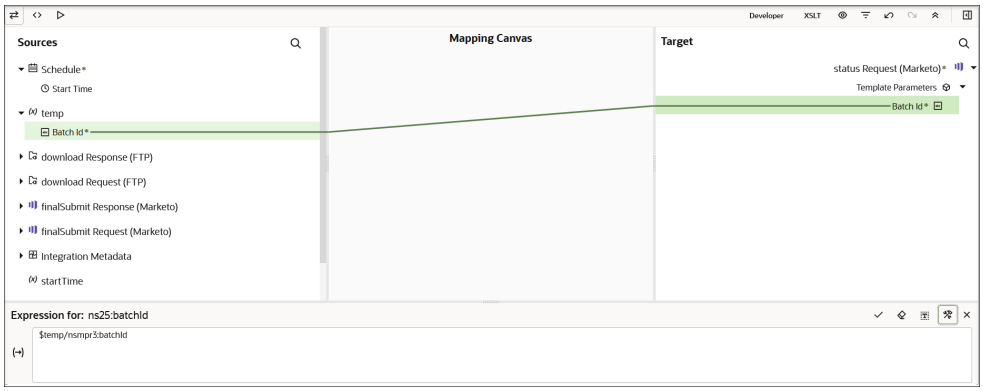
   The Expression Builder is displayed. A **Switch to Developer View** icon and **Close** icon are displayed on the right side of the Expression Builder.



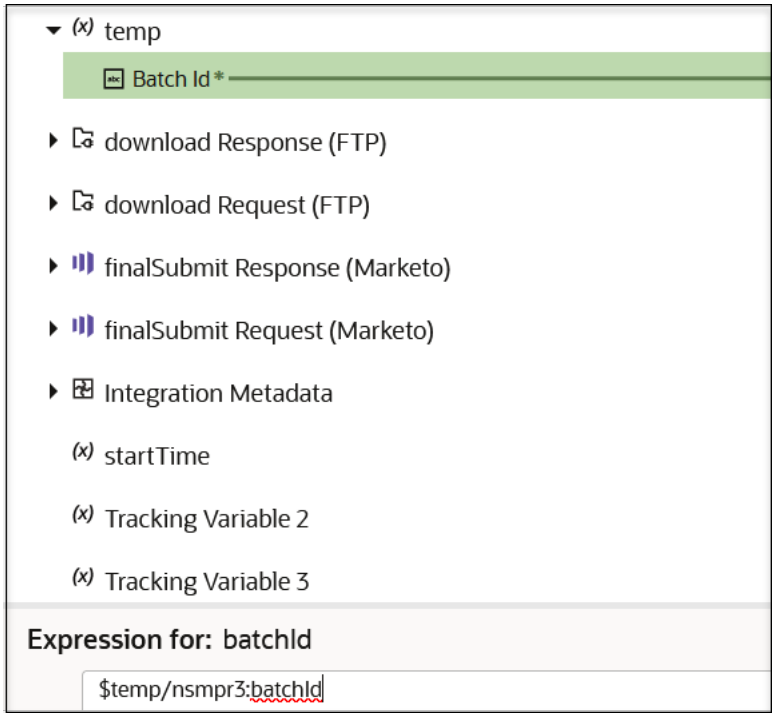
2. Drag a source element node to a target element node.

   The XPath expression is added to the Expression Builder.

3. Click **Switch to Developer View** to view the developer path of the mapping.



4. If you want to remove the value, click **Erase** , then click **Save** to completely remove the mapping.
5. Drag the source element node to the Expression Builder. You can also highlight the source element node and click **Insert selected item** to add a value to the Expression Builder.



6. Click **Save** to save the mapping.

Using Set Text Mode

When there is no mapping in the Expression Builder, there is a **Set Text** button. This option enables you to enter text in an element node. You can only have XPath expression or text in the Expression Builder. You cannot have both types.

1. Highlight a target element node and click **Set Text** in the Expression Builder to enter set text mode.
2. Enter text in the Expression Builder.

   A letter icon is added to the node. When you place your cursor over the icon, the value you entered is displayed.

> **Note:**
>
> If you drag a source target node into the Expression Builder while in set text mode, the mapping value is literally added as text, and not as an XPath expression.

Entering Literal Values

You can enter literal values in the Expression Builder when you are *not* in set text mode.

1. Enter text in the Expression Builder.

   This creates a `value-of` expression in the XSTL file instead of straight text. See Edit XSLT Code in the Mapper.

Save Changes Automatically

As you work in the mapper, your changes are automatically saved. User actions in the mapper such as creating a mapping line by dragging and dropping from source to target or saving in the Expression Builder are saved automatically internally without the need to press **Validate**.

This is useful for scenarios in which your browser crashes or your instance undergoes patching while you are working in the mapper. When you resume editing, you are given an option to edit and recover your changes.

Access the Mapper

To create mappings in an integration, you need to first access the mapper.

As you add triggers and invokes to an app driven Orchestration integration, a map icon is automatically added. You can also add ad-hoc mappings to this type of integration, such as adding a mapper to a switch action.

1. Double-click a mapper icon or select the mapper icon and click **Actions** ⋮, then select **Edit**.

   For mappers in active integrations, you can only click **Actions** ⋮, then select **View**. Note the following details:

   - You cannot add or edit mappings.
   - You cannot validate mappings.
   - You cannot save or erase the XPath expression in the Expression Builder.
   - You cannot create or delete elements or mappings in the target context menus.
   - You cannot drag source element nodes to target element nodes.
   - You can view XSLT code and test your mappings.

2. See Creating Integrations.

2

Map Data

Use the mapper to drag element nodes in the source structure to element nodes in the target structure.

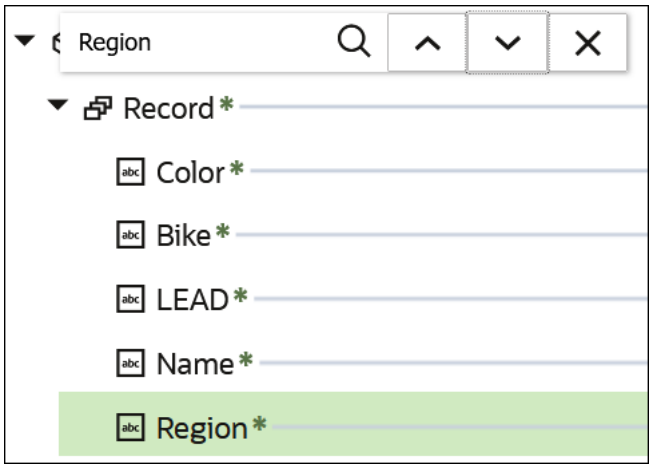
Topics

Search Data Fields

The mapper displays the source data structure on the left and the target data structure on the right. You can search for specific element nodes or attributes (identified by the icon) in either the source or target structure.

To search data fields:

1. In the **Sources** or **Target** section, click **Search** .
2. Enter the full or partial name, and click **Search** .



The tree is automatically expanded and scrolls to the first match. If you entered straight text (for example, `Region`), any element nodes and attributes of the same name are found. If you search by attribute (for example, `@Region`), only the attributes of the same name are displayed.

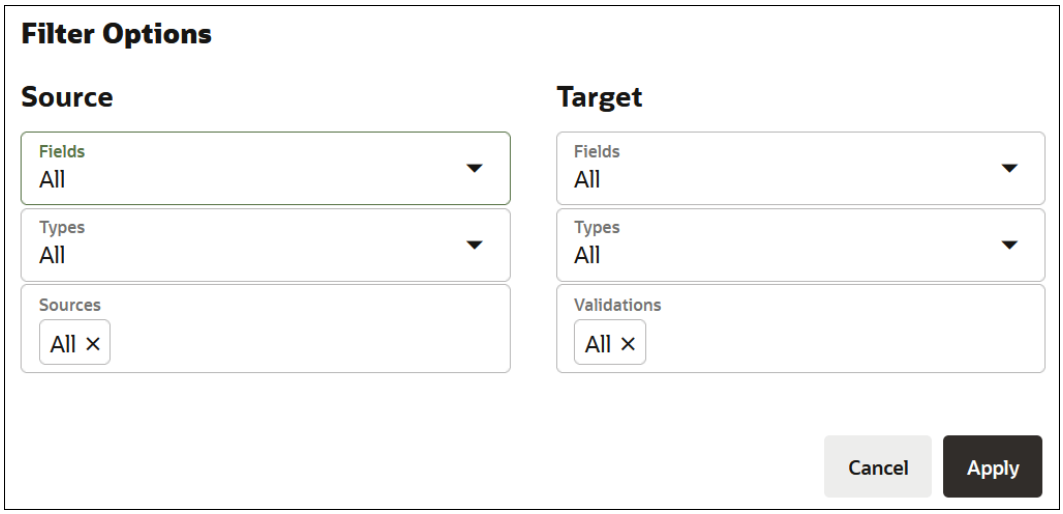
3. Click **Next** to scroll to the next match.
4. When done, click **X** to dismiss the search facility.

Filter the Source or Target Data Structures

You can filter the display of the source and target structures. This enables you to show only the details in which you are interested.

To filter the source or target data structures:

1. Click **Filter** in the **Target** section of the mapping toolbar.
2. Specify map filtering options based on the following criteria.
   - View the mapped element nodes, unmapped element nodes, or both.
   - View all element node types (required element nodes and custom element nodes you created in a prebuilt Oracle Integration that was edited in customization mode).
   - View the source data structures in the integration (main source and secondary sources).
   - View validation details (view only errors, only warnings, or only mappings with no issues).



3. Click **Apply**.

   Based on your selections, icons are displayed in the mapper toolbar. For example, **Mapped** is displayed for both data structures if you selected to show mapped element nodes in both the **Sources** and **Target** sections.

4. To remove the selected filtering, click the **x** on **Mapped**.

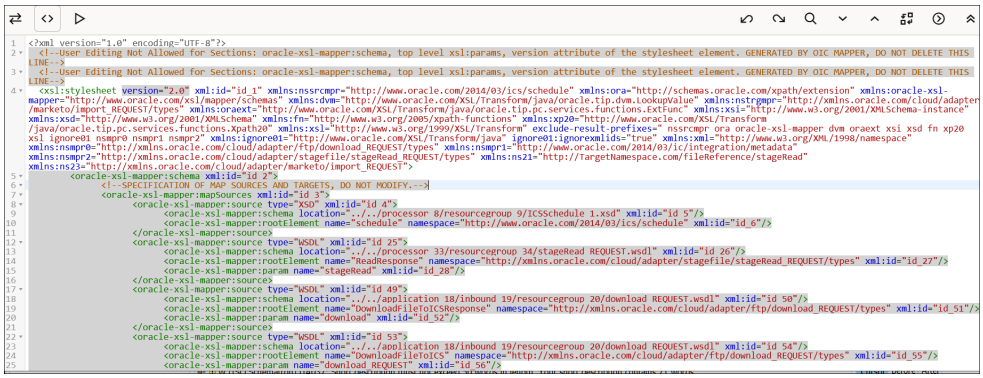
Edit XSLT Code in the Mapper

You can directly edit the XSLT code of your mappings inside the mapper. This action is useful for use cases in which mapping is not possible in the graphical mapper. This eliminates the need to export your XSLT code from Oracle Integration, edit the code manually in a text editor or in a separate graphical tool such as Eclipse or Oracle JDeveloper, and then re-import the code into the mapper in Oracle Integration.

Editing of XSLT code for advanced use cases such as the following is supported:

- Create internal variables using `<xsl:variable>`
- Correlate multiple sources grouped by key fields using `<xsl:for-each-group>`
- Dynamically create target name-value pairs based on runtime data using `<xsl:element>`, `<xsl:attribute>`
- Implement "push style" XSLT using `<xsl:template>`, `<xsl:call-template>`, and `<xsl:apply-templates>`
- Write your own functions in XSLT using `<xsl:function>`
- Copy node sets using `<xsl:copy>` and `<xsl:copy-of>`

1. Click **Code**.
   The current XSLT code for your mappings is displayed.



See the following sections for XSLT code editing guidelines:

- Restrictions on Editing
- Toolbar Options
- Edit the XSLT Code
- Code Validation

Restrictions on Editing

The following sections of your XSLT code remain read-only and cannot be edited.

- Attribute version numbers:

```
<xsl:stylesheet version="2.0"
```

- The mapper schema section where sources and targets are specified:

```
<oracle-xsl-mapper:schema xml:id="id_2">
              <!--SPECIFICATION OF MAP SOURCES AND TARGETS, DO NOT
MODIFY.-->
              <oracle-xsl-mapper:mapSources xml:id="id_3">
                    <oracle-xsl-mapper:source type="WSDL"
xml:id="id_4">
                          <oracle-xsl-mapper:schema location="../../
application_8/outbound_9/resourcegroup_10/echoRequest_REQUEST.wsdl"
xml:id="id_5"/>
                          <oracle-xsl-mapper:rootElement
name="execute" namespace="http://xmlns.oracle.com/cloud/adapter/
REST/echoRequest_REQUEST/types" xml:id="id_6"/>
                    </oracle-xsl-mapper:source>
                    <oracle-xsl-mapper:source type="WSDL"
xml:id="id_20">
                          <oracle-xsl-mapper:schema location="../../
processor_83/resourcegroup_84/sList_REQUEST.wsdl" xml:id="id_21"/>
                          <oracle-xsl-mapper:rootElement
name="ListFilesResponse" namespace="http://xmlns.oracle.com/cloud/
adapter/stagefile/sList_REQUEST/types" xml:id="id_22"/>
                          <oracle-xsl-mapper:param name="sList"
xml:id="id_23"/>
                    </oracle-xsl-mapper:source>
                    <oracle-xsl-mapper:source type="XSD"
```

```
xml:id="id_24">
                        <oracle-xsl-mapper:schema location="../../
processor_27/resourcegroup_28/ICSIntegrationMetadata.xsd" xml:id="id_25"/>
                        <oracle-xsl-mapper:rootElement name="metadata"
namespace="http://www.oracle.com/2014/03/ic/integration/metadata"
xml:id="id_26"/>
                        <oracle-xsl-mapper:param name="self"
xml:id="id_27"/>
                  </oracle-xsl-mapper:source>
            </oracle-xsl-mapper:mapSources>
            <oracle-xsl-mapper:mapTargets xml:id="id_7">
                  <oracle-xsl-mapper:target type="WSDL" xml:id="id_8">
                        <oracle-xsl-mapper:schema location="../../
application_8/outbound_9/resourcegroup_10/echoRequest_REQUEST.wsdl"
xml:id="id_9"/>
                        <oracle-xsl-mapper:rootElement
name="executeResponse" namespace="http://xmlns.oracle.com/cloud/adapter/
REST/echoRequest_REQUEST/types" xml:id="id_10"/>
                  </oracle-xsl-mapper:target>
            </oracle-xsl-mapper:mapTargets>
            <!--GENERATED BY ORACLE XSL MAPPER 12.1.2.0.0-->
      </oracle-xsl-mapper:schema>
```

- The global parameter declaration section:

```
<xsl:param name="sList" xml:id="id_28"/>
      <xsl:param name="self" xml:id="id_29"/>
      <xsl:param name="tracking_var_1" xml:id="id_30"/>
      <xsl:param name="tracking_var_2" xml:id="id_31"/>
      <xsl:param name="tracking_var_3" xml:id="id_32"/>
      <xsl:param name="var_assignment_1" xml:id="id_33"/>
```

Toolbar Options

The toolbar above your XSLT code provides a series of shortcuts for navigating through and editing XSLT code.

Option	Description
	Undo your last editing changes to the XSLT code.
	Redo your last editing changes to the XSLT code.
	Search for specific entries in your XSLT code. Click **Next** and **Previous**  to navigate through the code.

Option	Description

	Find and replace specific entries in your XSLT code.
	1. Enter the text to replace, and press **Enter**:
	`Replace: message`
	2. Enter the text to substitute, and press **Enter**:
	`With: message1`
	3. Replace the text individually or globally when prompted.
	`Replace? Yes No All Stop`

	Enter the line number in your XSLT code to access.

Edit the XSLT Code

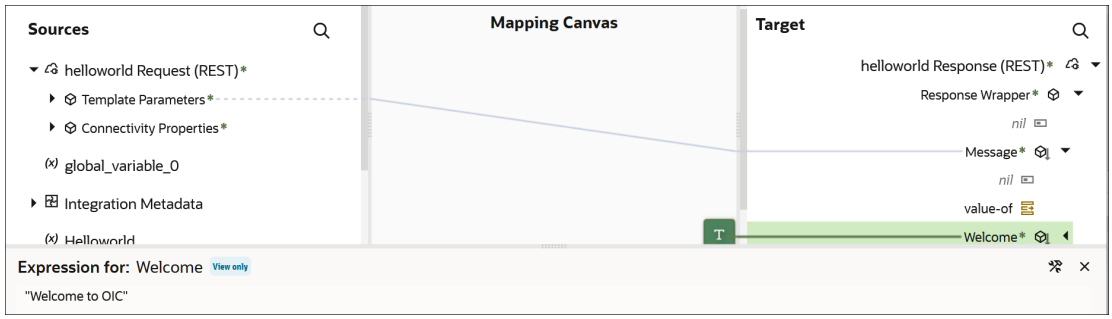
The following section of your XSLT code is where the changes you make in the graphical designer are reflected. You can also edit the XSLT code in this section and see your changes reflected in the designer.

```
<xsl:template match="/" xml:id="id_11">
            <nstrgmpr:executeResponse xml:id="id_12">
                  <nstrgdfl:response-wrapper xml:id="id_16">
                        <nstrgdfl:Message xml:id="id_17">
                              <xsl:value-of select="/nstrgmpr:execute/
nstrgmpr:TemplateParameters/nsmpr0:message" xml:id="id_18"/>
                        </nstrgdfl:Message>
                        <nstrgdfl:Welcome xml:id="id_19">"Welcome to
OIC!!!  Echo was successful."</nstrgdfl:Welcome>
                  </nstrgdfl:response-wrapper>
            </nstrgmpr:executeResponse>
      </xsl:template>
```

For example, assume you have the following XSLT code that you want to replace:

```
<nstrgdfl:Welcome
             <nstrgdfl:Welcome xml:id="id_19">"Welcome to OIC!!! Echo
was successful."
</nstrgdfl:Welcome>
```

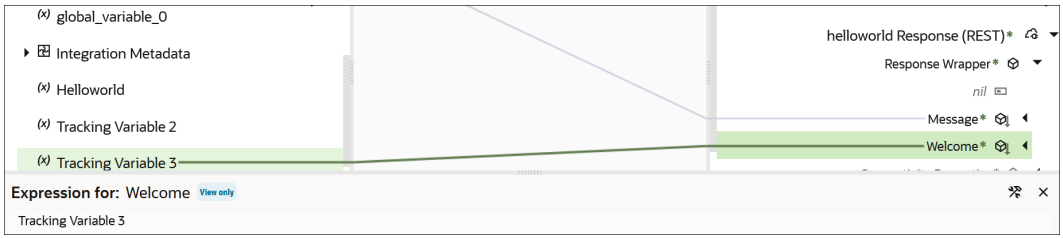
This code is currently visible in the mapper as literal text:



1. Replace the literal value of `"Welcome to OIC"` with the variable `$tracking_var_3`:

```
<nstrgdfl:Welcome xml:id="id_19">
            <xsl:value-of select="$tracking_var_3" xml:id="id_18"/>
</nstrgdfl:Welcome>
```

2. Click **Designer**  .
The designer is updated to reflect your changes:



> **Note:**
>
> If you make changes that the mapper does not recognize and attempt to click **Designer**, the tab is disabled and a message is displayed indicating that the change is not supported. You must resolve those issues before you can return to the designer.

3. Click **Validate** or **Close** (which also performs validation) to exit the mapper, then click **Apply** to save your changes when prompted.

Code Validation

As you make updates to the XSLT code, your changes are validated. Validation safeguards prevent you from returning to the designer if there are errors. For example:

- Typographical errors:
  1. Assume you make a typographical error such as forgetting the `f` at the end of `xsl:value o`.

```
<nstrgdfl:Welcome xml:id="id_19">
      <xsl:value-o select="$tracking_var_3" xml:id="id_18"/>
</nstrgdfl:Welcome>
```

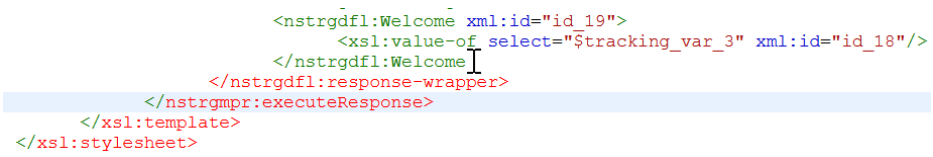
2. Click **Designer**. An error message is displayed in the banner and you are prevented from returning to the designer until you fix the error.

   ```
   Unknown: xsl:value-o
   ```

3. Correct the error and click **Designer** to return to the designer.

- XML syntax errors:

1. Assume you make an XML syntax error such as forgetting to enter a closing bracket (>) after `Welcome` in the following code. The subsequent code is highlighted in red to indicate an error.



2. Click **Designer**  . An error message is displayed in the banner and you are prevented from returning to the designer or accessing other tabs such as **Test**  until you fix the error.

   ```
   There are syntax errors in the edited code. Please fix them
   before navigating away from the tab
   ```

3. Correct the error and click **Designer**  to return to the designer.

- Unsupported constructs in the mapper (for example, you import XSLT code into the mapper that includes unsupported functions):

1. Open the mapper in the integration canvas.

   The **Code**  tab opens by default with the following message.

   ```
   This map cannot be loaded in graphical view.
   ```

2. Expand the number to the left of the error for details.
   These are coding patterns and constructors that are not supported in the designer. The designer does not know how to render or manage them. When the XSTL code includes unsupported patterns or constructors, you cannot navigate to the designer.

Test Your Mappings

Once you complete designing your mappings, you can test them by entering sample content of the message to process in the test mapper. When you execute the test, output is generated from the sample content.

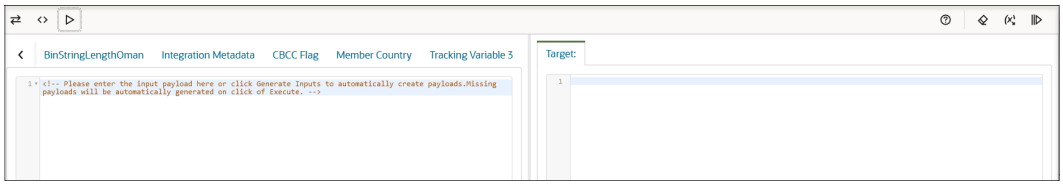
To test a mapping:

> **Note:**
>
> Custom JavaScript functions cannot be tested.

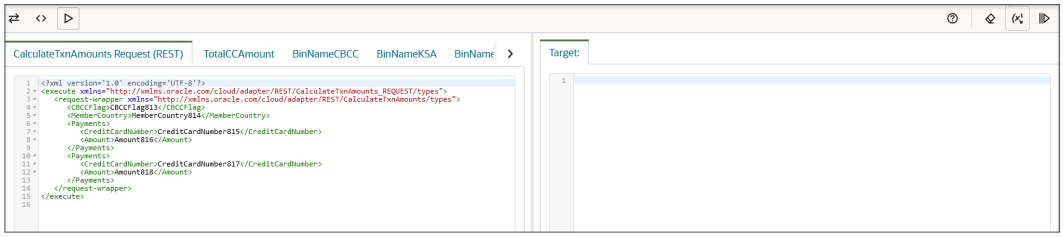
Two elements are generated and displayed for repeating nodes. This is useful for generating payloads containing at least two repeating elements to test for-each loops in the mapper. Instances for tracking variables and local variables in assign actions are also displayed.

1. In the mapper toolbar, click **Test** .

   The mapping tester is displayed with the defined elements.



2. In the **Input** panel, you can manually enter the payload, copy and paste the payload, or click **Generate Inputs**
   (x)
   to automatically generate and test the payload. Payloads for scalar parameters are not created.



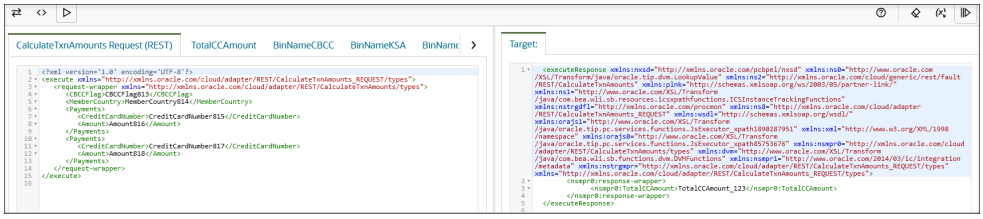
   If your mapping includes multiple source data structures, both names are displayed. Payloads for both sources can be generated.

> **Note:**
>
> If the payload is very large, it is not automatically generated and you receive the following error message:
>
> ```
> Payload could not be generated for the
> ''$SourceApplicationObject'' schema due to excessive size and a
> lack of system memory
> ```

3. Scroll through the input payload and note the following details:
   - Unbounded, repeating elements are displayed multiple times.
   - Schemas of up to 20 levels in depth can be displayed.

- Random values are automatically generated for payload elements. Based on the data type of the element, the correct values (for example, numerical or string values) are generated.
- You can manually edit the randomly-generated values, as necessary.

4. Click **Execute** to generate results in the **Target** panel.
5. Review the results in the **Target** panel to ensure that your input payload was processed correctly.



6. Test your mapping and, as necessary, return to the mapper to make mapping changes, such as changing the XSLT statements or functions used.
7. To clear the **Input** and **Target** panels, click **Clear** .
8. When testing is complete, click **Designer** to return to the mapper.

   Multiple entries are generated for template parameters. There should be only be one instance of each template parameter. This is the expected behavior.

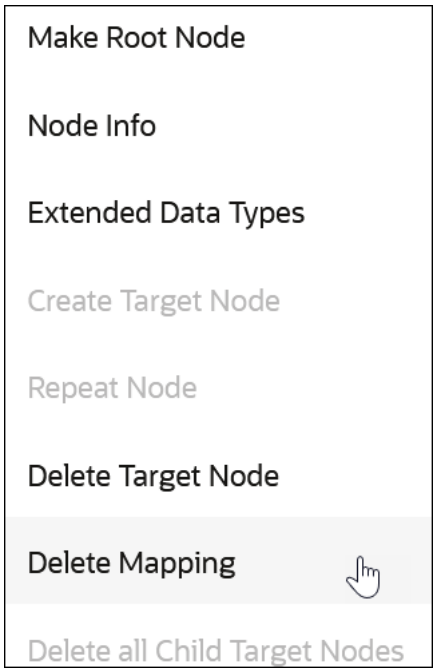
Delete Mappings and Target Element Nodes

You can delete mappings and target element nodes from a context menu. You can select this option for a parent to delete all children. For example, if you select the root, all mappings are deleted.

Deleting Mappings

To delete mappings:

1. Find the source-to-target mapping to delete.
2. Right-click the target element node name, and select **Delete Mapping**.



Deleting Target Nodes

To delete target nodes:

1. Find the source-to-target mapping.
2. Right-click the target element node name to delete, and select **Delete Target Node**.

   This action deletes the mapping *and* the target element. The element node is now grayed out (considered a ghost node). If you click **Code** <> and view the XSLT file of the mapping, note that this element does not exist. However, you can still map to it.
3. If you want to create this target element node, select **Create Target Node** to create it again in the XSLT file. As a short cut, you can also create a target element node by simply dragging a source element node to it.

> **Note:**
>
> - If you delete a parent element node, all of its child element nodes and any of their mappings are also deleted.
> - If you drag an XSLT statement to a target element node, the node must already exist (cannot be a ghost node). In those cases, you must first right-click the target element node and select **Create Target Node**.

Troubleshoot Errors

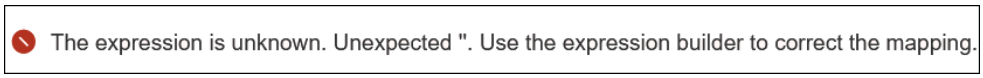
Mappings may contain errors. Resolve the errors, and you can activate your integration. Errors become visible at different times, such as after you click **Validate** during mapping design. Or, you might complete a mapping without errors and then change the overall

integration, such as by regenerating a WSDL. When you return to the mapper, the errors are visible.

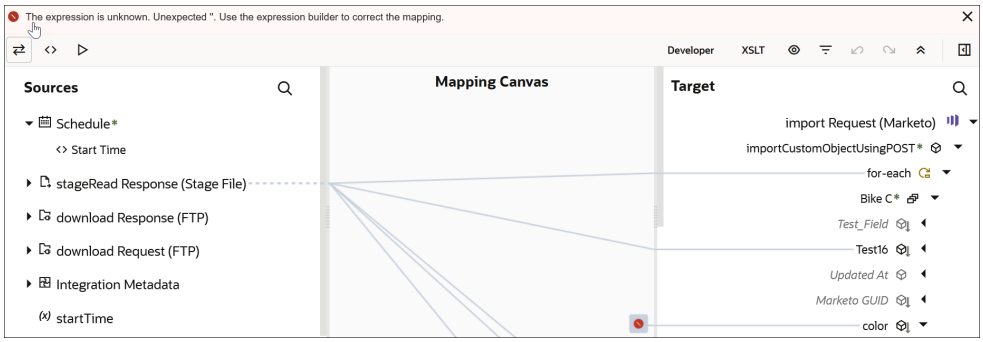
Error messages are identified by red icons and warning messages are identified by yellow icons above the **Sources** section of the mapper.

To troubleshoot errors:

1. Expand the numbers in the red and yellow icons to show additional messages.



2. Click the message to access the error or warning in your mappings.



   For this example, there are two invalid target errors. The targets are in the XSLT file, but not in the schema. This may have occurred because the WSDL was regenerated after you previously completed mapping.

   When adding functions to your mappings, you can also receive errors if you do not enter all the parameters in the Expression Builder. For example, you add a **concat** function to your mapping, but forget to add one or both parameters to the function.

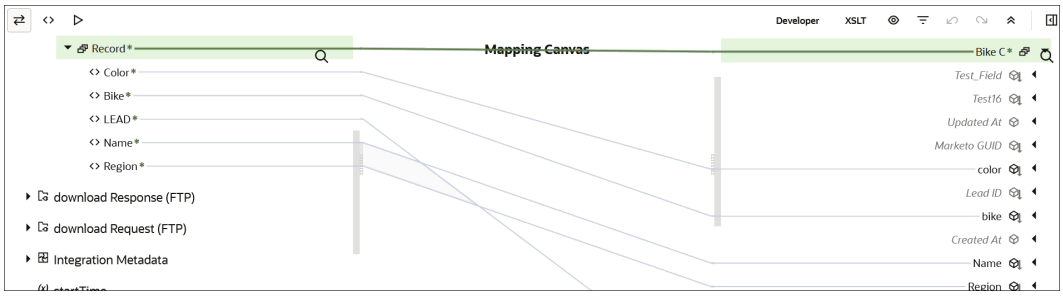
3. To show only the mappings with errors and warnings, click **Filter** and select **Errors** and **Warnings**. See Filter the Source or Target Data Structures.

Repeat a Target Element to Map to Different Sources

You can repeat a target element in the mapper. This enables you to map different sources to the same target element. Elements defined in the target schema with the `maxOccurs` attribute set to a value greater than one can be repeated.

To repeat a target element to map to different sources:

1. In the target data structure, right-click the element node to repeat, and select **Repeat Node**. This option is only available on elements that you can repeat. Elements that can be repeated are identified by a special icon to the right of the name: .



The element is repeated.

2. Expand the existing and repeated elements to see that the attributes in each element are repeated.

3. Drag appropriate source mappings to the repeated targets.

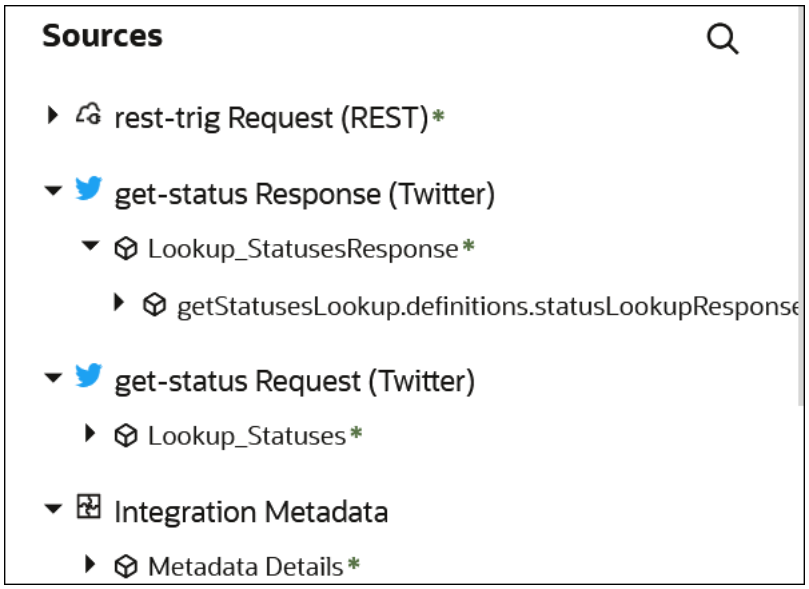
> **Note:**
>
> If you create a repeatable element in which you do not do any mapping, click **Close**, and apply your changes when prompted, the empty element is not saved.

Map Multiple Source Structures to a Target Structure

You can map fields from multiple source structures to a single target structure in certain parts of integrations (for example, integrations in which message enrichment points have been added or integrations with a response mapping). This action applies to the creation of new maps.

To map multiple source structures to a target structure:

1. In the mapper, note that two source structures are displayed:
   - The initial request mapping source (for this example, **process**)
   - The secondary request (for this example, **$TargetApplicationObject1**)



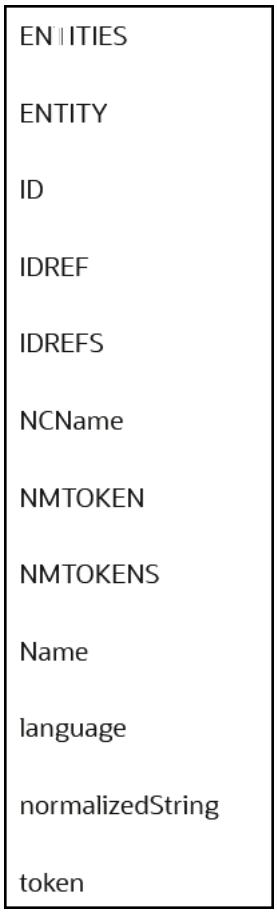
2. Expand the initial source data structure and drag appropriate source element nodes to target element nodes.
3. Expand the secondary source data structure and drag appropriate source element nodes to target element nodes.
4. To test the mappings, see Test Your Mappings.
5. When complete, click **Close**, then apply your changes when prompted.

Extend a Data Type

You can extend a data type in the mapper. An extended data type is a primitive data type or container with a supplementary name and some additional properties. Extended data types are user-defined types based on the primitive data types boolean, integer, real, string, and date, and the composite type container.

1. Right click a target element and select **Extended Data Types**.

2. From the **Ext Datatypes** list, select the data type to extend.



Import a Map File into an Orchestrated Integration

You can import an XSL map file that was previously exported from the *same* integration. This action overwrites the existing mapping file.

For example, you can export the map from a specific integration, edit the XSL file as per a user requirement, save it, and import it back into the same integration. You cannot import an XSL map file into an orchestrated integration that was exported from a different integration in Oracle Integration or from an application in Oracle JDeveloper.

1. Click the map that you want to import into an integration.
2. Click **Actions** ⋮, then select **Import**.
3. Browse for the map file to import, then click **Import**. You only import the map file of an exported integration into Oracle Integration. You do *not* import the entire integration in which the map file is included into Oracle Integration.

3

Work with Functions, Operators, and XSLT Statements

You can add functions, operators, and XSLT statements to your mappings.

Topics

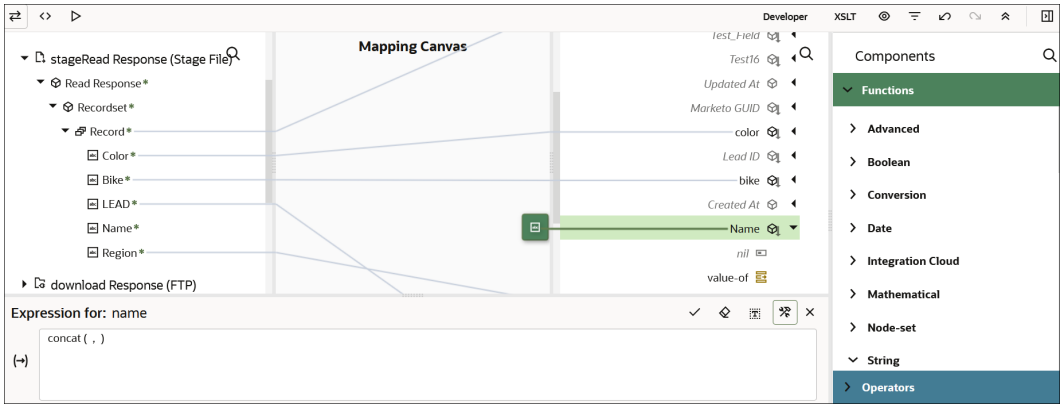
Add Functions, Operators, and XSLT Statements

You can add functions, operators, and XSLT statements to your mappings.

Working with Functions

1. In the **Target** section, highlight the element node to which to connect.
2. In the upper right corner, click **Toggle functions**  to launch the **Components** panel.
3. Expand **Functions**.
4. Select a function. For this example, **String** is expanded and **concat** is dragged to the target element node. The element can be an existing or ghost (not yet created) element.

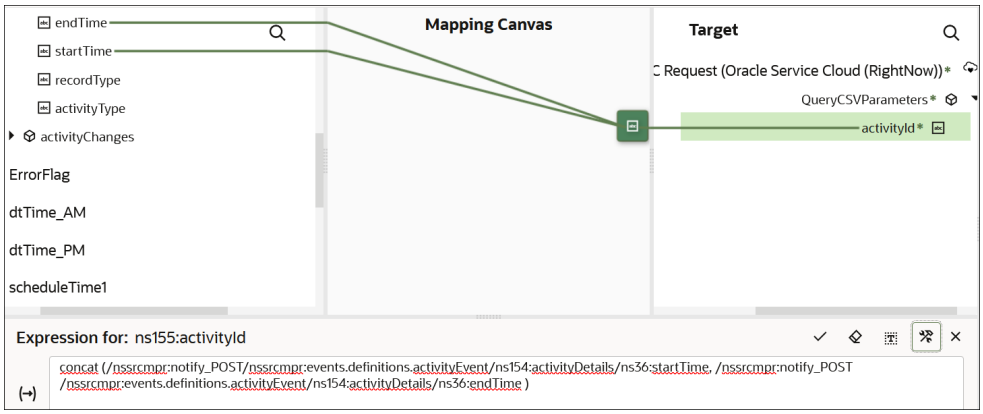
   A **function** icon is added to the **Mapping Canvas** section for the target element node and the function XPath expression is added to the Expression Builder at the bottom of the page. This icon indicates that a function is used in this mapping.



> **Note:**
>
> You can also initially drag functions to the Expression Builder and then connect the source element(s) to the function.

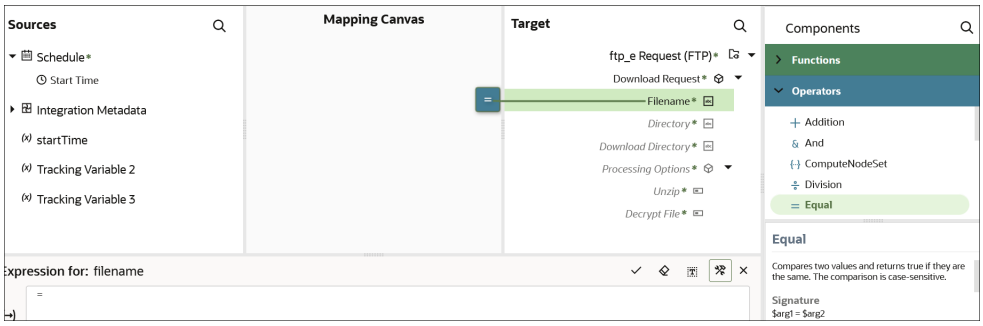
5. In the **Sources** section, drag the source element nodes to the function in the Expression Builder. For this example, **startTime** and **endTime** are dragged to the two sides of the comma in the **concat(,)** function in the Expression Builder. Do *not* drag source element nodes to the **function** icon in the **Mapping Canvas** section.

6. Click **Save** to save your updates.



Working with Operators

1. Expand the **Operators** section.
2. Drag an operator to the target element node (for this example, a **=** is added). The **=** operator is also added to the Expression Builder. The element node can be a created or ghost element node.



3. Drag appropriate source elements to both sides of the operator or manually enter values.

4. Click **Save** to save the operation.

The operator icon is displayed in the **Mapping Canvas**.

Working with XSLT Statements

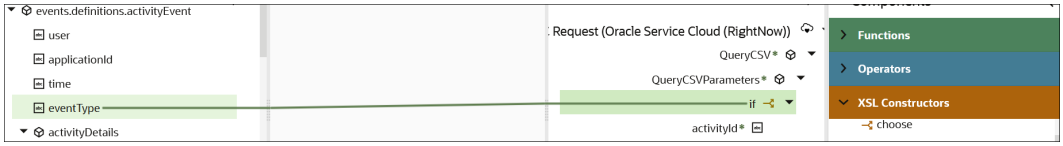
1. Click **Toggle Functions**  .
2. Click **XSLT**.

   An **XSLT Constructors** header is added to the **Components** panel.
3. Expand **XSLT Constructors**.
4. Browse for and drag the appropriate XSLT statement onto the target element node or use the search facility to manually enter and search for the XSLT statement.

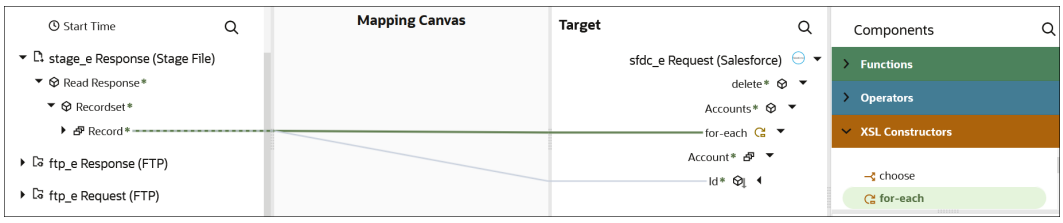
   Note the following conventions:

   - You can drag statements onto parent or child elements. Note the following conventions about dragging XSLT statements:
     - A green icon is displayed when you drag the XSLT statement to the front or the back of the element.
     - If a green icon is not displayed, you cannot insert as a parent.
     - Drag the statement to the end of the name to insert it as a parent.
     - Drag the statement to the front of the name to insert it as a child.
   - You can only drag XSLT statements onto created elements. If the element on which you want to drag the statement is grayed out (is a ghost node), right-click the element and select **Create Target Node**.

   For example, drag an **if** statement to the target element, then map a source element to the target element.



Or drag a **for-each** statement to a repeatable element.



Get Online Help for Functions, Operators, and XSLT Constructs

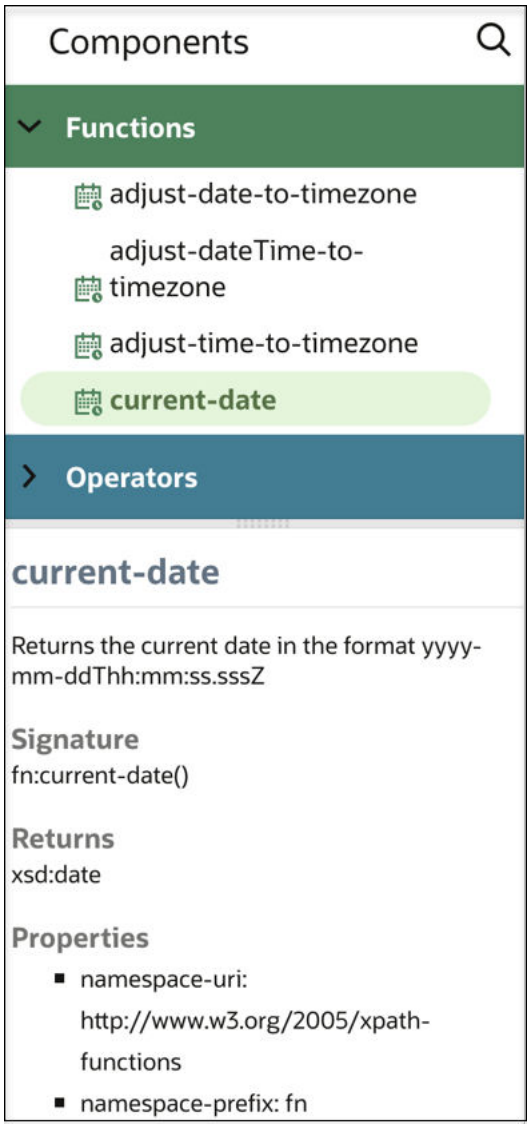
Embedded online help is provided for functions, operators, and XSLT constructs in the mapper. Function help provides a description, signature, parameter, what the function returns, and supported properties. Operator help provides a description, signature,

parameters, what the operator returns, and examples. XSLT construct help provides a description, signature, and examples.

The following example describes how to obtain online help for a function.

1. In the upper right corner, click **Functions**  to launch the **Components** panel.
2. Expand **Functions**.
3. Select a function. For this example, **current-date** is selected.

   Online help opens below the function.

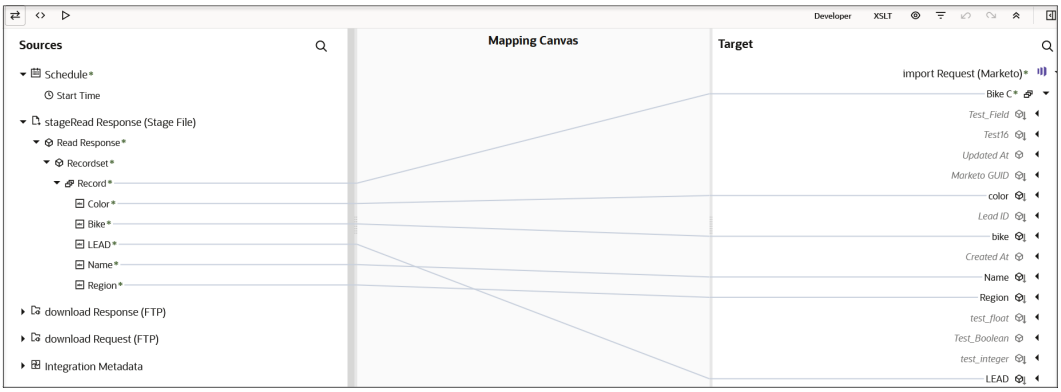


Automatically Create for-each Statements

You can automatically create for-each statements when mapping between repeatable source and target elements in the mapper.

To automatically create for-each statements:

1. In the **Source** section, identify the repeatable source and target elements to which to map. Repeatable elements are identified by the  icon to the left of the name. When you right-click these elements and select **Node Info**, **Repeating: true** is displayed in the message details about the element.
2. In the **Source** section, map the child repeatable element to the child target repeatable element. You cannot map repeatable elements to nonrepeatable elements.



The mapper creates a for-each statement to loop through the source **Relationship** element and place the mapping into the target **Organization** element. This statement does not include a value to select because parent elements do not typically contain attributes to map.

3. Click **Code** to view the for-each statement.

```
<xsl:for-each xml:id="id_37" select="$stageRead/nsmpr2:ReadResponse/
ns21:recordset/ns21:record">
   <nstrgmpr:bike_c xml:id="id_38">
         <ns23:color xml:id="id_39">
           <xsl:value-of xml:id="id_40" select="ns21:color"/>
         </ns23:color>
         <ns23:bike xml:id="id_41">
            <xsl:value-of xml:id="id_42" select="ns21:bike"/>
         </ns23:bike>
         <ns23:name xml:id="id_43">
            <xsl:value-of xml:id="id_44" select="ns21:name"/>
         </ns23:name>
         <ns23:region xml:id="id_47">
            <xsl:value-of xml:id="id_48" select="ns21:region"/>
         </ns23:region>
         <ns23:lEAD xml:id="id_45">
            <xsl:value-of xml:id="id_46" select="ns21:lEAD"/>
         </ns23:lEAD>
      </nstrgmpr:bike_c>
   </xsl:for-each>
```

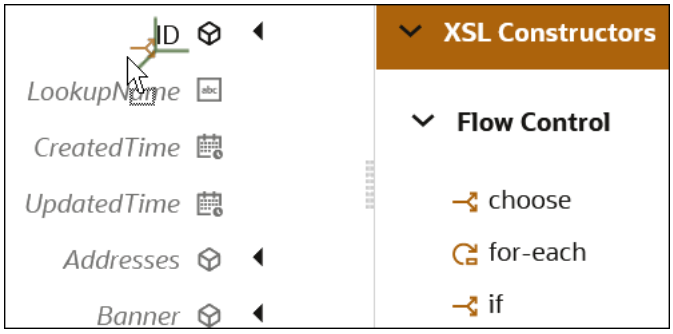
Create Conditional Mappings

The if and choose statements are two ways to create conditions. If statements allow you to specify a single condition. Choose/when/otherwise statements allow you to specify multiple conditions, similar to if/then/else.

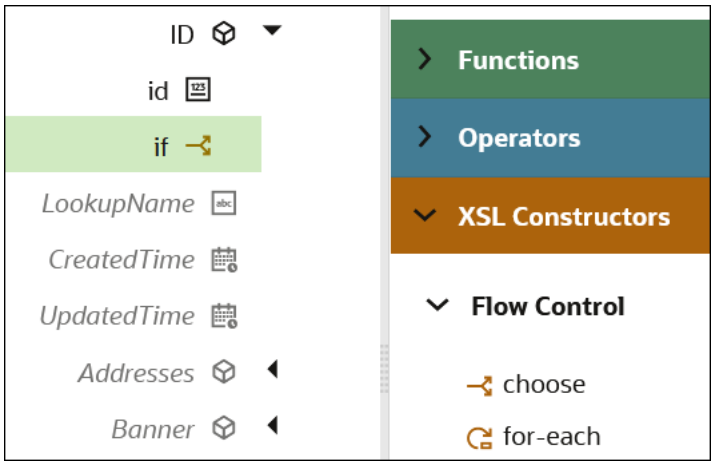
To create conditional mapping:

1. Drag a source to a target to create a mapping.
2. Click **View** , then select **Advanced**.
3. In the upper right corner, click **Toggle functions** to launch the **Components** panel.
4. Expand **XSLT**, and drag appropriate XSLT statements onto the target element.

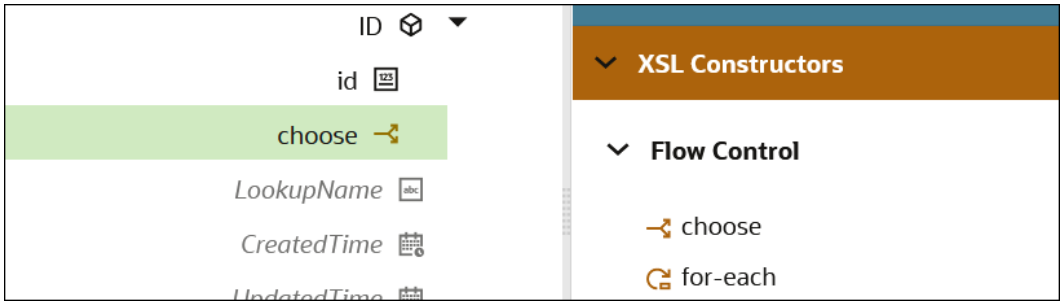
   You can either search or browse for the function.
5. Drag the **if** or **choose** function onto the target element. (for this example, an **if** statement is dragged to an ID element).



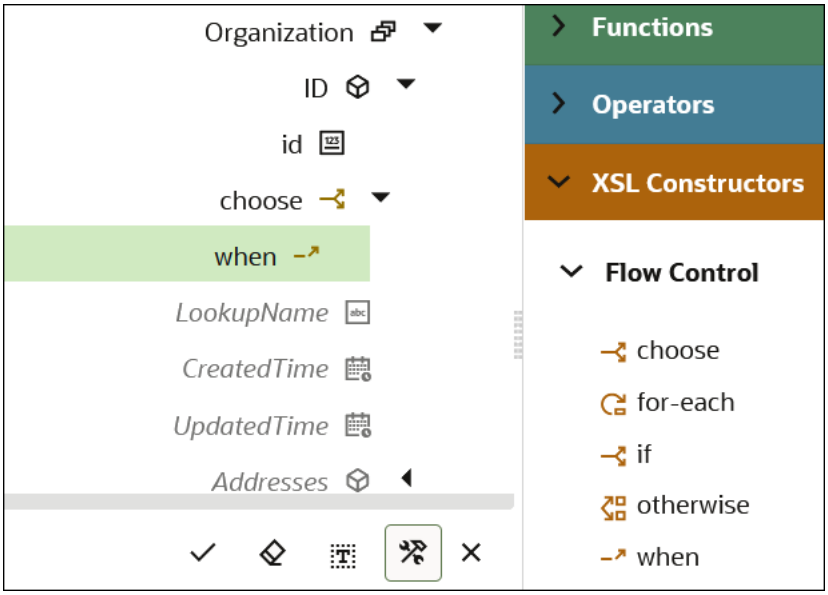
The **if** statement is displayed.



6. If you add a **choose** statement, you may specify additional **when** and **otherwise** conditions.



- Highlight the **choose** action, then drag and drop a **when** or **otherwise** statement.



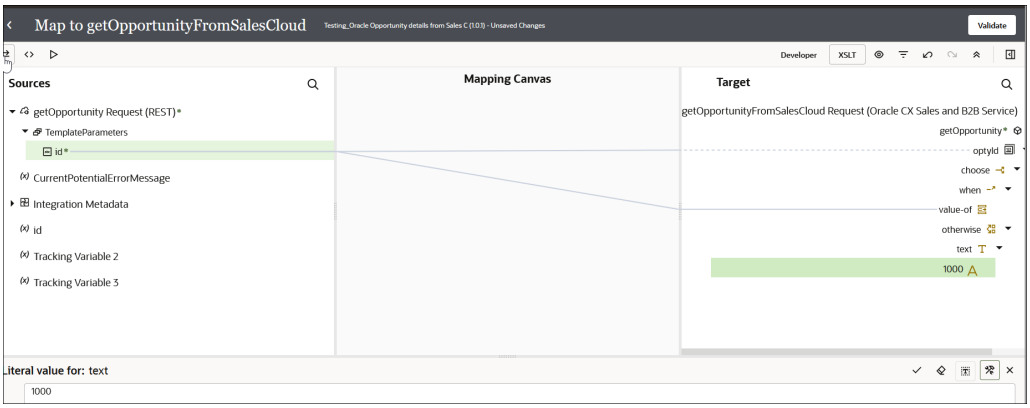
7. Click **<**, then apply your changes when prompted.

See Use Conditional Mappings.

Set Default Values in the Mapper

You may have scenarios in which you need to set some fields to default values. The mapper contains a set of functions that you can use to set default values (for example, the **when** function that you can use to set default values).

For example, the following conditional mapping is performed.



In the payload, you can set the default value in the mapper.

```
<nstrgmpr:getOpportunity xml:id="id_12">
      <nstrgmpr:optyId xml:id="id_16">
            <xsl:value-of select="/nssrcmpr:execute/nssrcmpr:TemplateParameters/nsmpr1:id" xml:id="id_17"/>
            <xsl:choose>
                  <xsl:when test="">
                        <xsl:value-of select="/nssrcmpr:execute/nssrcmpr:TemplateParameters/nsmpr1:id"/>
                  </xsl:when>
                  <xsl:otherwise>
                        <xsl:text>1000</xsl:text>
                  </xsl:otherwise>
            </xsl:choose>
      </nstrgmpr:optyId>
</nstrgmpr:getOpportunity>
```

This syntax checks if the **iD** node is present in the payload. If so, it assigns that value. Otherwise, it adds the default value, which in this case is **1000**.

Create the lookupValue Function

You can create the parameter values for the `lookupValue` function with the Map Lookup Value wizard. This wizard enables you to define the lookup table, source column, target column, and default value to use in the function. For these parameter values to be selectable in the wizard, you must have already created a lookup on the Lookups page.

Topics:

- Access the Build Lookup Function Wizard
- Select the Lookup
- Select the Source and Target Columns
- Specify the Default Value
- Review Your Lookup Table Selections

Access the Build Lookup Function Wizard

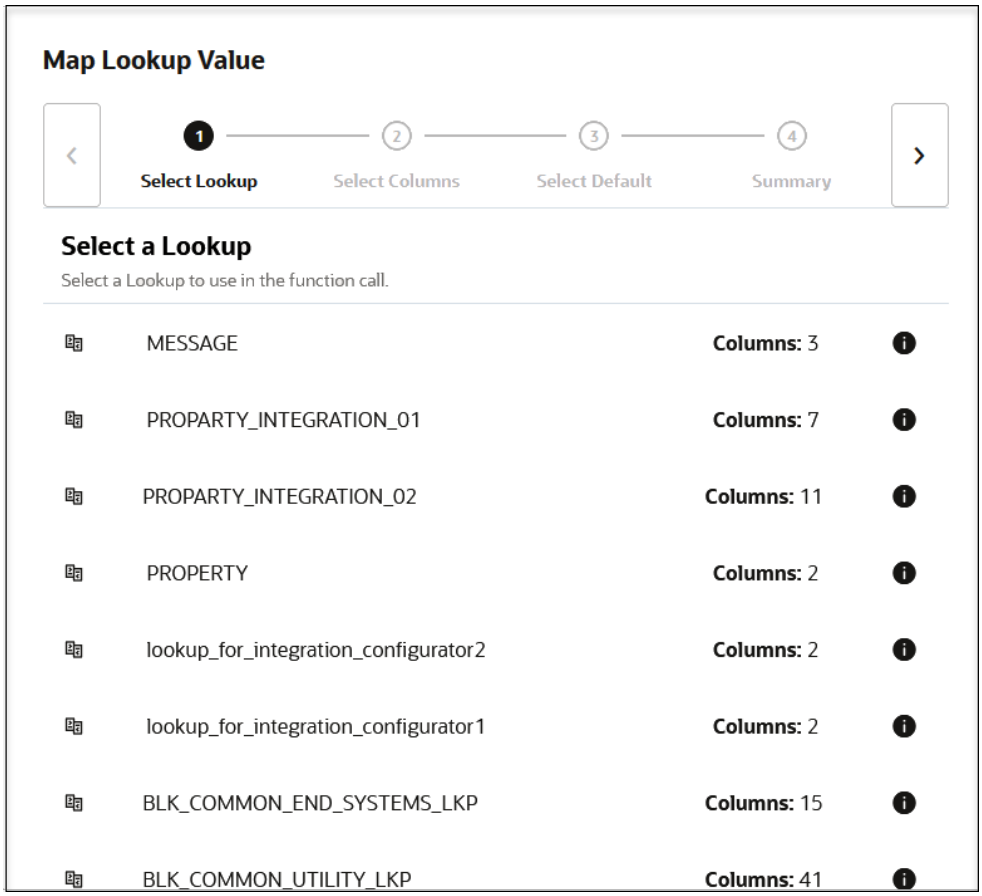
The Map Lookup Value wizard for creating the `lookupValue` function parameter values is accessible from the mapper or the Expression Builder in actions that support functions in Oracle Integration.

> **Note:**
>
> You must already have created lookups to use this wizard.

- From the Mapper
- From the Expression Builder In an Action that Supports Functions

From the Mapper

1. In the upper right corner, click **Toggle functions**  to launch the **Components** panel.
2. Type `lookupValue` in the **Search** field, and click **Search**.
3. Drag the function onto the target element node.
   The Map Lookup Value wizard is displayed.



From the Expression Builder in an Action that Supports Functions

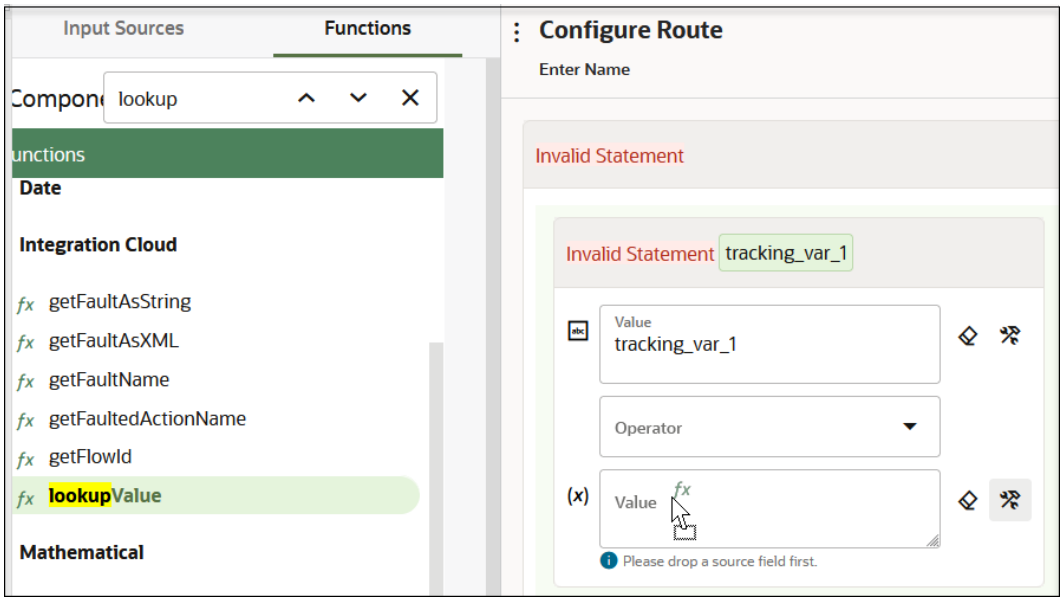
You can access the Map Lookup Value wizard in the Expression Builder in an action that supports functions, such as a switch, stitch, or others.

1. Open an integration for editing.
2. Open an action that supports functions in the integration (for this example, a switch action is shown, but other actions such as a stitch can be used).

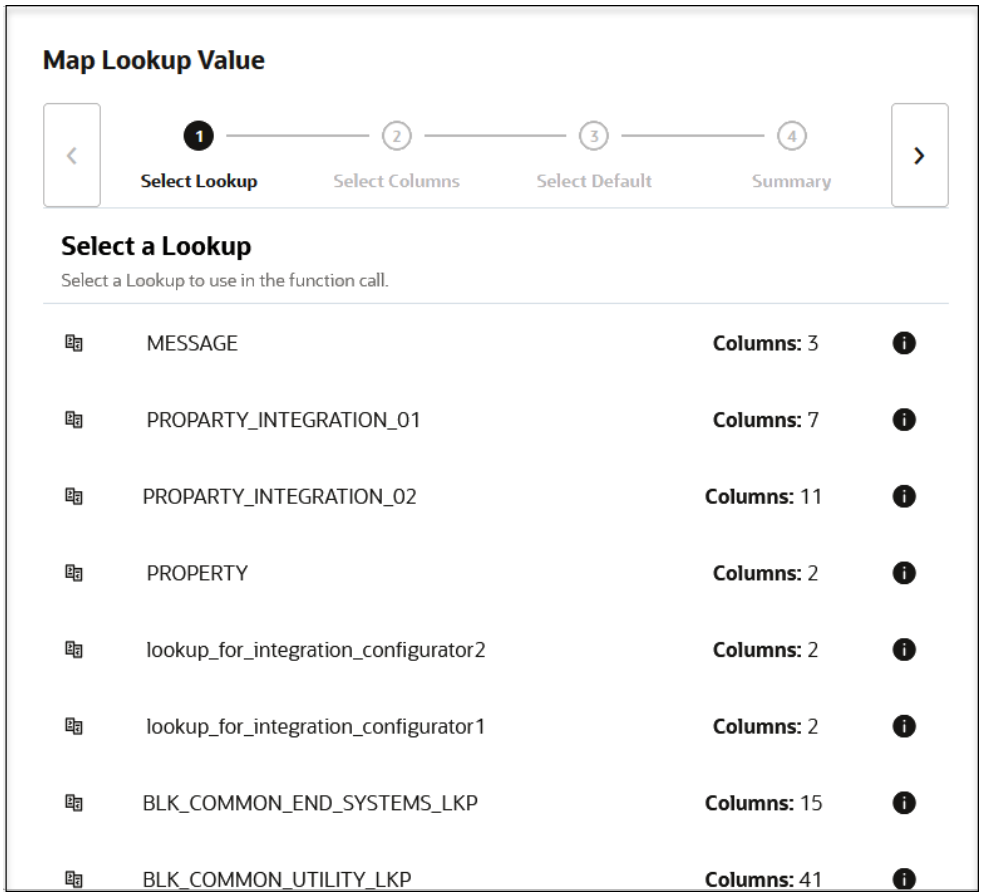
   The Configure Route panel opens.
3. Manually enter a value, select a value from the drop-down list, or drag a value from the **Sources** tree to the first **Value** field. For this example, **tracking_var_1** is selected.
4. Select an operator in the **Operator** field.
5. Click **Functions**, and browse or search for `lookupValue`.
6. Click `lookupValue`.
7. View the description below the function for information about the supported parameters. You define values for these parameters in the Map Lookup Value wizard.
   - `dvm-location`
   - `src-column`

- `src-value`
- `target-column`
- `default-value`

8. Click **Switch to Developer Mode** in the second **Value** field.
9. Drag the `lookupValue` function into the second **Value** field.



The Map Lookup Value wizard is displayed.



Create the `lookupValue` function parameter values.

Select the Lookup

You can select the lookup to use in the `lookupValue` function. You must already have created a lookup. Otherwise, no lookups are displayed for selection.

1. Select the lookup to use in the function. You can view the lookup description by clicking the information icon in the row. This can guide you in selecting the required lookup. The number of columns defined in the lookup is also displayed.
2. Click **>**.

Select the Source and Target Columns

You can select the source and target columns to use in the `lookupValue` function. The `lookupValue` function requires one source column and one target column. When you select a source and target column, the values available with the columns are displayed.

1. Select the source and target columns to use in the `lookupValue` function.

Element	Description
Select Source	Click the source column header to select from a list of available columns for this lookup table. The data included with the selected column is displayed. Both adapter and domain name columns are displayed.
Select Target	Click the target column header to select from a list of available columns for this lookup table. The data included with the selected column is displayed. Both adapter and domain name columns are displayed.

2. Click **>**.

Specify the Default Value

You must specify a default value to use.

1. Enter the default value to use in the `lookupValue` function if no match is found. If there is no match that satisfies all the search values, the lookup fails and the default value is returned (for example, an actual default value to use or an error message such as `No Value Found`).
2. Click **>**.

Review Your Lookup Table Selections

You can review the lookup table values to use in the `lookupValue` function on the Summary page.

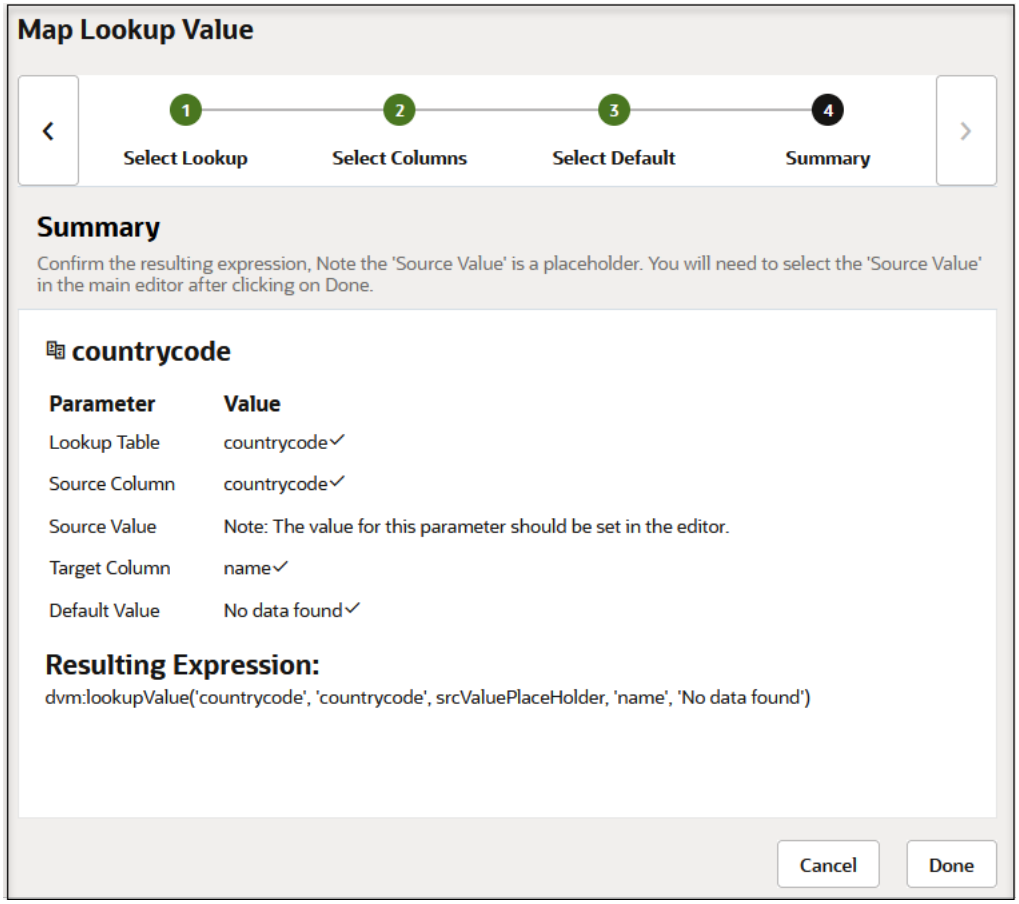
1. Review the lookup table values. The Summary page is the final wizard page after you have completed your configuration.

Element	Description
Parameter and Value Table	Displays a summary of the parameters and values you defined on previous pages of the wizard. To return to a previous page to update any values, click the appropriate tab in the left panel or click **Back** .

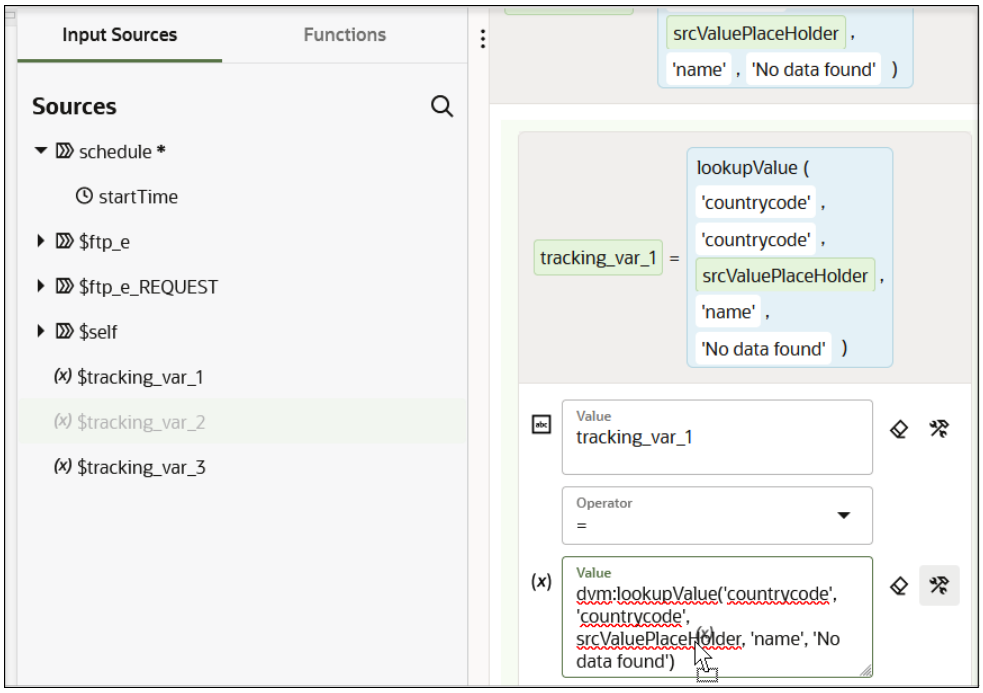
Element	Description

Resulting Expression	Displays the expression you defined on the previous pages of the wizard. The `lookupValue` function takes the following format:
	
	`lookupValue(dvm-location, src-column, src-value, target-column, default-value)`
	
	Where:
	• **Lookup Table**: The lookup table selected on the Select Lookup page.
	• **Source Column**: The source column selected on the Select Columns page.
	• **Source Value**: The source value you enter in the mapper or Expression Builder after completing this wizard. Complete this wizard, then define the `srcValue` parameter value.
	• **Target Column**: The target column selected on the Select Columns page.
	• **Default Value**: The default value entered on the Default Value page.
	For example, a defined `lookupValue` function after you have completed the wizard and defined the `srcValue` parameter value in the Expression Builder can look as follows:
	
	`dvm:lookupValue('countrycode', 'countrycode', srcValuePlaceHolder, 'name', 'No data found')`

For example:



2. Click **Done**.
3. In the **Sources** tree, drag a value to replace **srcValuePlaceHolder** in the **Value** field.



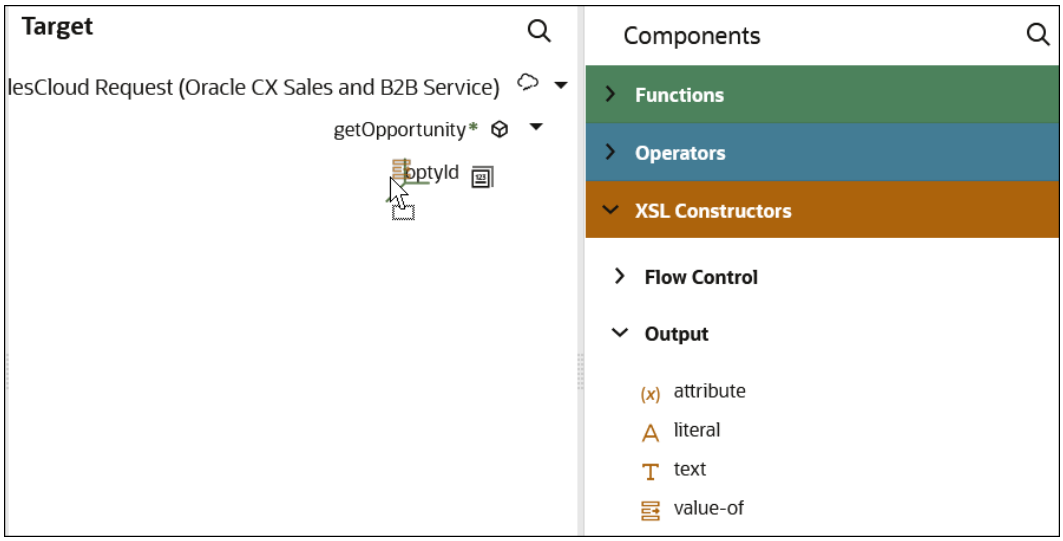
4. Click **Save**.
   You can use the configured function in the mapper. See Add Functions, Operators, and XSLT Statements in *Using the Oracle Mapper with Oracle Integration 3*.

Work with Multiple Value Statements

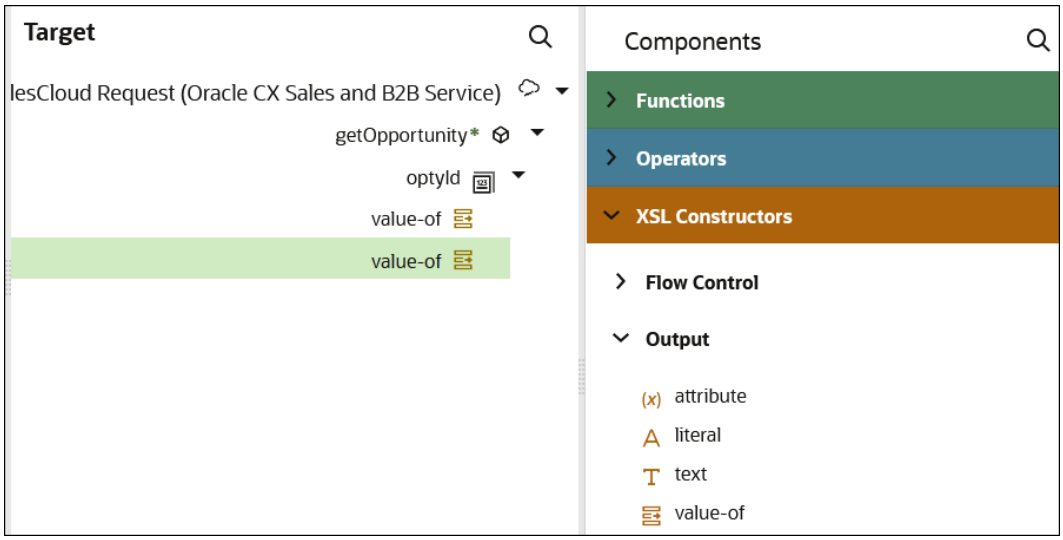
You can add multiple value-of statements or multiple XSLT statements under a leaf node.

To work with multiple value statements:

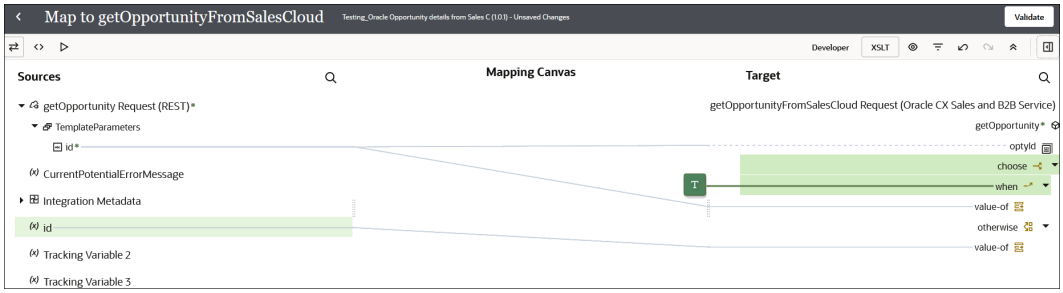
1. Click **Toggle functions** .
2. Click **XSLT**.
3. Drag a **value-of** statement to a leaf element target in the mapper. For this example, **value-of** is added as a child of **optyId**.



Multiple **value-of** statements are added to the leaf node.



4. Define appropriate mapping logic for each **value-of** statement. For example, add a **choose** statement and a **when** statement with a defined value to the first **value-of** statement and an **otherwise** statement to the second **value-of** statement.



> **Note:**
>
> Multiple **value-of** XSLT statements in a leaf node continue to remain visible in the mapper even if you disable **XSLT**.

4

Mapper Use Cases

Learn about use cases with the mapper.

Topics:

Convert an Integer to a String

You can use convert an integer to a string in the mapper.

1. Open the mapper.
2. In the **Target** section, highlight the integer element node to convert to a string.
3. In the upper far right corner, click **Toggle functions**  to open the **Components** panel.
4. Expand **Functions** and then **Conversion**.
5. Select the **string** function and drag it to the target element node.

   A function icon is added to the mapping canvas for the target element node and the **string** function XPath expression is added to the Expression Builder at the bottom of the page.
6. Drag the source integer node (or you can use a literal) inside the parenthesis and click the **Save**  icon to save.

   For example:

   ```
   string (12345)
   ```
7. Click **Validate**.
8. Optional: To see how the XPath function performs at runtime, click **Test**  , then click **Generate Inputs**

, and then click **Execute** .

Use Conditional Mappings

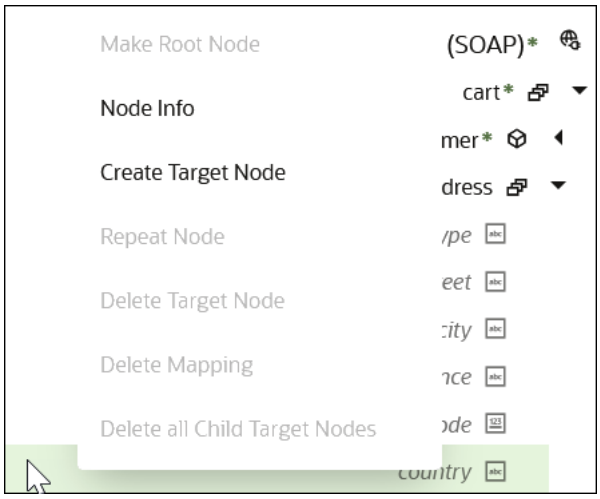
You may have a requirement to map data dynamically depending on other data in your integration. This requirement can be achieved with conditional mappings.

This use case describes how to create mappings that specify conditions to take based one of the following countries. If none of those countries are specified, an otherwise condition is taken.

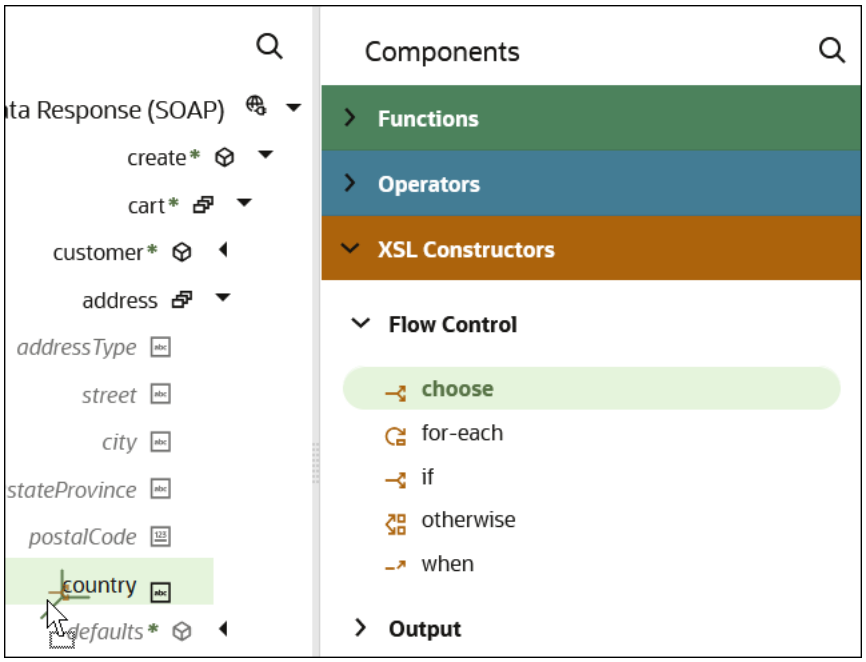
- United States of America
- India
- United Kingdom

1. Click **XSLT**.
2. In the upper right corner, click **Toggle functions** .
3. Expand **XSL Constructors**, then **Flow Control** to display the constructs required to create conditional mappings.
4. Locate the target element (for this example, named **country**) in the tree.

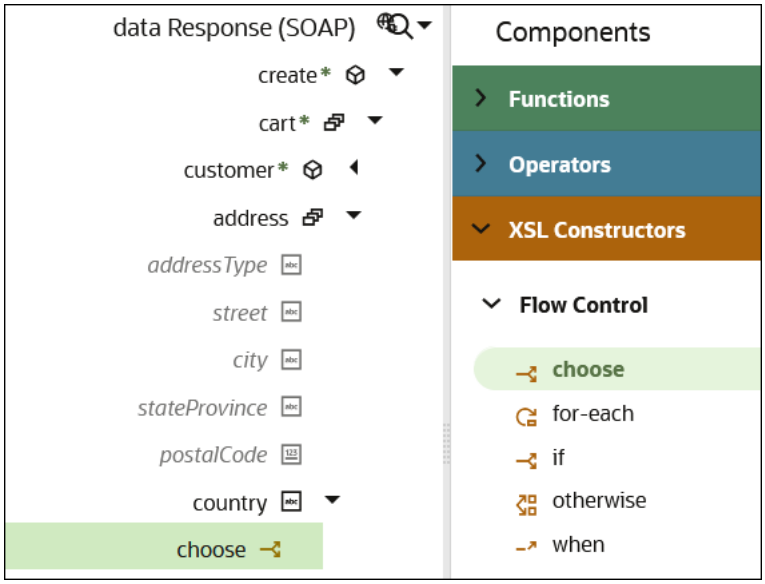
   This is the element for which to create conditional mappings.
5. If the selected element is a lighter color, that means the element does not exist in the mapper's output. Right-click and select **Create Target Node**. You cannot insert conditions around **country** without this step.



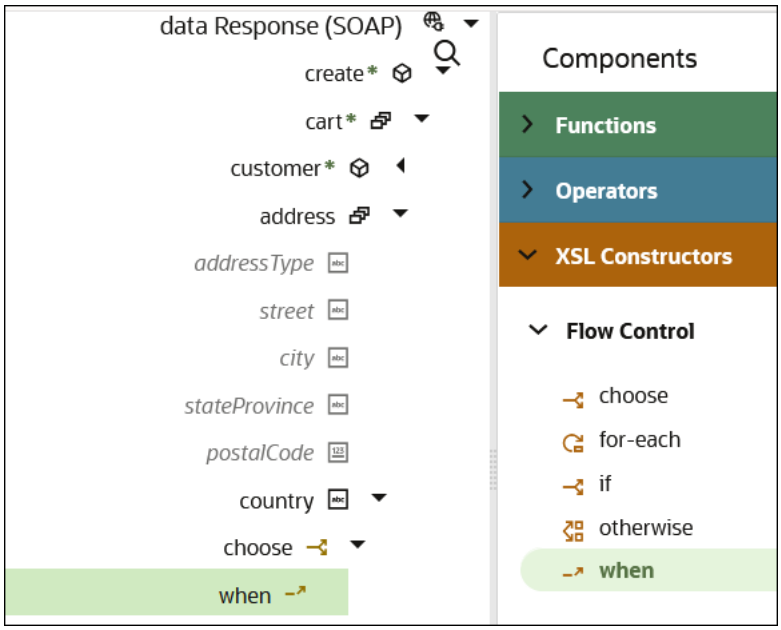
6. Drag and drop the **choose** construct as a child of **country**. The cursor position surrounding **country** indicates whether the **choose** construct can be inserted as a child (bottom left) or a parent (upper right).



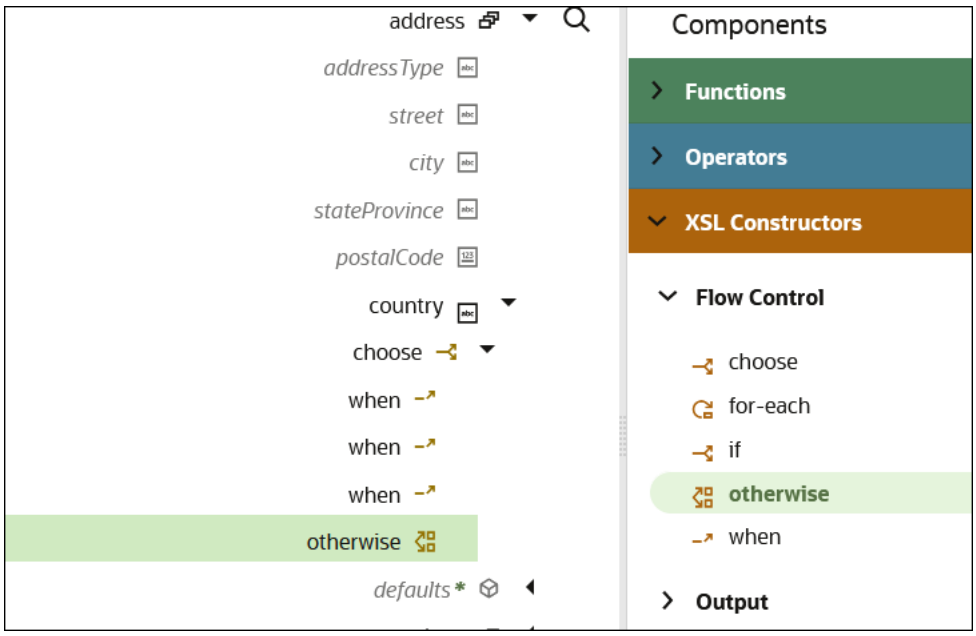
For this case, **choose** is inserted as a child.



7. Drag and drop a **when** construct as a child of the **choose** construct.

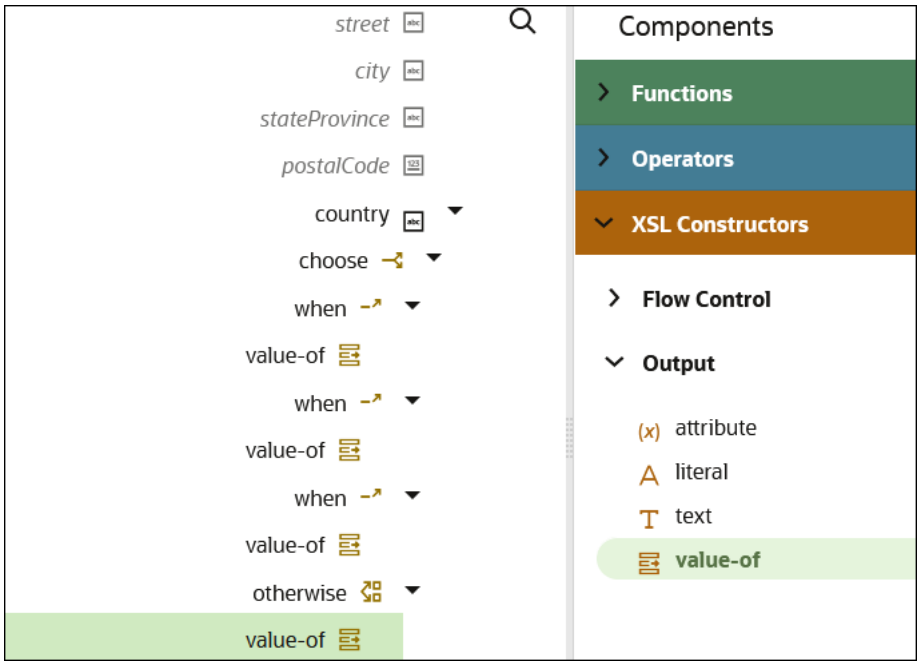


8. Drag and drop two additional **when** constructs as children of the **choose** construct to create placeholders for the three conditions.
9. Drag an **otherwise** construct below the third **when** construct.

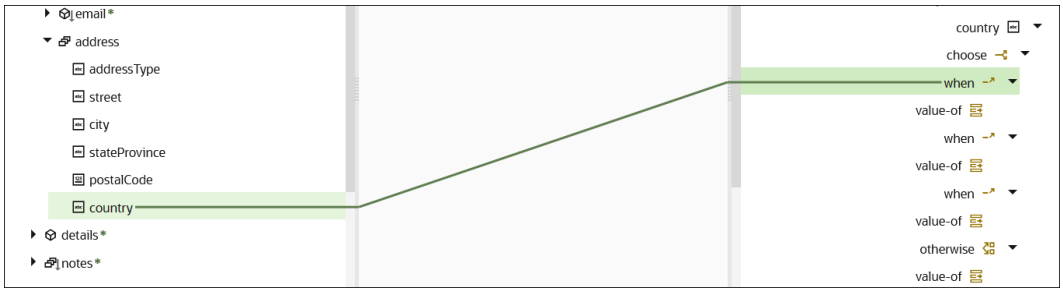


Each condition also requires a corresponding mapping value.

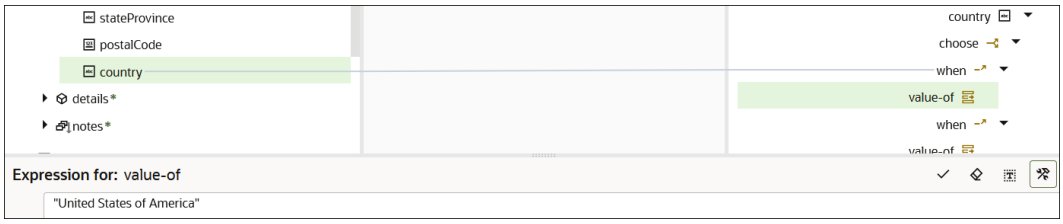
10. Drag and drop a **value-of** construct as a child of each **when** and **otherwise** construct. The tree structure needed to create conditional expressions and mapping expressions is now complete.



11. Select the first **when** construct in the target tree to create the first condition.
12. Map **country** from the source tree to the first **when** construct.



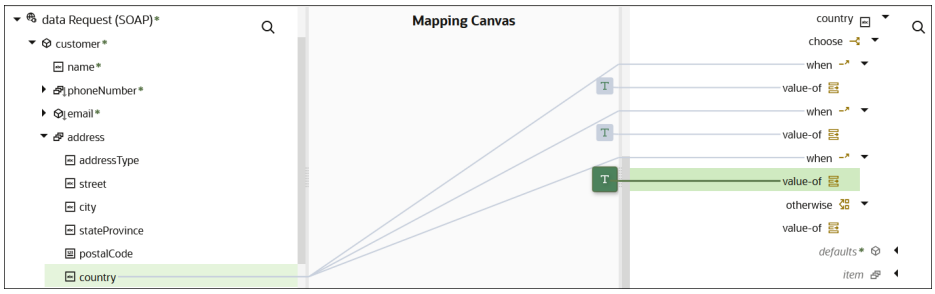
13. Select the **value-of** element for the first **when** construct.
14. Enter = `"United States of America"` in the Expression Builder to complete the expression.



15. Click the **checkmark** to save the expression.

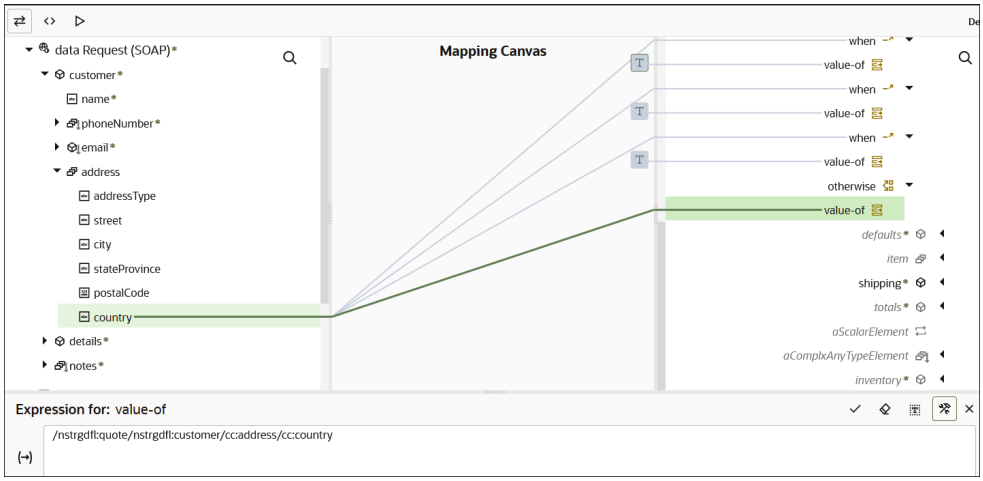
    The text you entered is identified with a **T** for text (or literal) value.
16. Repeat steps 11 through 15 for the second and third **when** constructs.

- Add `"India"` to the **value-of** element of the second **when** construct.
- Add `"United Kingdom"` to the **value-of** element of the third **when** construct.



17. Drag and drop **country** from the source tree to the **value-of** element of the **otherwise** construct.
18. Do not enter a literal value for the **otherwise** construct.

    When complete, the design looks as follows.



19. Click the **checkmark** to save the expression.
20. Click **Validate**.
21. Click **Code**  to view the code for the conditional mapping.

```
<nstrgmpr:address xml:id="id_138">
      <cc:country xml:id="id_139">
            <xsl:choose>
                  <xsl:when test="/nstrgdfl:quote/nstrgdfl:customer/
cc:address/cc:country">
                        <xsl:value-of select="&amp;quot;United States
of America&amp;quot;"/>
                  </xsl:when>
                        <xsl:when test="/nstrgdfl:quote/
nstrgdfl:customer/cc:address/cc:country">
                        <xsl:value-of select="&amp;quot;India&amp;quot;"/>
```

```
                  </xsl:when>
                        <xsl:when test="/nstrgdfl:quote/nstrgdfl:customer/
cc:address/cc:country">
                        <xsl:value-of select="&quot;United
Kingdom&quot;"/>
                  </xsl:when>
                  <xsl:otherwise>
                        <xsl:value-of select="/nstrgdfl:quote/
nstrgdfl:customer/cc:address/cc:country"/>
                  </xsl:otherwise>
               </xsl:choose>
      </cc:country>
</nstrgmpr:address>
```

Iterate Across Groups with a for-each-group Constructor

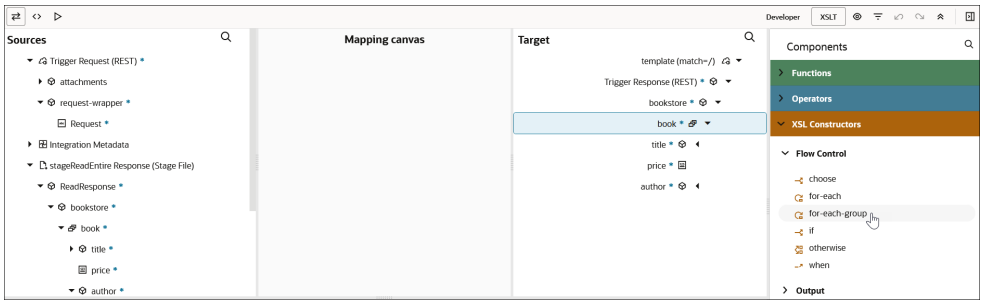
You can iterate across a series of groups with a for-each-group constructor in designer mode in the mapper. The grouping criteria is specified by the attributes. The required `select` attribute identifies the elements to sort and group. The `group-by` attribute describes how to group the elements.

Capabilities

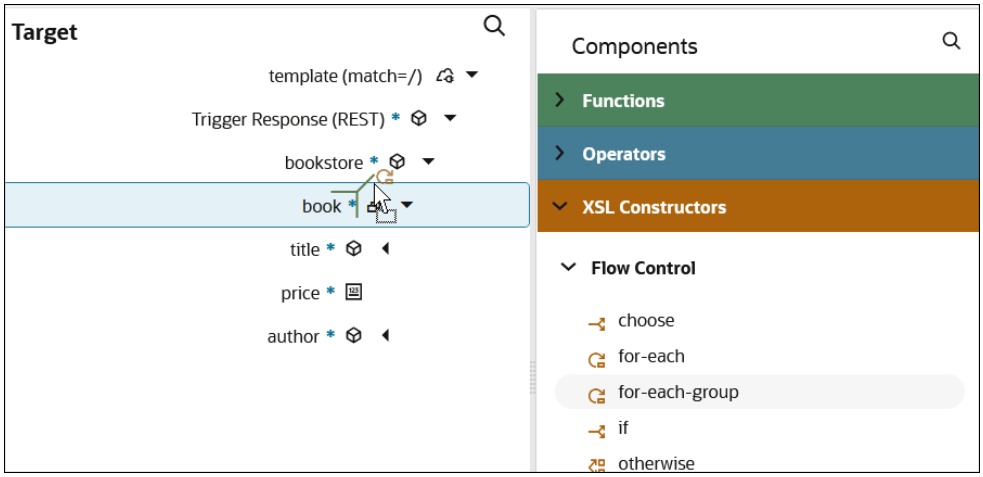
- You can only map repeated source elements to the `for-each-group` node (same as a `for-each` constructor).
- You can only drop the `for-each-group` as a parent in a target tree node (same as a `for-each` constructor).
- When you drop a `for-each-group` on a target node, a child `group-by` attribute node is created.
- The **Delete node** option is disabled for the `group-by` attribute node.
- When you delete a `for-each-group`, the `group-by` attribute node is also deleted.
- If you are using an attribute other than `group-by` in code mode (such as `group-adjacent`, `group-starting-with`, or `group-ending-with`), you cannot switch to designer mode.

Iterate Across Groups

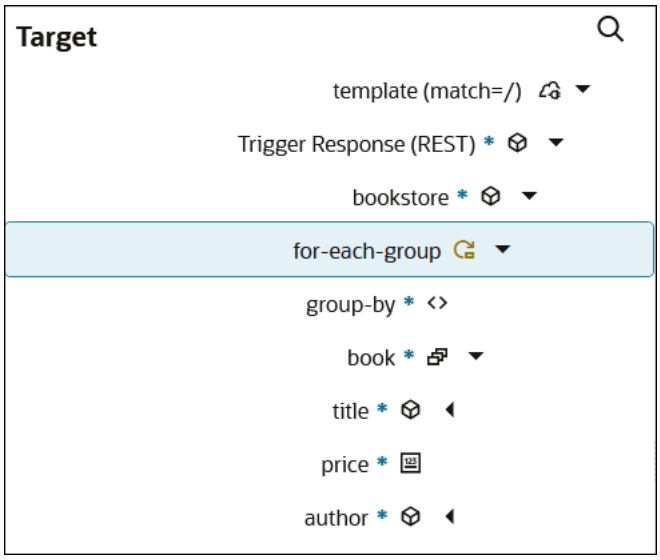
1. Click **XSLT**.
2. In the upper right corner, click **Toggle functions**  .
3. Expand **XSL Constructors**, and then **Flow Control**.
   For this use case, the mapper shows the following source and target elements.



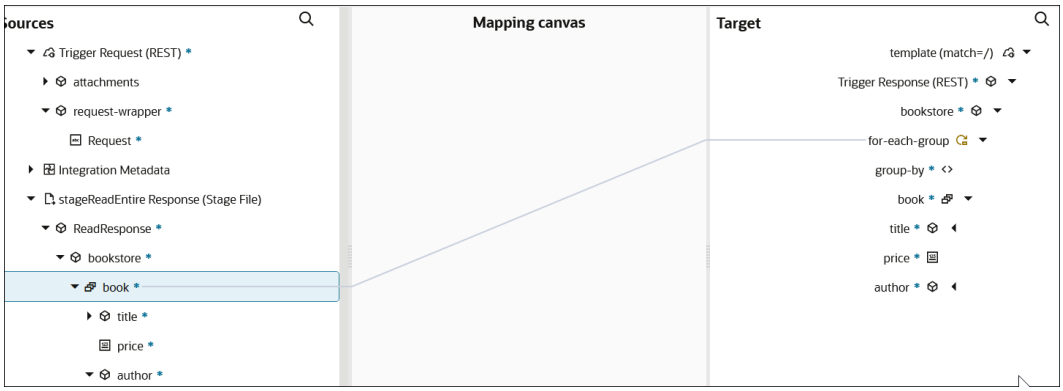
4. Drag a **for-each-group** to the target parent **book** element. The element must be a repeating node.



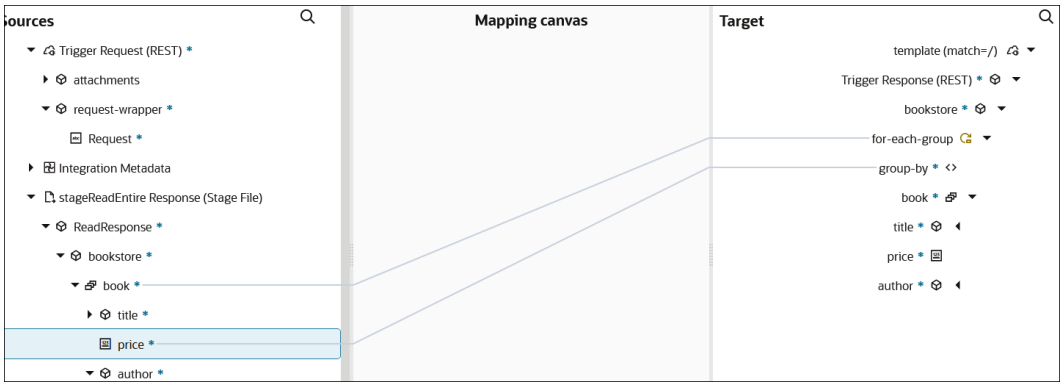
The **for-each-group** constructor includes the **group-by** attribute.



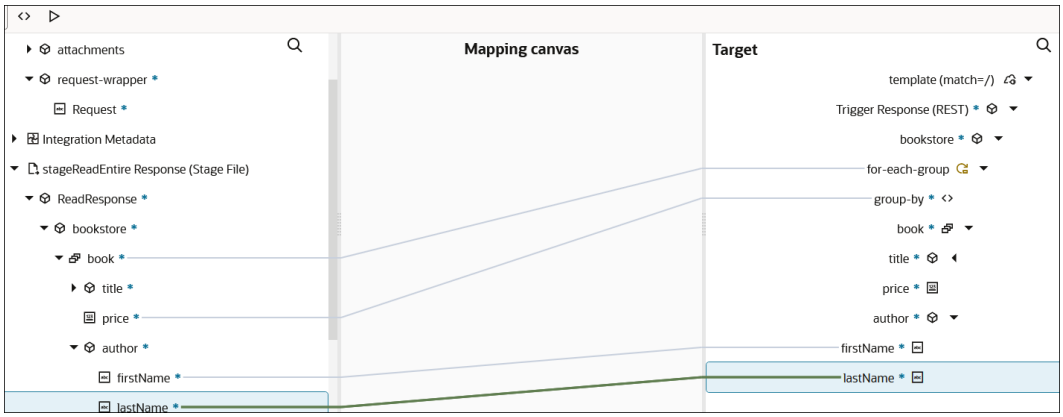
5. Connect the **book** source element to **for-each-group**.



6. Connect the **price** source element to **group-by** to iterate on the basis of book price.



7. Drag the source element **firstName** to the target element **firstName** and the source element **lastName** to the target element **lastName**.

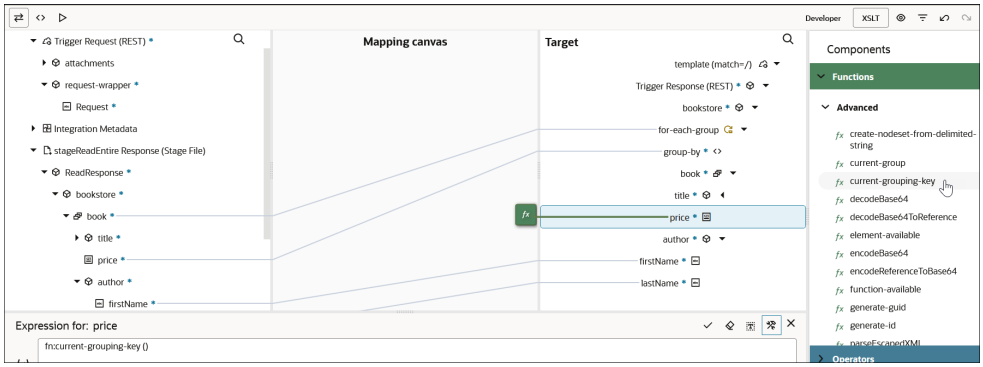


Two supporting functions provided for use with **for-each-group** are available under the **Advanced** category of **Functions**:

- **current-group**: Returns the group currently being processed for each group instruction.
- **current-grouping-key**: Returns the grouping key of the group currently being processed for each group instruction.

This use case provides an example of using **current-grouping-key**.

8. Drag **current-grouping-key** to **price**. This function takes no arguments. For this use case, it returns the price of the group currently being processed.



9. Click **Code** <> to view the code of your mappings. You can edit the code, as necessary.

```
<xsl:for-each-group xml:id="id_87" select="$stageReadEntire/
nsmpr1:ReadResponse/ns26:bookstore/ns26:book" group-by="ns26:price">
      <ns22:book xml:id="id_68">
            <ns22:title xml:id="id_69" lang=""/>
            <ns22:price xml:id="id_73">
                  <xsl:value-of xml:id="id_96" select="fn:current-
grouping-key ()"/>
            </ns22:price>
            <ns22:author xml:id="id_70">
                  <ns22:firstName xml:id="id_72">
                        <xsl:value-of xml:id="id_94"
select="ns26:author/ns26:firstName"/>
                  </ns22:firstName>
                  <ns22:lastName xml:id="id_71">
                        <xsl:value-of xml:id="id_95"
select="ns26:author/ns26:lastName"/>
                  </ns22:lastName>
            </ns22:author>
      </ns22:book>
</xsl:for-each-group>
```

When you exit and return to the mapper, note that the **for-each-group** is no longer visible in designer view. You must select **XSLT** to see the **for-each-group** construct.

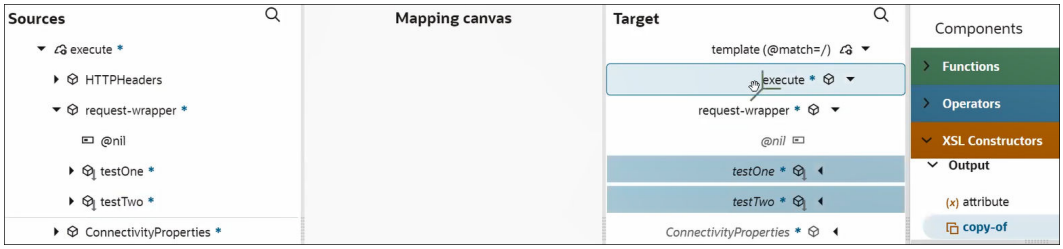
Perform a Deep Copy of Elements with a copy-of Constructor

You can perform a deep copy of all child elements of a source parent node to the child elements of a target parent node in the mapper. This eliminates the need to individually map each source child element to each target child element.

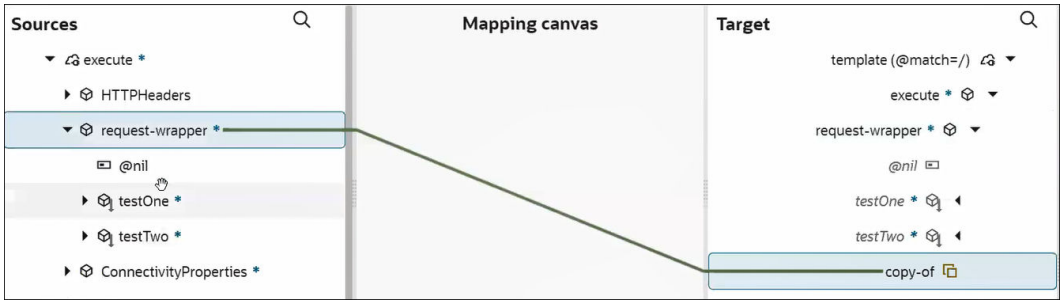
1. Click **XSLT**.

2. In the upper right corner, click **Toggle functions** .

3. Expand **XSL Constructors**, then **Output**.
4. Drag **copy-of** to the target parent element (for this example, **execute**).

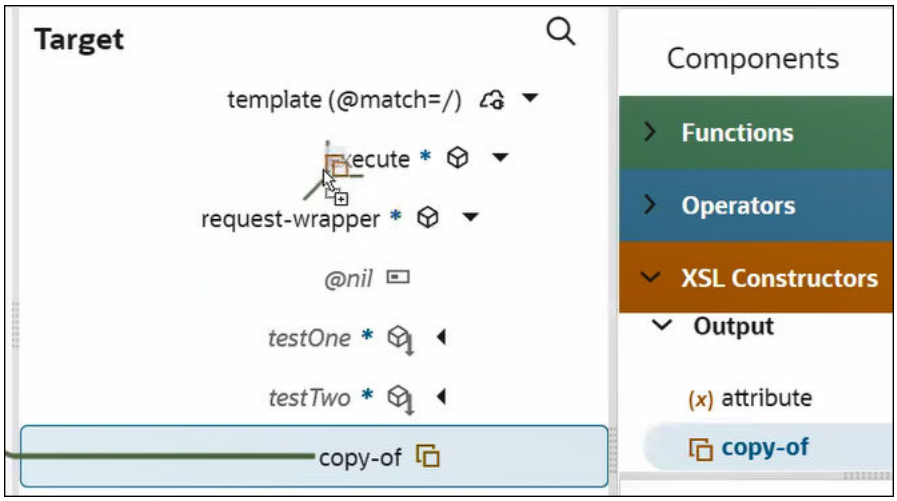


5. Drag the parent element **request-wrapper** in the **Sources** section to the **copy-of** constructor in the **Target** section. This automatically maps all child elements under the source parent element **request-wrapper** to the child elements under the target parent element **request-wrapper**.

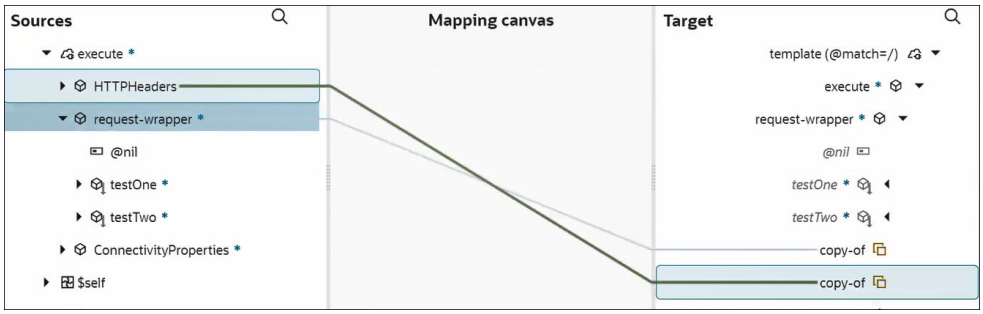


6. Click **Code**  to view the mappings in code view.

7. If required, you can add additional **copy-of** constructors to the target tree.



8. Drag the parent element **HTTPHeaders** in the **Sources** section to the **copy-of** constructor in the **Target** section. This automatically maps all child elements under the source parent element **HTTPHeaders** to the child elements under the target parent element **request-wrapper**.



9. Click **Code** <> again to view the additional mappings in code view.

Use a Counter Inside a For-Each Loop to Track the Number of Loop Iterations

You can use a counter inside a for-each loop to track the number of iterations processed by the loop. This task can be achieved with an `xsl:variable` through direct edit of the XSLT code.

For example:

- Use the `count()` and `position()` XPath functions.
  - The `count(location path to the element)` returns the number of instances for the node-set (for the element).
  - Within an `<xsl:for-each>` loop, the `position()` function returns the iteration number.
- The XSL code snippet for the payload looks as follows:

```
<ns0:po>
  <ns0:details>
    <ns0:item>One</ns0:item>
    <ns0:item>Two</ns0:item>
  </ns0:details>
</ns0:po>
```

- The pseudo XSL code snippet looks as follows:

```
<xsl:template match="/">
  <t01:root>
    <t01:Total_Items>
      <xsl:value-of select= "count(/ns0:po/ns0:details/ns0:item)" />
    </t01:Total_Items>
    <t01:Items_collection>
      <xsl:for-each select= "/ns0:po/ns0:details/ns0:item">
        <t01:Item_Position>
          <xsl:value-of select= "position()" />
        </t01:Item_Position>
        <t01:name>
          <xsl:value-of select= "." />
        </t10:name>
      </xsl:for-each>
    </t01:Items_collection>
  </t01:root>
</xsl:template>
```

- The output looks as follows:

```
<t01:root>
  <t01:Total_Items>2</t01:Total_Items>
  <t01:Items_collection>
    <t01:Item_Position>1</t01:Item_Position>
    <t01:name>One</t10:name>
    <t01:Item_Position>2</t01:Item_Position>
    <t01:name>Two</t10:name>
  </t01:Items_collection>
</t01:root>
```

See Edit XSLT Code in the Mapper.

Create an XSLT Map to Read Multiple Correlated Payloads

You can create XSLT maps to loop thought different sources (input payloads) with instances that are correlated by key fields.

Example for 1:0..*n* and 1:1 Relationships Between Sources

The following business units and employees example is provided:

- Each business unit can have 0..*n* employees (1:0..*n* relationship).
- The G/L accounts source with a 1:1 correlation with business units.

You can create an XSLT map that combines them.

The sources (input payloads) for this example are as follows:

- `$BusinessUnits`

```
<company>
  <bu>
    <id>SD</id> <name>Software Development</name>
    <accounbtid>i9</accountid>
  </bu>
  <bu>
    <id>BS</id> <name>Sales</name>
    <accounbtid>i1</accountid>
  </bu>
  <bu>
    <id>MD</id> <name>Marketing</name>
    <accounbtid>i2</accountid>
  </bu>
 </company>
```

- `$Employees`

```
<people>
  <emp> <buid>SD</buid> <name>Joe Smith</name> </emp>
  <emp> <buid>SD</buid> <name>Mike Jones</name> </emp>
  <emp> <buid>BS</buid> <name>Dave Johnson</name> </emp>
 </people>
```

- `$GLAccounts`

```
<gl>
   <account> <id>i1</id> <number>001.345</number> </account>
   <account> <id>i2</id> <number>001.477</number> </account>
   <account> <id>i9</id> <number>001.223</number> </account>
</gl>
```

The link between `$BusinessUnits` and `$Employees` is the business unit ID. The header is `$BusinessUnit` and the detail is `$Employees`. The link for the GL accounts and business units is the account ID.

The following output is needed:

```
<xxx>
  <yyy>
    <BU id='SD'>Software Development</BU>
    <empName>Joe Smith</empName>
    <accNumber>001.223</accNumber>
  </yyy>
  <yyy>
    <BU id='SD'>Software Development</BU>
    <empName>Mike Jones</empName>
```

```
    <accNumber>001.223</accNumber>
  </yyy>
  <yyy>
    <BU id='BS'>Sales</BU>
    <empName>Dave Johnson</empName>
    <accNumber>001.345</accNumber>
  </yyy>
</xxx>
```

Solution

When the instances (records) of the sources have a 1:1 correlation, you can use a predicate.

When the instances have 1:0..*n* correlation, using an `xsl:for-each-group` performs better than using predicates because it avoids overparsing the source.

The XSLT content is as follows:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<xsl:stylesheet version="2.0"  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:fn="http://www.w3.org/2005/xpath-functions"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:param name="BusinessUnits" />
  <xsl:param name="Employees" />
  <xsl:param name="GLAccounts"/>
  <xsl:template match="/" >
     <xxx>
        <xsl:for-each-group select="$Employees/people/employee" group-
by="buid">
           <!-- this section will be executed only once per 'buid' -->
           <!-- Store the Business Unit Record in a variable -->
           <xsl:variable name="BURecord">
               <xsl:copy-of select="$BusinessUnits/company/bu[id =
fn:current-grouping-key()]"/>
           </xsl:variable>
           <!-- Store the GL Account Record in a variable -->
           <xsl:variable name="GLAccountRecord">
               <xsl:copy-of select="$GLAccounts/gl/account[id = $BURecord/bu/
accountid]" />
           </xsl:variable>
           <!-- end: executed only once per 'buid' -->
           <xsl:for-each select="current-group()">
               <!-- iterates the employees within the current 'buid' -->
               <yyy>
                   <BU id="{./buid}">
                       <xsl:value-of select="$BURecord/bu/name" />
                   </BU>
                   <empName>
                       <xsl:value-of select="./name" />
                   </empName>
                   <accNumber>
                        <xsl:value-of select="$GLAccountRecord/account/
number"/>
                   </accNumber>
```

```
            </yyy>
          </xsl:for-each>
        </xsl:for-each-group>
      </xxx>
  </xsl:template>
</xsl:stylesheet>
```

Summary

- When there is a 1:1 relationship, using predicates instead of `<xsl:for-each-group>` is faster because XSLT does not need to sort the data to create the group.
- When there is a 1:0..*n* relationship, using `<xsl:for-each-group>` performs faster than using predicates. This is because predicates, in the above example, parse the entire business unit source and GL account source per every employee.

See:

- XPath predicates
- xsl:for-each example

Perform Date Conversions in the Mapper

You can perform date conversion tasks in the mapper such as converting dates in different time zones, formats, and timestamps.

- Convert Date from One Time Zone to Another Time Zone
- Convert Date from One Format to Another Format
- Add and Subtract Dates from the Current Date Time
- Convert dateTime to and from Epoch Time
- Convert a Date Timestamp to a UNIX Timestamp
- Convert a Month Number to a Month Name

Convert Date from One Time Zone to Another Time Zone

For example, to convert between India Standard Time (IST) and Greenwich Mean Time (GMT):

- IST to GMT:

```
fn:adjust-dateTime-to-
timezone(xsd:dateTime('2021-05-20T10:49:07.859+05:30'),
xsd:dayTimeDuration('PT0H'))
```

- GMT to IST:

```
fn:adjust-dateTime-to-
timezone(xsd:dateTime('2021-05-20T10:00:00Z'),
xsd:dayTimeDuration('PT5H30M'))
```

Convert Date from One Format to Another Format

Format Definition	XSLT Mapping	Explanation

• Source Format: `DD-MON-YYYY`
 	
`19-Nov-2021`
• Target Format: `YYYY-MM-DD`
 	
`2021-11-19`
	`<xsl:variable name="monthListByNum">`
	`<months>`
	`<month id="1" value="jan"/><month id="2" value="feb"/><month id="3" value="mar"/><month id="4" value="apr"/>`
	`<month id="5" value="may"/><month id="6" value="jun"/><month id="7" value="jul"/><month id="8" value="aug"/>`
	`<month id="9" value="sep"/><month id="10" value="oct"/><month id="11" value="nov"/><month id="12" value="dec"/>`
	`</months>`
	`</xsl:variable>`
	
	`<xsl:value-of select="concat (substring-after (substring-after ($srcDateString, '-' ), '-' ), '-', $monthListByNum/months/month[@value=fn:lower-case (substring-before (substring-after ($srcDateString, '-' ), '-' ) )]/@id, '-', substring-before ($srcDateString, '-' ) )" xml:id="id_18"/>`
		Use an XSLT variable to hold the `MON` to `MM` conversion. Use string manipulation to get the correct format.

Add and Subtract Dates from the Current Date Time

- If the data is in `xsd:dateTime` or `ISO-8601` or `(YYYY-MM-DDTHH:MM:SS+TZ)` format:
  - Add 10 days to the current date time:

    ```
    fn:current-dateTime() + xsd:dayTimeDuration('P10D')
    ```

  - Add one year to the current date time:

    ```
    fn:current-dateTime() + xsd:yearMonthDuration('P1Y')
    ```

  - Subtract 10 days from the current date time:

    ```
    fn:current-dateTime() - xsd:dayTimeDuration('P10D')
    ```

  - Subtract one year from the current date time:

    ```
    fn:current-dateTime() - xsd:yearMonthDuration('P1Y')
    ```

- If the data is in `xsd:date` or `YYYY-MM-DD` format:
  - Convert the data to `xsd:dateTime` format using `xsd:dateTime(concat($inputDate,'T00:00:00'))`.

- If the data is in any other format:
  - Convert the data to `xsd:dateTime` format using the `xp20:format-dateTime()` function or string functions such as `concat()`, `substring-before()`, or `substring-after()`.

Convert dateTime to and from Epoch Time

- If the data is in `xsd:dateTime` or `ISO-8601` or `(YYYY-MM-DDTHH:MM:SS+TZ)` format:
  - Convert the current `dateTime` to epoch time:

```
(fn:current-dateTime() - xsd:dateTime('1970-01-01T00:00:00'))
div xsd:dayTimeDuration('PT1S')
```

  - Convert epoch time to `dateTime`:

```
(xsd:dateTime('1970-01-01T00:00:00') + ($epochTime *
xsd:dayTimeDuration('PT1S')))
```

- If the data is in `xsd:date` or `YYYY-MM-DD` format:
  - Convert the data to `xsd:dateTime` format using `xsd:dateTime(concat($inputDate,'T00:00:00'))`.
- If the data is in any other format:
  - Convert the data to `xsd:dateTime` format using the `xp20:format-dateTime()` function or string functions such as `concat()`, `substring-before()`, or `substring-after()`.

Convert a Date Timestamp to a UNIX Timestamp

```
(fn:current-dateTime() - xsd:dateTime('1970-01-01T00:00:00')) div
xsd:dayTimeDuration('PT1S')
```

Convert a Month Number to a Month Name

If you have the exact date and time, you can use the `format-dateTime` function.

```
xp20:format-dateTime ((fn:current-dateTime ( ), "[MNn" )
```

Perform Data Manipulations in the Mapper

You can perform data manipulation tasks in the mapper such as padding characters, rounding numbers, extracting values, adding CDATA content, and calculating the sum of a node.

- Pad Characters to a String
- Round a Number to the Required Digits
- Extract a Value for a Key-Value Pair Type of XML
- Add CDATA Content to an XML Element
- Calculate the Sum of a Node from a Group of Nodes

Pad Characters to a String

- Left-pad zeroes to nine digits:

```
fn:format-number (12345, '000000000')
```

- Right-pad zeroes to nine digits:

```
concat(string(12345),substring-before(fn:format-number (12345,
'000000000'),
string(12345))
```

Round a Number to the Required Digits

Round a number to two decimals:

```
fn:format-number (12345.12345, '#.00')
```

Extract a Value for a Key-Value Pair Type of XML

XML Snippet	XPath Expression	Explanation

`<ns:SomeElement xmlns:ns="http:// xmlns.oracle.com/some/ namespace">`
`  <ns:ParameterList>`
`  <ns:Parameter>`
`  <ns:Name>ID</ns:Name>`
`  <ns:Value>1</ns:Value>`
`  </ns:Parameter>`
`  <ns:Parameter>`
`  <ns:Name>NAME</ns:Name>`
`  <ns:Value>Oracle</ ns:Value>`
`  </ns:Parameter>`
`  <ns:Parameter>`
`  <ns:Name>AGE</ns:Name>`
`  <ns:Value>25</ns:Value>`
`  </ns:Parameter>`
`  </ns:ParameterList>`
`</ns:SomeElement>`
	Extract the value of parameter name `'NAME'`:
	
	`/ns:SomeElement/ ns:ParameterList /ns:Parameter[ns:Name='NAME']/ ns:Value`
		Anything within the set braces in an XPath is called a predicate. The expression finds the parameter value whose name is `NAME`.

Add CDATA Content to an XML Element

XSLT Snippet	Output XML	Explanation

• Input XML:
 	
`<?xml version = '1.0' encoding = 'UTF-8'?>`
`<nstrgmpr:process>`
`<nstrgmpr:input>`
`<nstrgmpr:data>I SHOULD BE IN CDATA CONTENT<nstrgmpr:data>`
`</nstrgmpr:input>`
`</nstrgmpr:process>`
 	
• XSLT snippet:
 	
`<?xml version = '1.0' encoding = 'UTF-8'?>`
`<xsl:stylesheet version="2.0"`
`xml:id="id_1"`
`xmlns:nstrgmpr="http://`
`xmlns.oracle.com/simpleSvc/SyncSvc/`
`Sync"`
`xmlns:oracle-xsl-mapper="http://`
`www.oracle.com/xsl/mapper/schemas"`
`xmlns:xsd="http://www.w3.org/2001/`
`XMLSchema" exclude-result-prefixes="`
`ora oracle-xsl-mapper`
`oraext xsi fn xp20 xsl ignore01"`
`xmlns:ignore01="http://`
`www.oracle.com/XSL/Transform/java"`
`ignore01:ignorexmlids="true"`
`xmlns:xml="http://www.w3.org/XML/1998/`
`namespace">`
 	
`<xsl:output method="xml" cdata-`
`section-elements="nstrgmpr:result"/>`
 	
`<xsl:template match="/"`
`xml:id="id_11">`
`<nstrgmpr:processResponse`
`xml:id="id_12">`
 	
`<nstrgmpr:result xml:id="id_16">`
`<xsl:value-of select="/`
`nstrgmpr:process/nstrgmpr:input/`
`nstrgmpr:data"`
`xml:id="id_17"/> </nstrgmpr:result>`
 	
`</nstrgmpr:processResponse>`
`</xsl:template>`
`</xsl:stylesheet>`
	`<nstrgmpr:processResponse>`
	
	`<nstrgmpr:result><![CDATA[I SHOULD BE IN CDATA CONTENT]]></nstrgmpr:result>`
	`</nstrgmpr:processResponse>`
		You must manually add the `xsl:output` tag in the XSLT.
		The output attribute must be `xml`.
		The attribute `cdata-section-elements` indicates which fields have the CDATA content post-transformation. In case of multiple elements, the value must be separated by spaces.
		The data should be normally mapped to the element in which CDATA should be present.
		Note: This approach doesn't work for file-based operations such as stage file actions and FTP where the content is rewritten from XSLT.

Calculate the Sum of a Node from a Group of Nodes

XML Snippet	XPath Expression	Explanation

• Input XML
 	
`<ns:SomeElement xmlns:ns="http://xmlns.oracle.com/some/namespace/request">`
 	
`<ns:ParameterList>`
`<ns:Parameter>`
 	
`<ns:Name>ABC</ns:Name>`
 	
`<ns:Value>10</ns:Value>`
`</ns:Parameter>`
`<ns:Parameter>`
 	
`<ns:Name>ABC</ns:Name>`
 	
`<ns:Value>20</ns:Value>`
`</ns:Parameter>`
`<ns:Parameter>`
 	
`<ns:Name>DEF</ns:Name>`
 	
`<ns:Value>20</ns:Value>`
`</ns:Parameter>`
`</ns:ParameterList>`
`</ns:SomeElement>`
	• XSLT snippet:
	
	`<nstrgdfl:ResultElement>`
	`<xsl:for-each-group select="/ns:SomeElement/ns:ParameterList/ns:Parameter" group-by="ns:Name">`
	`<nstrgdfl:ParameterList>`
	`<nstrgdfl:Parameter>`
	`<nstrgdfl:Name>`
	`<xsl:value-of select="fn:current-grouping-key()"/>`
	`</nstrgdfl:Name>`
	`<nstrgdfl:SumOfValues>`
	`<xsl:value-of select="sum(fn:current-group/ns:Value)"/>`
	`</nstrgdfl:SumOfValues>`
	`<nstrgdfl:Parameter>`
	`</nstrgdfl:ParameterList>`
	`</xsl:for-each>`
	`</nstrgdfl:ResultElement>`
	
	• Output XML:
	
	`<ns:ResultElement30xmlns:ns="http://xmlns.oracle.com/some/namespace/response">`
	`<ns:ParameterList>`
	`<ns:Parameter>`
	`<ns:Name>ABC</ns:Name>`
	`<ns:SumOfValues>1</ns:Value>`
	`</ns:Parameter>`
	`<ns:Parameter>`
	`<ns:Name>DEF</ns:Name>`
	`<ns:SumOfValues>20</ns:Value>`
	`</ns:Parameter>`
	`</ns:ParameterList>`
	`</ns:ResultElement>`
		The `SUM` function must be set on `current-group`.

Pass Single Quotes in a Mapper Variable

You can pass single quotes in a mapper variable without using an escape character. Consider the following example. The Expression Builder shows the following literal value.

```
'ACTIVATED'
```

Use an XSL variable to denote single quotes. You can also use an assign action variable.

```
<xsl:variable name="quote" value="'"/>;
```

Use a `concat` function to append the quotes to the actual value.

```
<xsl:value-of select="concat($quote,$value,$quote)"/>
```

Note that the output on the test page and the Track Instances page shows the following.

```
apos;ACTIVATED&apos
```

This is a display issue only and can be ignored. The value is successfully passed.

5

Troubleshoot the Mapper

Review the following topics to learn about troubleshooting issues with the mapper.

Topics:

Current-dateTime Function Does Not Return the Same Number of Digits for All Timestamp Values

The `Current-dateTime function` in the mapper does not return the same number of digits for all timestamp values.

For example, the three digit microsecond value is not the same format each time.

```
YYYY-MM-DDT24:59:59.123Z
YYYY-MM-DDT24:59:59.12Z
```

If you want the specific format value to be consistent, use the `xp20:format-dateTime` function to format the timestamp. For example:

```
xp20:format-dateTime (fn:current-dateTime(), "[Y0001]-[M01]-[D01]T[H01]:
[m01]:[s01].[f001]" )
```

This function returns the following format:

```
2020-10-30T21:58:15.172Z
```

Import XSLT Code into the Mapper

For some functionality that is not available in the mapper, you can import XSLT code.

- To use a nested for-each loop in the target mapper tree and access values from different sources elements, you can use `xsl:variable` with different sources. However, the mapper is locked from editing. As a workaround, you can use imported maps. See Import a Map File into an Orchestrated Integration.

- The ability to use copy-of functionality is not currently available. As a workaround, import XSLT code into the mapper to achieve copy-of functionality. See Import a Map File into an Orchestrated Integration.

Function Not Found Errors During Validation in the Mapper

If you receive the following error when you attempt to validate your mappings in the mapper, you are using an unsupported function (for this example, `document()`). This error can occur when you import an XSLT file into Oracle Integration that includes functions that were supported in another tool such as Oracle JDeveloper.

```
javax.xml.transform.TransformerException: Could not find function:
document:
javax.xml.transform.TransformerException: Could not find function:
document
```

The mapper validates if all the referenced functions are supported. You can only use supported functions (those visible in the **Functions** palette in the mapper user interface) in the mapper in Oracle Integration.

format-number Function Error

The `format-number` function fails with a `cannot convert string to number` error for nondecimal input instead of returning a Not a Number (NaN) response. This is the expected current behavior and applies to all design time and runtime usage of this function.

Transform an Incoming UTC Timestamp into a Standard Timestamp

You can transform an incoming UTC timestamp into a standard timestamp in the mapper.

For example, the incoming data is in the following format:

```
2021-05-24T13:34:45.000000+00:00
```

and needs to transformed as follows:

```
2021-05-24 13:34:45
```

Specify the following expression in the Expression Builder.

```
xp20:format-dateTime( string(fn:current-dateTime() ),"[Y0001]-[M01]-
[D01] [H01]:[m01]:[s01]")
```

CDATA in XSLT String Functions Causes Problems

Do not use CDATA in XSLT string functions (`value-of` turns the content into a string).

For example, assume you are using the mapper to hard code a SOAP Adapter `header` attribute. The mapper encodes the data, which causes a problem for the endpoint service at runtime.

- Mapping:

```
[Expression for: "header"]
"<![CDATA[ <CrosstalkMessage> <CrosstalkHeader>
<ProcessCode>1004</ProcessCode> <MessageType>100</MessageType>
<ExchangePattern>7</ExchangePattern> <EnterpriseId>ace7d6ae-78a8-f3gh-1d04
-9fe0416d053c</EnterpriseId> <Token>h12749ed-913b-7e3e-2aef
-8dd78255cb40</Token> <DestinationId>b3fbf48e-df96-f27d-5fac
-38895618064f</DestinationId> <ContentEncoding>utf-8</ContentEncoding>
<ContentType>text/xml</ContentType> </CrosstalkHeader> </
CrosstalkMessage>
]]>"
```

- Code:

```
<tns:header xml:id="id_48">
      <xsl:value-of xml:id="id_82" select="'&lt;!
[CDATA[ &lt;CrosstalkMessage&gt;
&lt;CrosstalkHeader&gt; &lt;ProcessCode&gt;1004&lt;/ProcessCode&gt;
&lt;MessageType&gt;100&lt;/MessageType&gt;
&lt;ExchangePattern&gt;7&lt;/ExchangePattern&gt;
&lt;EnterpriseId&gt;ace7d6ae
-78a8-f3gh-1d04-9fe0416d053c&lt;/EnterpriseId&gt;
&lt;Token&gt;h12749ed-913b
-7e3e-2aef-8dd78255cb40&lt;/Token&gt; &lt;DestinationId&gt;b3fbf48e-df96-
f27d
-5fac-38895618064f&lt;/DestinationId&gt; &lt;ContentEncoding&gt;utf
-8&lt;/ContentEncoding&gt; &lt;ContentType&gt;text/xml&lt;/
ContentType&gt;
&lt;/CrosstalkHeader&gt; &lt;/CrosstalkMessage&gt; ]]&gt;'"/>
</tns:header>
```

To achieve this:

- Write the XML structure using a stage file action.
- Read the XML as opaque content.
- Map to the header element by using `decodeBase64` of read content.
- Use the `cdata-section-elements` attribute in XSLT referring to the header element.