Informatica® Cloud Application Integration
April 2024

Design

Informatica Cloud Application Integration Design
April 2024

© Copyright Informatica LLC 1993, 2024

See patents at https://www.informatica.com/legal/patents.html.

Table of Contents

Preface

Use *Design* to learn how to create assets such as processes, guides, process objects, service connectors, system services, listeners, and connectors.

Understanding Data Types and Field Properties

The following diagram explains what you learn in this chapter:
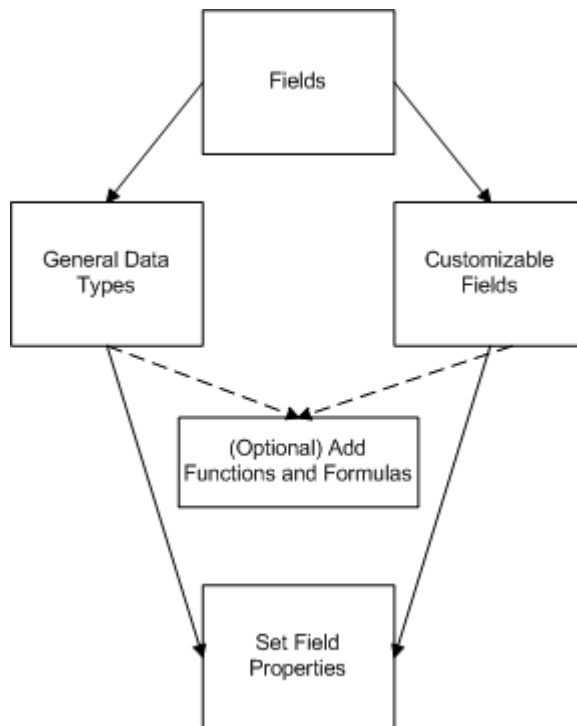
Introduction to Data Types and Field Properties

Process Designer uses information contained within your application database. Each database table contains object types that you can access through your connections to work with the fields needed in a process.

When you define process or step properties, you drill down to view the objects and select associated fields from a list.

For example, in following case, the Salesforce objects give you access to many object types that you can use in the process:
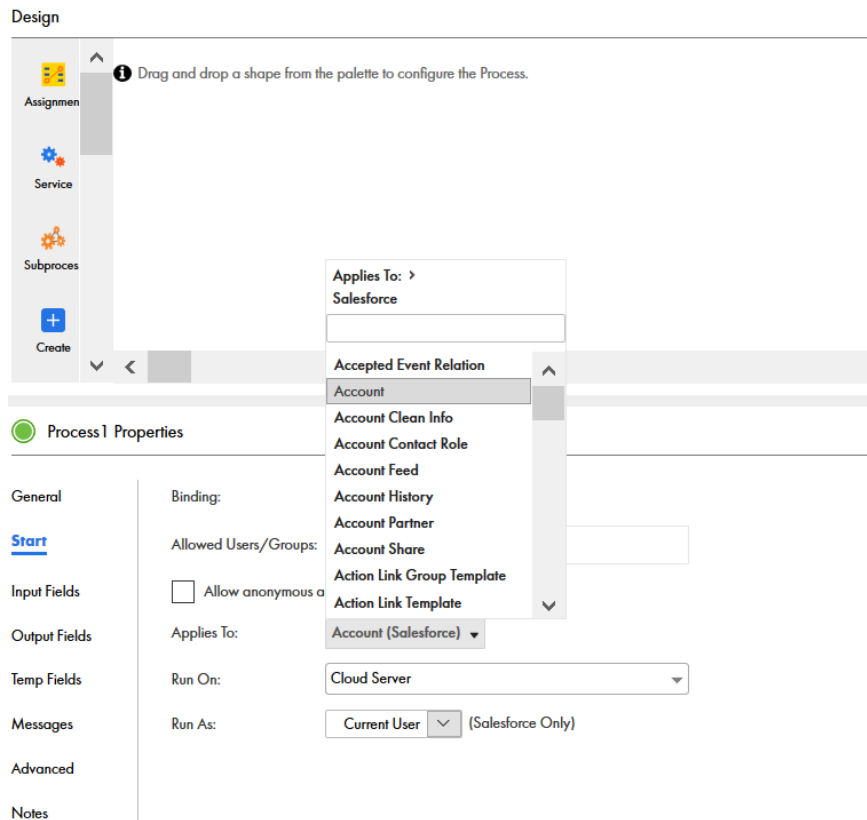
In most cases, a process determines what type of data is being stored based on the application. For other fields, you may need to define them as Simple types and select the appropriate data type. Apart from the data type, you might want to control other field properties. For example, you could specify a default value or hover text to help the user.

To update field properties, simply click in the field to edit.

Read the topics in this section to learn more about the field properties and data types.

Types of Data and Field Properties

You can specify Field Properties, derived from API data types and options, in the Informatica Process Designer. Field Properties may be either of the following:

- General data type properties. For example, specified on the **Fields** tab of the Process Properties.
- A field that you can tailor for the context. For example, a pick list on a Screen step.

The following base options are available for many data types in the Field Properties:

- **Default Value** or **Initial Value**: The value for an item when it is initially displayed.
- **Field**: The name of the field from which the **Field Properties** dialog is invoked.
- **Hover text**: Text that a user sees when they place their cursor over the field.
- **Required**: If checked, a value for this field must exist.
- **Show List**: Values to display are determined at runtime based on what is configured here.

The following table explains built in data types:

Note: Depending on where you access the Field Properties, the available data type options may vary.

Data Type	Description

Any	Use the Any data type if you want to declare fields that are process objects without setting the Reference To option. This means that the data type is an object, but you do not need to specify which object it is. If you use the Any object, you cannot drill further into the object.
	
	Some services return large amounts of data, but you may need only a small percentage of it. Use the Any data type to model all the data returned in to just a few fields.
	
	For example, when you use a REST proxy, it passes a received payload as input and then responds to this payload. The REST proxy would do some processing on the request or response, and it would not have to understand the whole message.
	
	A second example would be supporting heterogeneous lists. While simple REST APIs tend not to include heterogeneous lists, SOAP-based APIs do. For example, a SOAP API might have a "query" operation that returns data that depends on the query itself. Here, the static type of the output for the "query" operation would be an objectlist of the any type.
	
	A third example is calling a service that returns records that match a query. Each record can have a unique set of fields, so those could not be known at design time. At runtime, each field would just be turned into a child element of the record element, using a tagname that corresponds to the field name. The converted JSON returned as output might be:
	
	<pre>{
	  "record": [
	      {"ROWID_OBJECT": "2001", "CREATOR": "admin", ...},
	      {"ROWID_OBJECT": "2002", "CREATOR": "admin", ...},
	      ...
	   ]
	}</pre>

Attachment, Attachments	Use Attachment and Attachments to a process to pass through an attachment and extract data like file size and file name using functions on the response. The default Maximum File Size is 5,242,880 bytes and the default Maximum Number of Files is 10. You can change these field properties. **Note:** If you run a process on the cloud server, do not use an attachment whose size is more than the default 5,242,880 bytes. The cloud server cannot process attachments that are greater than 5,242,880 bytes.
Checkbox	A check box allows the user to make a true/false decision. This decision can alternatively be displayed as Yes/No or No/Yes. For example, you could use a check box to indicate if an address is a work phone number or a home phone number. As an alternative, a check box can be displayed as yes/no values. - **Show as**: A picklist that has the following items: Checkbox, Yes/No, and No/Yes. Your choice sets the labels that the user will see. You can also set the list of checkbox data type for input fields, output fields, temp fields, and messages in a process and guide. You can enter a comma-separated list of boolean values. **Note:** Messages are applicable only for processes. To set the type as list of checkbox, select a field of type **Simple Types > Checkbox**, and then in the **Edit Type** dialog box, select the **Allow a list of objects of this type** check box. To assign the initial values for the list of checkbox data type in the output field, you can enter the value in the following format: `(true())` or `(false())` For more information about using a list of simple type, see "Creating a field with a list of simple type" on page 103.

Data Type	Description
Currency	A currency field contains a monetary value. In addition to numbers and perhaps a decimal point, you can also use commas where they are needed. Commas are always optional. - **Length/Decimal places**: Specify how many numbers can be entered (the left box). Use the right box to enter the maximum number of digits to the right of the decimal point. If you enter 0, there are no digits to the right of the decimal point. - **Hide currency symbol**: When checked, a currency symbol does not display.
Date	The output date is in UTC, and the output date and time are in the ISO 8601 format. For example, if you enter an input date of 2016-03-29, and use the currentTime XQuery function to get date and time values, you see the following output: - Date: 2016-03-29Z - dateTime: 2016-03-29T06:00:48.525Z - Time: 06:00:48.525Z Some sample valid date values are as follows: - 2001-10-26 - 2001-10-26+02:00 - 2001-10-26Z - 2001-10-26+00:00 - -2001-10-26 - -2000-04-01 If you do not pass a time zone, Informatica Process designers assumes the time to be in UTC. If you want to pass a date as a query parameter in the endpoint URL and the date contains special characters, you must encode the date. For example, encode 2001-10-26+02:00 to 2001-10-26%2B02%3A00. If you need to format the way in which the date appears, see *Formatting Dates, Times, and Numbers*. You can also set the list of date data type for input fields, output fields, temp fields, and messages in a process and guide. You can enter a comma-separated list of date values. **Note:** Messages are applicable only for processes. To set the type as list of date, select a field of type **Simple Types > Date**, and then in the **Edit Type** dialog box, select the **Allow a list of objects of this type** check box. For more information about using a list of simple type, see "Creating a field with a list of simple type" on page 103.

Data Type	Description
Date Time	When selecting a Date Time field in a process, the user can select the time from a list. - **Default Value**: Either enter a specific date, a field, or an interval. - **30 Minute Increments**: If selected, a time picklist displays hours and half hours. If it isn't selected, the user can type in any time value (for example, 10:37). The output time is in UTC, and the output date and time are in the ISO 8601 format. For example, if you enter an input date of 2016-03-29, and use the currentTime XQuery function to get date and time values, you see the following output: - Date: 2016-03-29Z - dateTime: 2016-03-29T06:00:48.525Z - Time: 06:00:48.525Z Some sample valid dateTime values are as follows: - 2001-10-26T21:32:52 - 2001-10-26T21:32:52+02:00 - 2001-10-26T19:32:52Z - 2001-10-26T19:32:52+00:00 - -2001-10-26T21:32:52 - 2001-10-26T21:32:52.12679 If you do not pass a time zone, Informatica Process designers assumes the time to be in UTC. If you want to pass a date and time as a query parameter in the endpoint URL and the date and time contains special characters, you must encode the date. For example, encode 2001-10-26T19:32:52+00:00 to 2001-10-26T19%3A32%3A52%2B00%3A00. If you need to format the way in which the date and time appears, see *Formatting Dates, Times, and Numbers*. You can also set the list of date time data type for input fields, output fields, temp fields, and messages in a process and guide. You can enter a comma-separated list of date and time values. **Note:** Messages are applicable only for processes. To set the type as list of date time, select a field of type **Simple Types > Date Time**, and then in the **Edit Type** dialog box, select the **Allow a list of objects of this type** check box. For more information about using a list of simple type, see "Creating a field with a list of simple type" on page 103.
Email	The content of this field is an email address. - **Display # chars**: The number of characters that are displayed within the process for this address. - **Max # chars**: The largest number of characters that can be entered for this address.
Formatted Text	The format is a representation of the kind of character a user can type within the text being entered. - **Format**: A pattern for text that the user types. You can use the following characters when defining a format: 　- **a**: A lowercase letter; that is a through z. 　- **9**: A number. 　- *****: Any letter, number, or symbol **Note**: the "-" is not listed. If you can type this character, it is displayed when the user sees the step.
Image	A field that will contain an image. This is most often used when an automated action retrieves an image. - **Show preview**: Displays a preview of the image. - **Max width**: The maximum width of the area in which the image is displayed. - **Max height**: The maximum height of the area in which the image is displayed. - **Display # chars**: The number of characters that are displayed within the process for this address.

Data Type	Description
Integer	A positive or negative whole number.
	Use the **Digits** field to enter the maximum amount of numbers the user can type.
	Numeric values display using the format of the user's locale.
	If you need to format the way in which the number appears, see *Formatting Dates, Times, and Numbers*.
	You can also set the list of integer data type for input fields, output fields, temp fields, and messages in a process and guide. You can enter a comma-separated list of integer values. **Note:** Messages are applicable only for processes.
	To set the type as list of integer, select a field of type **Simple Types > Integer**, and then in the **Edit Type** dialog box, select the **Allow a list of objects of this type** check box.
	For more information about using a list of simple type, see "Creating a field with a list of simple type" on page 103.
Multi-Select Picklist	A picklist field from which the user can select one or more rows. To select multiple items, the user clicks each one while pressing the CTRL key.
	- **Height (rows)**: The number of rows, each containing one list item that displays in the process. - **Comma Separated List**: The list of items within the list. Separate each item in the list with a comma. Each item specified is available in the list at runtime. **Usage Notes** - If you declare the data type of an input field as a Multi-Select Picklist or a Picklist and you do not enter values for the settings, you must use a semi-colon ";" as a separator instead of a comma. Be sure to use a semi-colon for an Equals condition and a comma for a Contains condition. - If the available values are not specified in the Properties for a Multi-Select Picklist or a Picklist and you receive a list of values from a search service, you must: 1. Specify the search service; and 2. Populate the list from a Screen step or Input options in other steps. Note that the list of available values exists only in the current step. To use the same list in other steps, specify the search service again. - You can also assign values to an undefined Multi-Select Picklist or Picklist in the Assignment step. In that case, use Formula as the Source and the value(s) of the Picklist or Multi-Select Picklist as the Content. When you enter a formula, be sure to enclose the values in quotes, for example: *"value"* for a Picklist and *"value1;value2"* for a Multi-Select Picklist.
Number	A positive or negative decimal number.
	Use the **Length/Decimal places** field to specify how many numbers can be entered (the left box). Use the right box to enter the maximum number of digits to the right of the decimal point. If you enter 0, there are no digits to the right of the decimal point.
	Numeric values display using the format of the user's locale.
	If you need to format the way in which the number appears, see *Formatting Dates, Times, and Numbers*.
	You can also set the list of number data type for input fields, output fields, temp fields, and messages in a process and guide. You can enter a comma-separated list of numeric values. **Note:** Messages are applicable only for processes.
	To set the type as list of number, select a field of type **Simple Types > Number**, and then in the **Edit Type** dialog box, select the **Allow a list of objects of this type** check box.
	For more information about using a list of simple type, see "Creating a field with a list of simple type" on page 103.
Object ID	The object ID is the ID of the object instance. For example, it identifies one of your leads. (This value is assigned to the object by your application.)
	The unique parts of this dialog are discussed after this table. . Fields that are used here and in Field Properties dialogs are discussed at the `top` of this topic.

Data Type	Description
Object List	An object list is a set of object IDs. An object ID is the ID of an object instance. For example, it identifies one of your leads. - **Reference To**: The type of object that can be contained in the list. You must enter the object type's official name, not its label. If the list can contain more than one object types, you can enter a comma-separated list of object types. However, you should avoid doing this as there are query and table capabilities that can't be provided for fields unless they only have one **Reference To** type. - **Display Fields**: The columns (they must be manually entered) from the data associated with the object ID. - **Height (rows)**: The number of rows, each containing one list item, that displays in the process. - **Width** (%): The percent of the area in the used to display information. A read-only object list has a limit of 100 rows, even if the query associated with uses a LIMIT clause, asking for more rows to be returned. Other fields are discussed at the `top` of this topic.
Percent	A number expressed as a per cent. - **Precision**: The maximum number of digits in the number. - **Scale**: The maximum number of digits to the right of the decimal point. If you enter 0, there are no digits to the right of the decimal point. Numeric values display using the format of the user's locale. Here are some examples--each of these examples shows a number greater than 100%: - Precision 5, scale 2: 123.45 or 123.4 or 123. or 123 - Precision 5, scale 0: 12345 or 12 or 12,345 If you need to format the way in which the percent appears, see *Formatting Dates, Times, and Numbers*.
Phone	A field into which you can type a phone number. **Format**: Enter the character that represents a number, which is 9 and any other display characters. For example, 99-999 will display as "_ _ - _ _ _" in the process. **Note:** If you enter the format in any display character other than the number 9, the value becomes static, and you can't edit the field value further when you use the field in the subsequent process steps.
Picklist	A picklist field from which the user can select one row. - **Comma Separated List**: The list of items within the list. Separate each item in the list with a comma. Within the process, each item displays in its own row. **Usage Notes** - If you want to display text to the user and assign a different value to the field, separate the label from the value with an equals ("=") sign. For example, if you type *Red, White, Blue* as the values, the user sees these three strings and the selected string is assigned to the field. However, if you enter *Red=1, White=2, Blue=3*, the user sees the same three strings but the number value for the selected item is assigned to the field. - See also the Usage Notes for Multi-Select Picklists for information on receiving a list of values from a search service.
Rich Text Area	A field that will contain HTML commands as well as text. Enter this text using the HTML editor. - **Height (rows)**: The number of rows within the area the process displays into which the user can type text. - **Width (%)**: The width of the text area relative to the process width. You cannot enter JavaScript into a Rich Text Area.

Data Type	Description
Text	A control that changes the size and number of characters that the user can type text. - **Max # chars**: The maximum number of characters that can be typed when entering text. - **Display # chars**: The number of characters that are displayed within the process. As the process size is limited, use this field to show how much of what the user types is displayed. You can also set the list of text data type for input fields, output fields, temp fields, and messages in a process and guide. You can enter a comma-separated list of text values. **Note:** Messages are applicable only for processes. To set the type as list of text, select a field of type **Simple Types > Text**, and then in the **Edit Type** dialog box, select the **Allow a list of objects of this type** check box. For more information about using a list of simple type, see "Creating a field with a list of simple type" on page 103.
Text Area	A control into which the user can enter text. This can display as one or more rows. - **Width (%)**: The width of the text area relative to the process width. - **Height (rows)**: The number of rows within the area the process displays into which the user can type text.
Time	Select the Time field to select time from a picklist. The field properties dialogs that are displayed are: The output time is in UTC, and output data and time are in ISO 8601 format. For example, if you enter an input date of 2016-03-29, and use the currentTime XQuery function to get date and time values, you see the following output: - Date: 2016-03-29Z - dateTime: 2016-03-29T06:00:48.525Z - Time: 06:00:48.525Z Some sample valid time values are as follows: - 21:32:52 - 21:32:52+02:00 - 19:32:52Z - 19:32:52+00:00 - 21:32:52.12679 If you do not pass a time zone, Informatica Process designers assumes the time to be in UTC. If you check*30 Minute Increments*, you see a time picklist that displays hours and half hours. If it is not checked, you can enter any time value. For example, you can enter 10:37. If you want to pass a time as a query parameter in the endpoint URL and the time contains special characters, you must encode the time. For example, encode 21:32:52+02:00 to 21%3A32%3A52%2B02%3A00. If you want to format the way in which the time appears, see *Formatting Dates, Times, and Numbers*. You can also set the list of time data type for input fields, output fields, temp fields, and messages in a process and guide. You can enter a comma-separated list of time values. **Note:** Messages are applicable only for processes. To set the type as list of time, select a field of type **Simple Types > Time**, and then in the **Edit Type** dialog box, select the **Allow a list of objects of this type** check box. For more information about using a list of simple type, see "Creating a field with a list of simple type" on page 103.

Data Type	Description
URL	A field that will contain a URL. - **For Read-only Display**: Your choices are *Link*, *Button*, or *IFrame*. - **For Read-only Label**: Text that identifies the contents of the field. - **Height**: The height of the area in which the URL is displayed. - **Width**: The width of the area in which the URL is displayed. - **Display # chars**: The number of characters that are displayed within the process. As the process size is limited, use this field to show how much of what the user types is displayed. A URL can be an input, output or temporary process field or even a field from. The field's data is assumed to be a valid URL. To create a button in a screen step: 1. Create a field by using the *Fields* tab in the process properties. 2. Use an Assignment step to assign it data to the URL being linked to. This data, which will be a content, can contain references to other fields such as ID fields by using `{!FieldName}` syntax. Process Designer automatically inserts the data using the button to the right of the text entry area. This is very useful when linking to other pages or other data dependent hyperlinks. 3. On any screen following the Assignment step, place the field within a step by selecting the file from the Read-Only fields picklist. 4. Select the field's ... button to specify its properties. Change **For Read-only Display** to **Button** and enter the text you would like to appear in the button in the **For Read-only Label**.

XML	You can use the XML data type in a process and service connector.
	
	Choose one of the following ways to use the XML data type in a process:
	- Use the XML data type in the Assignment step of a process.
	- Use the XML data type with JMS or AMQP. Here, the payload can be XML data and you can design a process that sends or receives a message.
	
	  If you use XML data with JMS or AMQP, wrap the payload within a root element.
	
	To map simple XML data elements to other fields, use formula and XQuery functions.
	
	For example, consider the following XML:
	
	`<o:order xmlns:o="urn:purchasing:system">`
	`    <o:number>123</o:number>`
	`    <o:amount>9213.32</o:amount>`
	`  </o:order>`
	
	You can use this XQuery function to search for the order number:
	
	`$input.myInboundOrder/*:number/text()`
	
	When you query XML data with namespaces, you must use the asterisk (*) character.
	
	To use the XML data type in a service connector, select the type as **XML** for the input fields and output fields on the **Actions** tab.

Object ID

The object ID is the ID of the object instance. For example, it identifies one of your leads. (This value is assigned to the object by your application.)

- **Reference To**: The type of object that the field refers to. You must enter the object's official name, not its label. If the object can refer to more than one type, you can enter a comma-separated list of object types. However, you should avoid doing this as there are query and table capabilities that can't be provided for fields unless they only have one **Reference To** type.
  A polymorphic relationship is a relationship where the referenced objects can be one of several different object types. If the object ID can refer to more than one object, check the Polymorphic field. A picklist is now available for choosing these objects. For example, while the Account object was selected, objects that could also be referenced are "Accepted Event Relation", "Account Contact", and so on. In this list

- **Display Fields**: The columns (they must be manually entered) from the data associated with the object ID.

When you are using this field in other steps, the information displayed differs. Here's a Process Designer example. When inserting the field, you have two choices.

If you insert the first (PolymorphicID), and click on the "..." in inserted field, the **Field Properties** dialog that displays shows all of the items selected in the Properties dialog:

However, when you select the item from within the Polymorphic section of the Insert Field for Update picklist, what you see are the individual items that were named. You will need to click again to get to the item that you actually want.

Object ID Notes and Comments

When an Object ID is placed in a canvas as a read-only field or put into a column in a table, it displays as a link to the object. If you want to show the Object ID as a number, insert it into a text field as "`{!Id}`". However, when an ID field is included in a result from a **Lookup** dialog, the ID's value displays.

When an object ID is displayed in a column or picklist, Process Designer displays with a meaningful name. For example, `b6f0f0b2-9f38-4771-8365-58eb4f7b7d41` might display as "Acme". However, when an object ID field is included in the results from a Lookup Dialog, it displays using the value of the ID; for example, it might display as `b6f0f0b2-9f38-4771-8365-58eb4f7b7d41`.

If you use a content field to enter an object ID and are using a sandbox, you will need to adjust the ID when you move the process to your production environment. Rather than using a content, you could use a query to load the object ID by name for the object.

Formatting Dates, Times, and Numbers

You can format values within the **Field Properties** dialog (see "Types of Data and Field Properties" on page 11) or within a function invoked from a field where you set the Source to Formula.

The output of a format is a display value and therefore a string.

The `infa:format()` function has four arguments that enable you to specify the format. For example, if you set the Source to Formula, you can use an expression similar to the following:

```
infa.format($output.Created Lead.AnnualRevenue * .75, "#######.")
```

Process Designer evaluates the text of the format string and places a value in that format. For example, the format "#,##0.00" tells Process Designer that at least one number to the left of the decimal point must display as well as two to the right. if they are not present, use a zero. Also, depending upon the length, it can insert a comma.

Refer to "Using Functions" on page 29 for more information.

The patterns you can use for formatting values are the same patterns used in Java classes. If you need information on:

- Dates, see http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html (the SimpleDateFormat Java class).

- Numbers, see http://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html (the DataFormat class).

Using the Field Properties Dialog

When you define steps within a process, you insert fields to handle input, output, or other variables that you might need to receive from a service or pass to another step in the process. Each of the fields you add has a set of field properties that you can configure to determine the field source, the formatting, display properties, and list handling.

For some fields, you can also use the Expression Editor to define complex formulas that determine the field values and other attributes.

The topics in this section describe the options in the Field Properties dialog and the kinds of queries that you can write for use with field properties.

Controls in All Many or Some Field Properties Dialog

Most **Field Properties** dialogs have the following controls:

- **Default Value**: Sets a default value for the information being displayed. Your choices are *Content* (object type), *Field* (has the object type), *Formula*, and *Screen* (user specified object).
- **Display Fields**: Names the fields to be displayed in columns and the order in which they display.
- **Field**: The name of the field from which the **Field Properties** dialog was invoked.
- **Height (rows)**: Sets the number of rows that will be displayed.
- **Hover text**: Text that is displayed after a user places the cursor over the field.
- **Reference To**: The type of object to which the field applies. You cannot change this value--not in all dialog boxes.
- **Required**: Click this checkbox if this value must either be entered or cannot be deleted and not replaced.
- **Show List**: Defines the way in which objects are located.
- **Width (%)**: Sets the width of the displayed information as a percentage of the width of the process.

Show List: Advanced Query

Selecting **Advanced Query** lets you finely tune the information retrieved for the list.

After the query executes, the default action is to return a list whose values are object IDs; the list also has the name field for display.

The source for each of the query's four parameters can be Content, Field, Formula, or Screen. In almost all cases, the source is Content. The items that you can set are follows:

- *Object Type*: The object type upon which the query executes.
- *Where Clause*: Enter an SQL WHERE clause. The text you enter here does not include the WHERE keyword. Here, Content means you will write the WHERE clause, Field means that the field contains the WHERE clause, etc.
- *Display Field:* Set a value here to return a field other than the current one. Process Designer will retrieve this data, and then display it within a picklist. If you do not specify a field, Process Designer uses the *namefield* (the first one, if there is more than one).
- *Value Field*: Returns a different value than what Process Designer would use by default. If you do not set a value field, the ID of the returned object is used.

Advanced Technique: You could use an advanced query to create a step that displays the query's results within a step. You might want to do this in a different "testing" guide.

Show List: Custom List

Selecting **Custom List** let you enter your own list of values. For example, you might individually type the names of the New England states. When the process displays, the user sees the items you enter in a picklist.

You must separate items from one another using a comma (",").

Notice that you can also set a *Default Value*.

Show List: List Child Objects

Use related (or child) objects in the query. While similar to using related objects within fields, using **List Child Objects** is simpler as you do not need to specify the query's condition.

After you select **List Child Objects,** you also need to specify the child's object type. Your choice for source is automatically set to "Content", which is the only allowable source.

Show List: None

Use this setting to display a search box that allows the user to specify any value for the field. If the field is a reference to an object, not using *Show List* lets the user locate objects of the type named in the *Reference To* area.

You control how to display information about objects using the *Display Columns* item.

Show List: Object Query

Use a query to retrieve object information from all of the objects that are selected by the Where Clause.

After you select **Object Query**, the Process Designer adds the Object Query controls to the **Field Properties** dialog. The *Source* for the *Where Clause* is always Content. While you can enter your condition directly into the text area, it is usually far easier to click the **Add Condition** button.

Note: The difference between a value and what is shown in the display field may not be clear. In many cases, they may be the same. The difference is that the display field is what is shown to the user while the value is what is stored. For example, if a currency value is stored with two places after the decimal, its display could omit these numbers. This distinction is almost always used with ID fields as you want to display something like a Name while using the ID as the actual value that is stored when the user selects it. Also, you may be using code tables where the code is something other than an Object Id. For example, imagine a States object where the code is the two letter abbreviation that is stored as the value in other objects, but you want to display the full name to users when they are selecting State.

After using this dialog, you can edit the information that it added.

Selecting the field from the picklist is suggested as it is sometimes not obvious what the field's internal name is. The text entered here is a standard SQL WHERE clause. Also, this text does not include the WHERE keyword.

If you are creating more than one condition, you can use the standard AND and OR operators. You can also use the NOT operator to invert the meaning of the condition.

Each condition has four parts:

- The name of a field in the object. In this example, the field is one of those contained within the Account object (this is the object named in the *Reference To* area).

- An operator that Process Designer uses when it compares the field value on the left with values on the right.

- The kind of data that will be compared. Your choices are Content, Formula, or Field. If you choose *Field*, you are comparing the contents within the field in the current object with the contents of the field in the type of object being queried.

- The data to which the field on the left is being compared. If the source is Content and you click within this area, a small icon appears to the right. After selecting it, Process Designer displays a picklist from which you can select the name of a field in another object that will be used when making the comparison.

To add an SQL LIMIT clause to a query, enter it in the **Where** clause text box. For example, enter "Limit 200" to limit the number of retrieved rows to 200. Add a space after the WHERE information and then enter the SQL LIMIT. Else, enter the SQL LIMIT information on a new line.("Limit" can be in upper, lower, or mixed case.) By default, 100 rows are returned. If your process requires more than 100, you must add this clause. However, a read-only object list has a limit of 100 rows, even if you set a query associated with it to more than 100.

If you use a JDBC connection, you can use the ORDER BY unction to sort data.

To use the order by function, click **Order By** and select a field. For example, you can sort a list of names by a person's last name.

Note: You cannot use the order by function to make OData-enabled requests to a JDBC connector.

Show List: Picklist for Field

Record types are information that you associate with objects that let you select some of the objects. For example, suppose you have "On the Road" and "In the Office" records types for a lead. Your process can use these values to select what other information it should display. If the record type value is "On the Road", you could have a picklist with values A, B, and C. If the value was "In the Office", picklist values could be C, D, and E.

Use this option when using a "Create" Service step with a dependent picklist. While Object Type and Field Name are automatically filled in, you will need to enter a Controller Value. If the user will fill in the field containing the record type, set the Source to Field and also select the appropriate field that this is dependent on. If it is Content just enter its value.

For example, you have a Create Lead Service step that has a "Business Sector" picklist and an "Industry" picklist dependent on it. When the user selects "Technology" from the "Business Sector" picklist, Process Designer can display values set for that sector in the "Industry" picklist, values such as "Computer Hardware" and "Biotechnology." In order for this to operate correctly on the Create screen, you must edit the field properties for the "Industry" field and change the Controller Value" to Field and select "Business Sector" as the field.

Your options when you select Picklist for Field are:

- *Object Type*: The object type containing the object field. This should already be filled in correctly, so you don't need to change it.

- *Field Name*: The name of the field that defines the picklist values. This should already be filled in correctly, so you don't need to change it.

- *Controller Value*: For dependent picklists to display correctly, fill in this controller value. Setting this value to be content is not a good idea since the ID for a record type would be different when used in sandbox and or in the product. What you should do is use a query to dynamically set the controller value. This query must return just one value.

Show List: Users in Role

Users in Role tells Process Designer to locate all users who have the role that you identify.

When you choose this option for Show List, you also select the Role name that Process Designer will use to locate users in that role. The source for each of these can be "Content", "Field", "Formula", or "Screen".

Using the Expression Editor

The Expression Editor allows you to edit field definitions when you define a process or service connector.

You can open the Expression Editor from Process Designer in two contexts:

- When you select Formula as the Source type for a field definition so you can get the field value from a formula.
- When you define the URL, parameters, or HTTP headers for a service connector.

Expression Editor allows you to:

- See syntax highlighting for XML and XQuery.
- View syntax validation feedback.
- Select code completion options that appear as you type.
- Insert common code fragments from a drop-down list.
- Use keyboard shortcuts.
- Choose fields from a list of available input, output, or temp fields defined for the process, the current step, and any upstream steps in the current process. When you choose a field, the appropriate value (for example, '$input.Customer') is inserted into the editor.
- Choose a function from the list of available functions, grouped by category. The list includes some common XQuery functions and displays syntax help as a tool tip, when available.
- Insert the sample XML representing the selected item (for a process object xml element or for a child of a process object).
- Insert common code fragments from a drop-down list.

Note: The Expression Editor in Application Integration comes with powerful functions that can invoke operating system features. You must review the contents passed into the functions before using them.

Opening the Expression Editor

Where available, click the *f(x)* icon to open the Expression Editor.

For example, in an Assignment step, you might add an input field with a Formula as the data source. You can select **Formula** from the list, as shown in the following image:
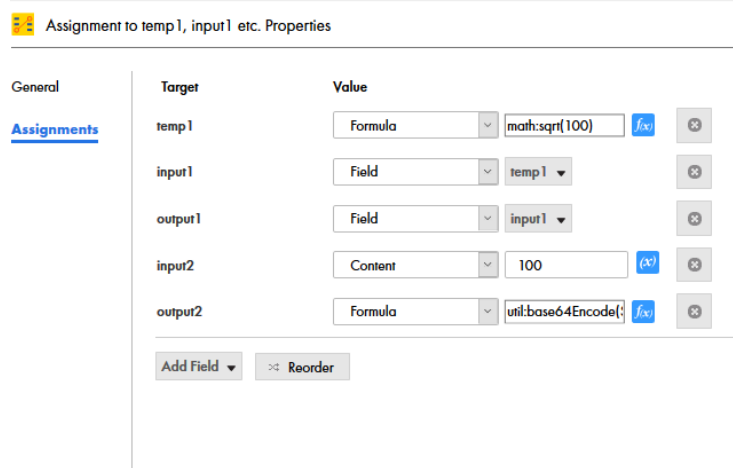
After the Expression Editor opens, you can use the features described below. The field type and name appears in the window title, as shown in the following image (input.TestInput):
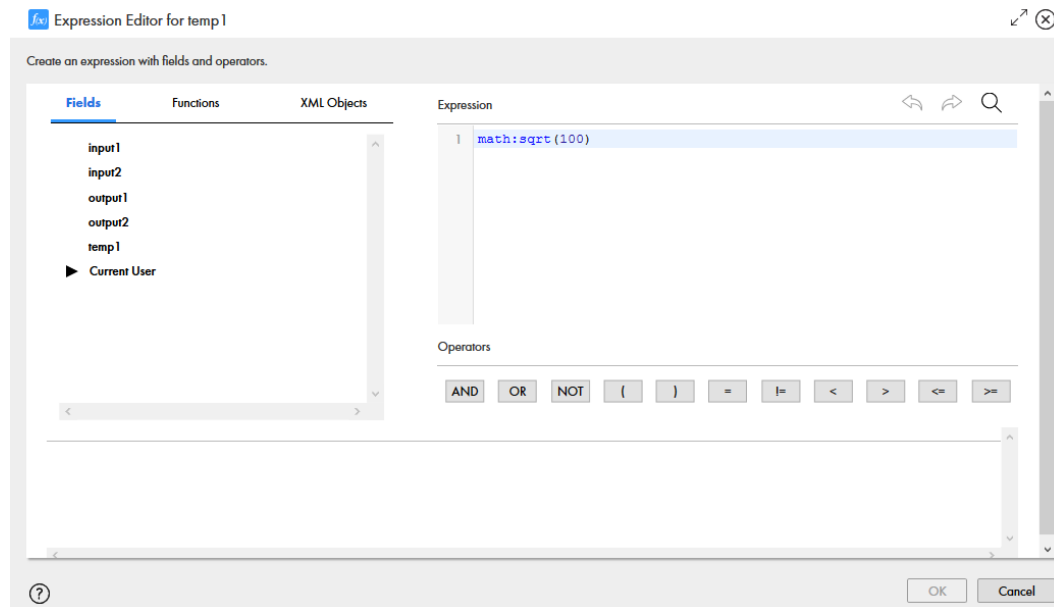
When opened from a service connector, you can choose a type (XML, XQuery, JSON, or Content) and see the fields available in the Connection Properties and Input Parameters, as shown in this image:
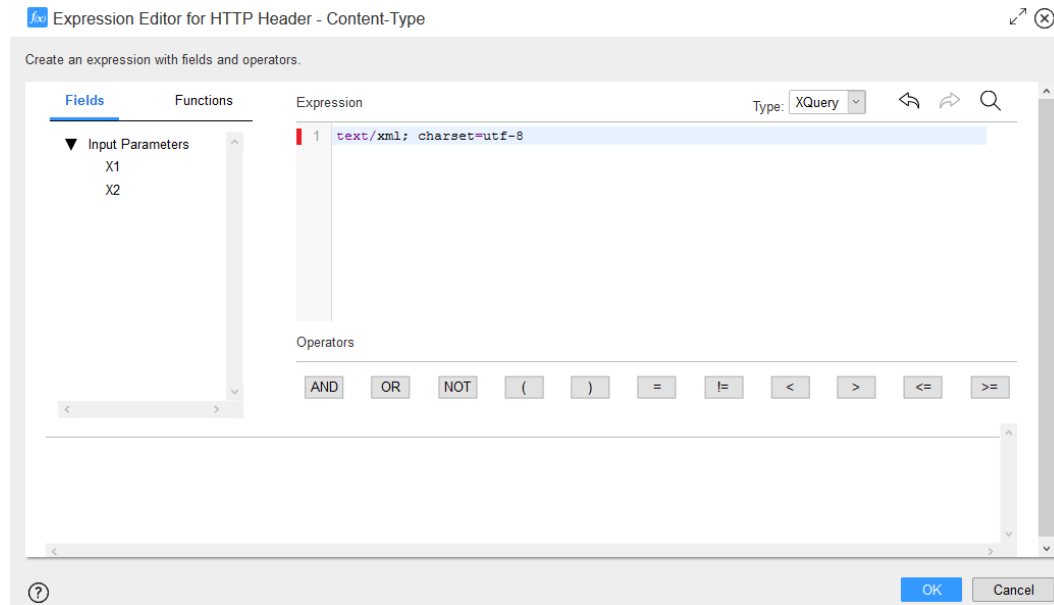
Editing Options

When using the Expression Editor, you can use the toolbar and/or the following keyboard shortcuts, which are available when the editor is active (the cursor is blinking):

Undo	Ctrl+Z
Redo	Ctrl+Y
Copy	Ctrl+C
Cut	Ctrl+X
Paste	Ctrl+V
Find	Ctrl+F
Indent four spaces	Tab
Show list of available variables	$
Show list of available insertions (namespaces, functions, fields and common code fragments)	Ctrl+Space (You can also start typing in the editor to filter the list and select the function or variable you need.)

You can also expand the editor window to full screen using the toolbar icon. Click Esc to close the full screen editor and return to the canvas.

Building a Formula

To enter a formula that determines the value for the selected field, you can access these options from the toolbar:

- Insert Field. Shows the list of available fields you can insert into the formula.
- Insert Function. Shows the list of available functions.
- XML Object. Shows a list of XML objects, if any, that you can add.

For example, in the following image, the **Insert** list shows the field names associated with the object:
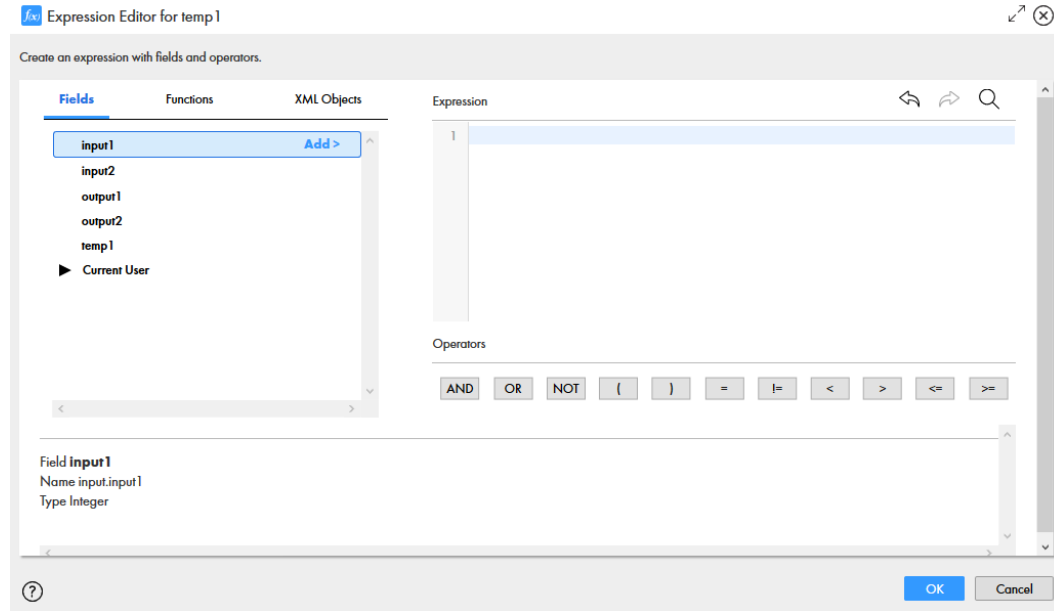
Choose **Functions** to show a list of those available, sorted by category.

Use the **XML Objects** list to add an XML code snippet for the field when you are defining a process object. For example, you might create an XML snippet to represent a process object and then use Insert Field to add the input field as shown in this image:
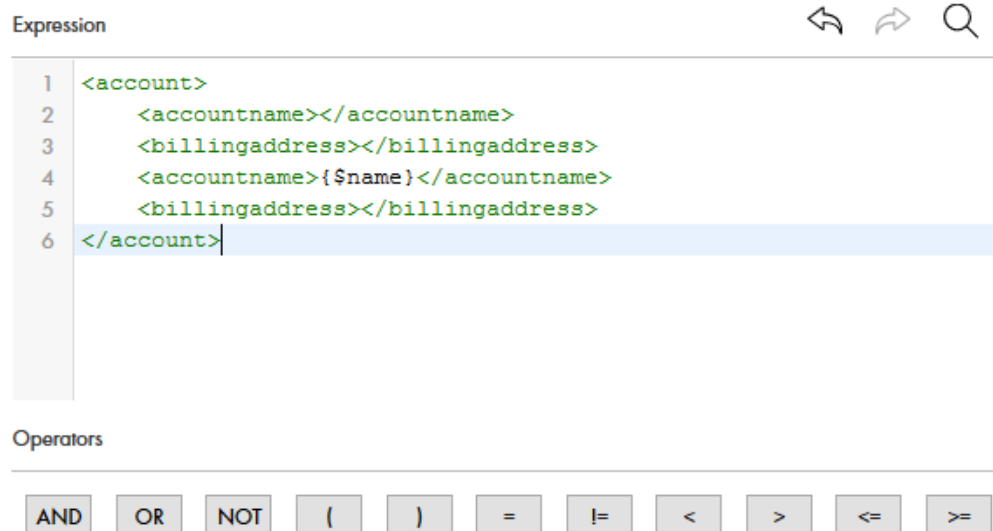
To get more information about a function, hover over the function name in the list to display a description. For example:
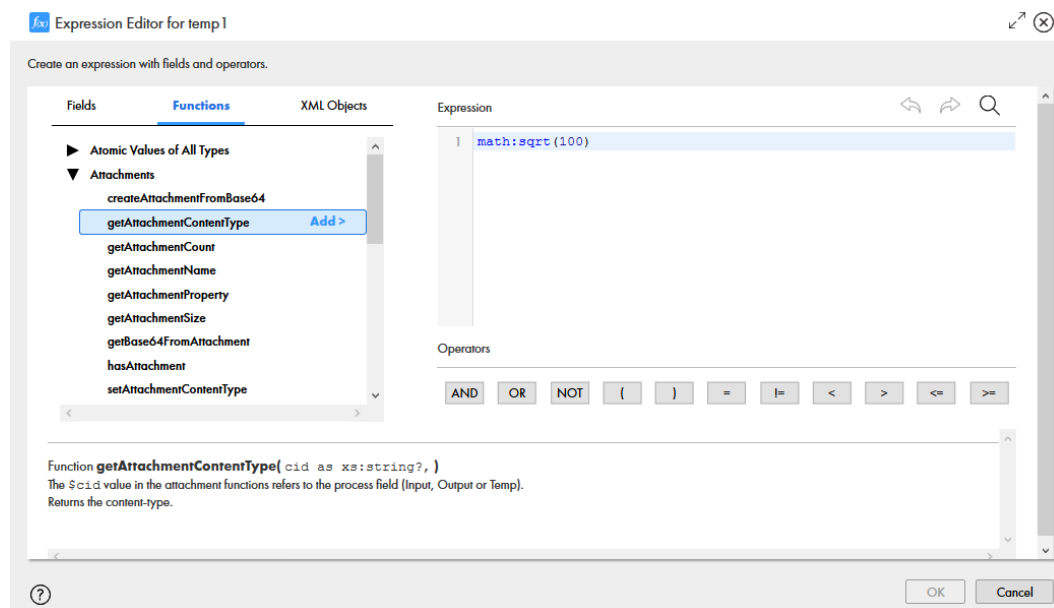
Note the following tips when you work with the editor:

- Press **Ctrl+Space** to display a list of available insertions (namespaces, functions, fields and common code fragments).

- Press **$** to display a list available functions, process fields and any locally declared XQuery variables.

- If the expression is used in XML, be sure to add braces around it, for example:

```
<Value>{2*fn:abs($input.In)}</Value>
```

- Enter a few characters to filter your list.
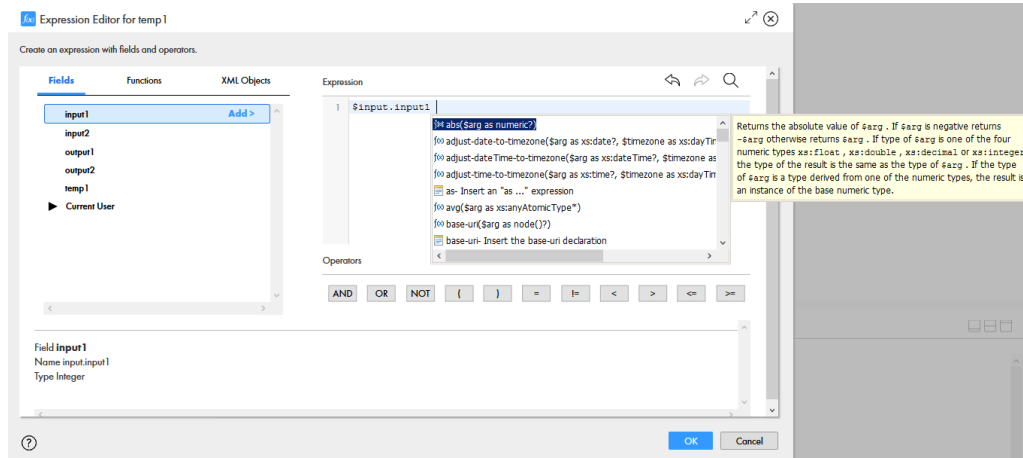
Syntax Validation

As you build a formula, the syntax is validated. If an error is detected, a red X appears next to the line that contains the error. Hover over the error indicator to display a description of the error:
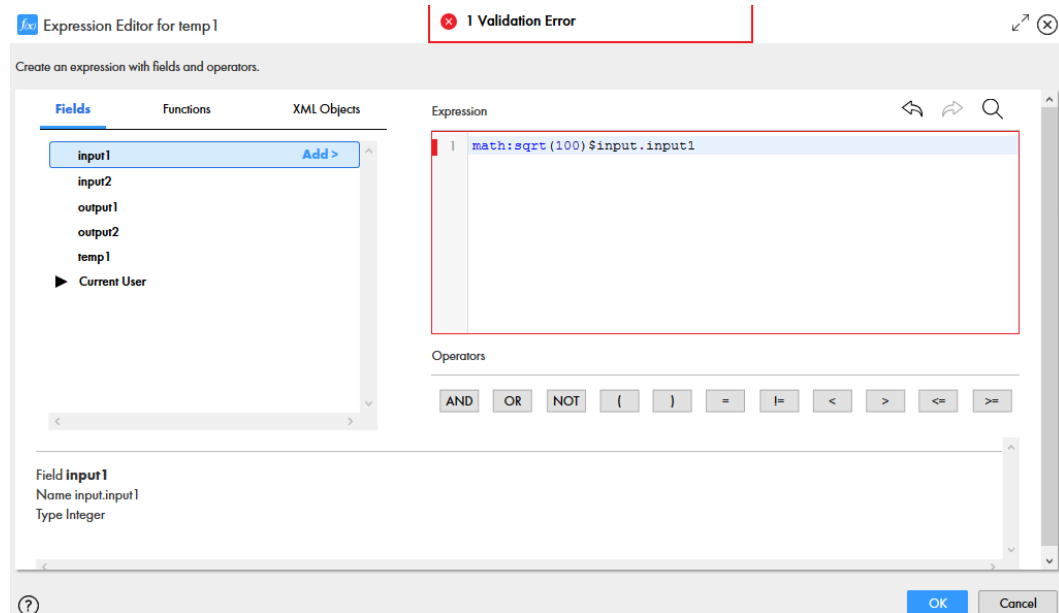
As you type, the syntax is validated and the Expression Editor displays a message if it encounters an error condition.

XQuery error messages are defined by the XQuery engine. You can learn more about XQuery validation and see a list of XQuery error conditions (in *Appendix F*) here: http://www.w3.org/TR/xquery-3/.

Using Functions

You can use the following functions when working with XQuery in Process Designer or in any field that uses a formula to set a value.

For information about XQuery functions that are available in the Expression Editor but are not described below, see http://www.xqueryfunctions.com/xq/alpha.html. Note that Application Integration doesn't support all functions documented in the URL.

For information about using the Expression Editor to work with functions, see "Using the Expression Editor" on page 23.

Note: Application Integration does not support inline functions in the Expression Editor. The Expression Editor comes with powerful functions that can invoke operating system features. You must review the contents passed into the functions before using them.

Atomic Values of All Types

You can use the following functions in processes and service connectors:

- boolean
- empty
- exists
- false
- nilled
- not
- number
- string
- true

Attachments for Processes

The functions described in the following table are available for handling attachments in processes.

In case of a process, the *$cid* value in the attachment functions refers to the process field (Input, Output or Temp). For example, if the input parameter is called customerPhoto (of type attachment), to get the attachment size, use:

sff:GetAttachmentSize($input.customerPhoto)

Note: Be sure that your field names do not have spaces so they can be easily referenced in XQuery.

For more examples of using the attachment functions, see "Attachments" on page 95.

Function	Syntax	Description
createAttachmentFromBase64	sff:createAttachmentFromBase64($contentName as xs:string, $encodedContent as xs:string, $mimeType as xs:string)	Creates an attachment from the base64-encoded content.
getAttachmentContentType	sff:getAttachmentContentType($cid as xs:string?)	Returns the content-type.
getAttachmentCount	sff:getAttachmentCount($cid as xs:long)	Returns the number of attachments.

Function	Syntax	Description
getAttachmentName	sff:getAttachmentName($cid as xs:string?)	Returns the attachment (file) name if available.
getAttachmentProperty	sff:getAttachmentProperty($cid as xs:string?, $attribute as xs:string)	Returns the attachment attribute, given the mime header name such as 'content-type'.
getAttachmentSize	sff:getAttachmentSize($cid as xs:long)	Returns the attachment size in bytes.
getBase64FromAttachment	sff:getBase64FromAttachment($cid as xs:string)	Returns base64-encoded attachment content from the variable, which has the attachment type.
hasAttachment	sff:hasAttachment($cid as xs:boolean)	Checks if an attachment exists.
setAttachmentContentType	sff:setAttachmentContentType($cid as xs:string, $val as xs:string)	Sets the attachment content-type.
setAttachmentName	sff:setAttachmentName($cid as xs:string, $val as xs:string)	Sets the attachment name.
setAttachmentProperty	sff:setAttachmentProperty($cid as xs:string, $attribute as xs:string, $val as xs:string)	Sets the attachment mime header attribute value.

Dates and Times

Process Designer supports most built-in XQuery date and time functions described at
http://www.xqueryfunctions.com/xq/alpha.html, in addition to the following functions:

Function Name	Syntax	Description
addToDate	date:addToDate(date, format, amount)	Adds a specified amount to one part of a datetime value, and returns a date in the same format as the date you pass to the function. For more information, see "addToDate" on page 42.
adjust-date-to-timezone	fn:adjust-date-to-timezone(arg, timezone)	Adjusts a date value to a specific time zone, or removes the time zone component from the date value. For more information, see "adjust-date-to-timezone" on page 44.
adjust-dateTime-to-timezone	fn:adjust-dateTime-to-timezone(arg, timezone)	Adjusts a dateTime value to a specific time zone, or removes the time zone component from the dateTime value. For more information, see "adjust-dateTime-to-timezone" on page 45.
adjust-time-to-timezone	fn:adjust-time-to-timezone(arg, timezone)	Adjusts a time value to a specific time zone, or removes the time zone component from the date value. For more information, see "adjust-time-to-timezone" on page 47.

Function Name	Syntax	Description
current-date	fn:current-date()	Returns the current date. For more information, see "current-date" on page 48.
current-dateTime	fn:current-dateTime()	Returns the current dateTime with time zone. For more information, see "current-dateTime" on page 49.
current-time	fn:current-time()	Returns the current time. For more information, see "current-time" on page 49.
dateDiff	date:dateDiff(date, date, format)	Returns the length of time between two dates in the format specified. For more information, see "dateDiff" on page 49.
dateTime	fn:dateTime(arg1, arg2)	Constructs a dateTime from a date and time. For more information, see "dateTime" on page 52.
dateToMillis	date:dateToMillis($date)	Converts the time to milliseconds.
day-from-date	fn:day-from-date(arg)	Returns the day portion of a date. For more information, see "day-from-date" on page 53.
day-from-dateTime	fn:day-from-dateTime(arg)	Returns the day portion of a date/time. For more information, see "day-from-dateTime" on page 53.
days-from-duration	fn:days-from-duration(arg)	Returns the number of days in a duration. For more information, see "days-from-duration" on page 54.
getDatePart	date:getDatePart(date, format)	Returns the specified part of a date as an integer value. For more information, see "getDatePart" on page 54.
getLocale	date:getLocale()	Returns a string representing the current locale the process is running in. For more information, see "getLocale" on page 56.
getTimeZone	date:getTimeZone()	Returns a string that specifies the timezone ID where the process is running or where the user is running it. For more information, see "getTimeZone" on page 57.
hours-from-dateTime	fn:hours-from-dateTime(arg)	Returns the hour portion of a date/time. For more information, see "hours-from-dateTime" on page 57.
hours-from-duration	fn:hours-from-duration(arg)	Returns the hours in a duration. For more information, see "hours-from-duration" on page 58.
hours-from-time	fn:hours-from-time(arg)	Returns the hour portion of a time. For more information, see "hours-from-time" on page 59.

Function Name	Syntax	Description
implicit-timezone	fn:implicit-timezone()	Returns the value of the implicit timezone property. For more information, see "implicit-timezone" on page 59.
lastDay	date:lastDay(date)	Passes the date for which you want to return the last day of the month. You can enter any valid expression that evaluates to a date. For more information, see "lastDay" on page 60.
toDate	date:toDate(date, format)	Converts a character string to a Date data type. You use the toDate format strings to specify the format of the source strings. For more information, see "toDate" on page 68.
millisToDate	date:millisToDate($millis)	Converts the current time from milliseconds. For more information, see "millisToDate" on page 61.
minutes-from-dateTime	fn:minutes-from-dateTime(arg)	Returns an xs:integer value between 0 and 59, both inclusive, representing the minute component in the localized value of $arg. For more information, see "minutes-from-dateTime" on page 62.
minutes-from-duration	fn:minutes-from-duration(arg)	Returns an xs:integer representing the minutes component in the value of $arg. For more information, see "minutes-from-duration" on page 62.
minutes-from-time	fn:minutes-from-time(arg)	Returns an xs:integer value between 0 and 59, both inclusive, representing the minute component in the localized value of $arg. For more information, see "minutes-from-time" on page 63.
month-from-date	fn:month-from-date(arg)	Returns an xs:integer between 1 and 12, both inclusive, representing the month component in the localized value of $arg. For more information, see "month-from-date" on page 63.
month-from-dateTime	fn:month-from-dateTime(arg)	Returns an xs:integer between 1 and 12, both inclusive, representing the month component in the localized value of $arg. For more information, see "month-from-dateTime" on page 64.
months-from-duration	fn:months-from-duration(arg)	Returns an xs:integer representing the minutes component in the value of $arg. The result is obtained by casting $arg to an xs:dayTimeDuration. For more information, see "months-from-duration" on page 64.
now	date:now()	Returns the current time in milliseconds.

Function Name	Syntax	Description
seconds-from-duration	fn:seconds-from-duration(arg)	Returns an xs:decimal representing the seconds component in the value of $arg. The result is obtained by casting $arg to an xs:dayTimeDuration. For more information, see "seconds-from-duration" on page 66.
seconds-from-dateTime	fn:seconds-from-dateTime(arg)	Returns an xs:decimal value greater than or equal to 0 and less than 60, representing the seconds and fractional seconds in the localized value of $arg. For more information, see "seconds-from-dateTime" on page 65.
seconds-from-time	fn:seconds-from-time(arg)	Returns an xs:decimal value greater than or equal to 0 and less than 60, representing the seconds and fractional seconds in the localized value of $arg. For more information, see "seconds-from-time" on page 66.
timezone-from-date	fn:timezone-from-date(arg)	Returns the timezone component of $arg, if any. For more information, see "timezone-from-date " on page 67.
timezone-from-dateTime	fn:timezone-from-dateTime(arg)	Returns the timezone component of an xs:dateTime. For more information, see "timezone-from-dateTime" on page 67.
timezone-from-time	fn:timezone-from-time(arg)	Returns the timezone component of an xs:time. For more information, see "timezone-from-time" on page 68.
toChar	date:toChar(date, format)	Converts the date passed to a string based on the specified format string. For more information, see "toChar" on page 68.
trunc	date:trunc(xs:dateTime('date'), 'format')	Truncates dates to a specific year, month, day, hour, minute, second, or millisecond. For more information, see "trunc (Dates)" on page 71.
year-from-date	fn:year-from-date(arg)	Returns an xs:integer representing the year in the localized value of $arg. The value might be negative. For more information, see "year-from-date" on page 74.
year-from-dateTime	fn:year-from-dateTime(arg)	Returns an xs:integer representing the year component in the localized value of $arg. For more information see "year-from-dateTime" on page 75.
years-from-duration	fn:years-from-duration(arg)	Returns an xs:integer representing the years component in the value of $arg. The result is obtained by casting $arg to an xs:yearMonthDuration. For more information, see "years-from-duration" on page 75.

Digital Signatures

The functions described in the following sections are available for use in digital signatures.

HMAC Functions

The following functions enable you to generate a digital signature by calculating a keyed-hash message authentication code (HMAC):

Function	Syntax	Description
hmacSignature	dsig:hmacSignature($data as xs:string, $key as xs:string, $algorithm as xs:string, $encoding as xs:string?) as xs:string	Generates an HMAC signature using the specified algorithm such as HMACSHA256 or HMACSHA512 and the optional encoding where $encoding is one of the following values: - Base64 (default) - Base64Url - Hex - Hex64
hmacSHA1signature	dsig:hmacSHA1signature($data as xs:string, $key as xs:string, $encoding as xs:string?) as xs:string	Calculates an HMAC using the SHA1 algorithm and the optional encoding where $encoding is one of the following values: - Base64 (default) - Base64Url - Hex - Hex64 $data consist of four parts, including a JSON string in base64 encoding. To ensure that you get hex binary and not base64 when you use the hash:hash function on the payload, use the following expression: `let $md5hex := hash:hash($jsonPayload,"MD5") return xs:base64Binary(xs:hexBinary($md5hex))`
hmacSHA256signature	dsig:hmacSHA256signature($data as xs:string, $key as xs:string, $encoding as xs:string?) as xs:string	Calculates an HMAC using the SHA256 algorithm and the optional encoding where $encoding is one of the following values: - Base64 (default) - Base64Url - Hex - Hex64
hmacSHA512signature	dsig:hmacSHA512signature($data as xs:string, $key as xs:string, $encoding as xs:string?) as xs:string	Generates an HMAC SHA512 signature using the optional encoding where $encoding is one of the following values: - Base64 (default) - Base64Url - Hex - Hex64
hmacSHA256signatureForList	dsig:hmacSHA256signatureForList($data as xs:string, @delimiter as xs:string, $key as xs:string, $encoding as xs:string?) as xs:string	Generates an HMAC SHA512 signature using the optional encoding where $encoding is one of the following values: - Base64 (default) - Base64Url - Hex - Hex64 For more information, see "hmacSHA256signatureForList" on page 94.

Key Signing Functions

The following functions enable you to generate digital signatures based on private keys:

Function	Syntax	Description
signWithKeyFile	dsig:signWithKeyFile($messageToSign as xs:string, $pathToKey as xs:string, $encryptionAlgorithm as xs:string, $digestAlgorithm as xs:string, $encoding as xs:string) as xs:string	Generates a signature using an asymmetric algorithm and a private key, specified in a PKCS8 file. Arguments: - $digestAlgorithm: SHA1 or SHA256 - $encryptionAlgorithm: RSA (common) or DSA (if using SHA1 for $digestAlgorithm). - $pathToKey: The PKCS8 certificate as a Base64-encoded string (-----BEGIN PRIVATE KEY----- ..... n-----END PRIVATE KEY-----\n) or a binary private key file. - $encoding (optional), which may be: - Base64 (default) - Hex64 - Base64Url
signWithKeyString	dsig:signWithKeyString($messageToSign as xs:string, $key as xs:string, $encryptionAlgorithm as xs:string, $digestAlgorithm as xs:string, $encoding as xs:string) as xs:string	Generates a signature using an asymmetric algorithm and a private key, specified in a PKCS8 certificate encoded string. Arguments: - $digestAlgorithm: SHA1 or SHA256 - $encryptionAlgorithm: RSA (common) or DSA (if using SHA1 for $digestAlgorithm). - $key: The PKCS8 certificate as a Base64-encoded string (-----BEGIN PRIVATE KEY----- ..... n-----END PRIVATE KEY-----\n) or a binary private key. - $encoding (optional), which may be: - Base64 (default) - Hex64 - Base64Url
signWithCertificate	dsig:signWithCertificate($messageToSign as xs:string, $pathToCertificate as xs:string, $keyRecoveryPassword as xs:string, $encryptionAlgorithm as xs:string, $digestAlgorithm as xs:string, $encoding as xs:string $keyStorePassword as xs:string, $aliasName as xs:string, $keyStoreType as xs:string) as xs:string	Generates a signature using a PKCS12 certificate. Arguments: - $pathToCertificate: File location on the agent that contains either a PKCS12 or JKS (Java keystore) certificate. - $keyRecoveryPassword: Password to access the key in the certificate. - $keyStorePassword: Password to open the key store. If empty, assumes the keystore is not password-protected. - $aliasName: Optional. Alias of entry in the keystore on the Secure Agent which contains the key. If the alias is not supplied (or empty), the first entry is used. - $keyStoreType: Type of keystore, which may be: - PKCS12 (default) - JKS (Java keystore)

Hashing Functions

The following functions enable you to generate a message hash string:

Function	Syntax	Description
hash	hash:hash($string, $alg)	Generates a hash string for a message using the specified algorithm (optional), where $alg is one of the following: - MD5 (default) - SHA1 - SHA256

List Functions

The list functions support only the native Saxon functions. The following functions allow you to work with lists:

Function	Syntax	Description
append	list:append($objectlist, $newItem)	Appends a new item to a list.
count	list:count($objectlist)	Counts the items in a list.
head	list:head($objectlist)	Returns the first item in a list.
list	list:list($sequence)	Converts a sequence of IDs into a semicolon-separated list of IDs for an object list.
remove	list:remove($objectlist, $position)	Removes the item at the specified position from the list.
replace	list:replace($objectlist, $position, $newItem)	Replaces an existing item in a list with a new value.
sequence	list:sequence($objectlistFieldName)	Converts a semicolon-separated list to a sequence. When an object list field is inserted into formula and content, the field is wrapped in the list:sequence(object_field_name) XQuery function. For object lists in hosted objects, this converts the semicolon-separated list of IDs in the object list into a sequence. For object lists for process objects, the value is already a sequence and the function returns the list unchanged.
tail	list:tail($objectlist)	Return all items from the list with the exception of the first item.

List Function Examples

Perhaps the most common example of using the list functions is to get an object from a list each time a step is invoked, possibly from within a repeated Process or Service step. For example:

```
let mylist.Current:=list:head(mylist.List)
let mylist.List:=list:tail(mylist.List)
```

The following example converts an object list in a semicolon-separated list, then iterates over each item:

```
for $objectid in list:sequence($objectlist)
...
```

The following example converts a sequence of IDs into a semicolon-separated list of IDs for an object list for hosted objects. For an object list for process objects, it returns a sequence of values:

```
let $mergedObjectLists :=
    ( as:sequence($objectlist1), as:sequence($objectlist2) )
return list:list($mergedObjectLists)
```

Math

In processes and service connectors, you can use the math functions described at http://www.w3.org/2005/xpath-functions/math.

Miscellaneous

The following miscellaneous functions are available:

Function	Syntax	Description
aesEncryption	util:aesEncryption(key, dataToEncrypt)	Encrypts the specified data with the provided key using the Advanced Encryption Standard (AES) algorithm. For more information about the function, see "aesEncryption" on page 76.
aesDecryption	util:aesDecryption(key, dataToDecrypt)	Decrypts the specified data with the provided key using the Advanced Encryption Standard (AES) algorithm. You must use the same key that you had used for encrypting the data in the aesEncryption function. For more information about the function, see "aesDecryption" on page 76.
base64Decode	util:base64Decode(data, charSet)	Returns the base64-decoded version of the input string provided based on the character set specified in the charSet argument. This function is typically used for attachments. Application Integration supports the character sets that Azul JDK supports for encoding. For example, US-ASCII, ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, and UTF-16. Default for the charSet argument is UTF-8.
base64Encode	util:base64Encode(data, charSet)	Returns the base64-encoded version of the input string provided based on the character set specified in the charSet argument. Application Integration supports the character sets that Azul JDK supports for encoding. For example, US-ASCII, ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, and UTF-16. Default for the charSet argument is UTF-8.
base64EncodeURL	util:base64EncodeUrl(str, charSet)	Returns a base64-encoded version of the string provided, safe to use in a URL. Any "+" and "/" characters are replaced with "-" and "_". Any "=" characters are removed. Default for the charSet argument is UTF-8.
decode	util:decode(value, search1, result1, args, default)	Searches a field for the specified value. For more information, see "decode" on page 77.
error	fn:error(error, description, error-object)	Raises a custom error.

Function	Syntax	Description
escapeJsonString	util:escapeJsonString(str)	Escapes special characters in the provided string to make it valid for usage in JSON. See JSON standard for details. For more information about the function, see "escape-Json-String " on page 79.
exactly-one	fn:exactly-one(arg)	Returns a sequence if it contains exactly one item, otherwise returns errors.
format	util:format(value, pattern, timezoneId, locale)	Formats string content. Use the following arguments: - value: The string being formatted. - pattern: A pattern describing how the value must be formatted. See "Formatting Dates, Times, and Numbers" on page 19. - timezoneId: Optional. The time zone ID. If you do not use a stored value (typically returned by a call to date:getTimeZone()), you can omit this argument as Process Designer will call date:getTimeZone. - locale: Optional. The locale. If you do not use a stored value (typically returned by a call to date:getLocale()), you can omit this argument as Process Designer will call date:getLocale(). If you use a locale argument but do not use a timezoneId argument, you must add the comma that would follow timezoneId. For example: `util:format("789","##00.00","",date:getLocale())` If the pattern argument contains a date, use lowercase letters to denote the year format. For example, use **yyyy**. If you use uppercase letters, errors might occur.
generate-random-string	util:generate-random-string(length as xs:integer)	Generates a random string of the specified length.
generateUUID	util:generateUUID()	Generates a universally unique identifier.
getAssetLocation	util:getAssetLocation()	Returns the location where the process or the guide that uses the function is stored. For more information, see "getAssetLocation" on page 79.
getAssetName	util:getAssetName()	Returns the name of the process or the guide that uses the function. For more information, see "getAssetName" on page 79.
getCatalogResource	util:getCatalogResource()	Returns the resource based on a catalog location URL within the organization. Takes any catalog resource in the Informatica Cloud organization as its parameter and returns an element node. To display the element data, add a wrapper function. For example: `serialize(util:getCatalogResource("project:/ spi.ipd/services.xml"))`

Function	Syntax	Description
getInstanceStartTime	util:getInstanceStartTime()	Returns the start time of the running instance of the specified process or guide. For more information, see "getInstanceStartTime" on page 80.
getOrganizationId	util:getOrganizationId()	Returns a string that is the organization Id of the context of the currently executing process. For more information, see "getOrganizationId" on page 80.
getOrganizationName	util:getOrganizationName()	Returns the organization name in the context of the executing process, guide, service connector, or data access service connector. For more information, see "getOrganizationName" on page 80.
getProcessId	util:getProcessId()	Returns the process Id of the currently executing or completed process.
getUserName	util:getUserName()	Returns a string that is the login name or ID of the authenticated User running the process .
getUserSystem	util:getUserSystem()	Returns a string that is the name of the system that authenticated the user running the process.
iif	util:iif(condition, val1, val2)	Returns val1 if the condition is true. Else, returns val2. For more information, see "iif" on page 80.
in	util:in(valueToSearch, values, caseFlag)	Matches input data with a list of values. By default, the match is case sensitive. For more information, see "in" on page 82.
isNull	util:isNull(value)	Checks whether the input passed is empty. For more information, see "isNull" on page 83.
one-or-more	fn:one-or-more	Returns a sequence if it contains one or more items, otherwise returns errors.
parseJSON	util:parseJSON(jsonStr)	Parses the provided JSON string and converts it to XML elements.
parseXML	util:parseXML(xmlStr)	Parses the provided XML string and converts it to an XML element.
random	util:random()	Returns a random number from 0 to 1.
resolveURN	util:resolveURN()	Retrieves the URN mapping for an organization.
safeNumber	util:safeNumber()	Holds a number value safely so it cannot be changed.
toDecimal	util:toDecimal(value, scale)	Converts a string or numeric value to a decimal value. For more information, see "toDecimal" on page 84.
toInteger	util:toInteger(value, flag)	Converts a string or numeric value to a decimal value. For more information, see "toInteger" on page 86.
setProcessTitle	ipd:setProcessTitle()	Sets the title of the process. **Note:** Use this function only in Informatica Process Designer.

Function	Syntax	Description
simplifyXml	util:simplifyXml(undefined)	Utility function that is used to remove all namespaces and convert attributes into child elements. For more information, see "simplifyXml" on page 89.
toJSON	util:toJSON(elements)	Converts the provided list of XML elements to a JSON string.
toXML	util:toXML(element)	Converts the provided XML element to an XML string.
trace	util:trunc(arg, precision)	Provides an execution trace to be used in debugging queries. For more information, see "trace" on page 87.
trunc	util:trunc(arg, precision)	Truncates numbers to a specific digit based on the number of decimal places specified by the precision. For more information, see "trunc (Numbers)" on page 88.
zero-or-one	fn:zero-or-one(arg)	Returns a sequence if it contains zero or one items, otherwise returns an error.

Numbers

The following number functions are available in the Expression Editor:

- abs
- avg
- ceiling
- floor
- max
- min
- round
- round-half-to-even
- sum

Sequences

The following sequences functions are available in the Expression Editor:

- count
- distinct-values
- index-of
- insert-before
- last
- position
- remove
- reverse
- subsequence
- unordered

String

The following string functions are available in the Expression Editor:

- codepoint-equal
- codepoints-to-string
- compare
- concat
- contains
- default-collation
- ends-with
- lang
- lower-case
- matches
- normalize-space
- normalize-unicode
- replace
- starts-with
- string-join
- string-to-codepoints
- substring
- substring-after
- substring-before
- tokenize
- translate
- upper-case

XML

You can add common XML functions as you build expressions. From the Expression Editor, a list of common XML functions displays in these categories:

- XML Documents, URIs, and IDs (processes only)
- XML Namespaces and Names
- XML Nodes

The following XML Nodes functions are available in the Expression Editor:

Function	Syntax	Description
data	fn:data(arg)	Returns the atomic value of a node. The function takes a sequence of items and returns a sequence of atomic values. For more information, see "data" on page 89.
deep-equal	fn:deep-equal(parameter1, parameter2, collation)	Assesses whether two nodes have the same content and attributes. For more information, see "deep-equal" on page 90.
root	fn:root(arg)	Returns the root of the tree that contains the argument. For more information, see "root" on page 91.

Date and time functions

The following date functions are available in the Dates and Times section of the Expression Editor:

addToDate

Adds a specified amount to one part of a datetime value, and returns a date in the same format as the date you pass to the function. The addToDate function accepts positive and negative integer values. Use addToDate to change the following parts of a date:

- **Year.** Enter a positive or negative integer in the *amount* argument. Use any of the year format strings: Y, YY, YYY, or YYYY. The following expression adds 10 years to all dates in the SHIP_DATE column:

    ```
    date:addToDate(xs:dateTime('SHIP_DATE'), 'YY', 10)
    ```

- **Month.** Enter a positive or negative integer in the *amount* argument. Use any of the month format strings: MM, MON, MONTH. The following expression subtracts 10 months from each date in the SHIP_DATE column:

    ```
    date:addToDate(xs:dateTime('SHIP_DATE'), 'MONTH', -10)
    ```

- **Day.** Enter a positive or negative integer in the *amount* argument. Use any of the day format strings: D, DD, DDD, DY, and DAY. The following expression adds 10 days to each date in the SHIP_DATE column:

    ```
    date:addToDate(xs:dateTime('SHIP_DATE'), 'DD', 10)
    ```

- **Hour.** Enter a positive or negative integer in the *amount* argument. Use any of the hour format strings: HH, HH12, HH24. The following expression adds 14 hours to each date in the SHIP_DATE column:

    ```
    date:addToDate(xs:dateTime('SHIP_DATE'), 'HH', 14)
    ```

- **Minute.** Enter a positive or negative integer in the *amount* argument. Use the MI format string to set the minute. The following expression adds 25 minutes to each date in the SHIP_DATE column:

    ```
    date:addToDate(xs:dateTime('SHIP_DATE'), 'MI', 25)
    ```

- **Seconds.** Enter a positive or negative integer in the *amount* argument. Use the SS format string to set the second. The following expression adds 59 seconds to each date in the SHIP_DATE column:

    ```
    date:addToDate(xs:dateTime('SHIP_DATE'), 'SS', 59)
    ```

- **Milliseconds.** Enter a positive or negative integer in the *amount* argument. Use the MS format string to set the milliseconds. The following expression adds 125 milliseconds to each date in the SHIP_DATE column:

    ```
    date:addToDate(xs:dateTime('SHIP_DATE'), 'MS', 125)
    ```

- **Microseconds.** Enter a positive or negative integer in the *amount* argument. Use the US format string to set the microseconds. The following expression adds 2,000 microseconds to each date in the SHIP_DATE column:

    ```
    date:addToDate(xs:dateTime('SHIP_DATE'), 'US', 2000)
    ```

Syntax

```
date:addToDate(xs:dateTime('date'), 'format', amount)
```

Note: You must manually add the xs:dateTime phrase and enclose the date values within single quotation marks.

The following table describes the arguments:

Argument	Required/ Optional	Description
date	Required	Date/Time data type. Passes the values that you want to change. You can enter any valid transformation expression.
format	Required	A format string that specifies the portion of the date value that you want to change. Enclose the format string within single quotation marks, for example, 'mm'. The format string is not case sensitive.
amount	Required	An integer value that specifies the amount of years, months, days, hours, and so on by which you want to change the date value. You can enter any valid transformation expression that evaluates to an integer.

Return Value

Date in the same format as the date you pass to this function.

NULL if a null value is passed as an argument to the function.

Examples

The following expressions all add one month to each date in the DATE_SHIPPED column. If you pass a value that creates a day that does not exist in a particular month, addToDate returns the last day of the month. For example, if you add one month to Jan 31 1998, addToDate returns Feb 28 1998.

Also, addToDate recognizes leap years and adds one month to Jan 29 2000:

```
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'MM', 1)
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'MON', 1)
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'MONTH', 1)
```

The following table shows some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Jan 12 1998 12:00:30AM	Feb 12 1998 12:00:30AM
Jan 31 1998 6:24:45PM	Feb 28 1998 6:24:45PM
Jan 29 2000 5:32:12AM	Feb 29 2000 5:32:12AM *(Leap Year)*
Oct 9 1998 2:30:12PM	Nov 9 1998 2:30:12PM
NULL	NULL

The following expressions subtract 10 days from each date in the DATE_SHIPPED column:

```
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'D', -10)
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'DD', -10)
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'DDD', -10)
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'DY', -10)
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'DAY', -10)
```

The following table shows some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:30AM	Dec 22 1996 12:00AM
Jan 31 1997 6:24:45PM	Jan 21 1997 6:24:45PM
Mar 9 1996 5:32:12AM	Feb 29 1996 5:32:12AM *(Leap Year)*
Oct 9 1997 2:30:12PM	Sep 30 1997 2:30:12PM
Mar 3 1996 5:12:20AM	Feb 22 1996 5:12:20AM
NULL	NULL

The following expressions subtract 15 hours from each date in the DATE_SHIPPED column:

```
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'HH', -15)
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'HH12', -15)
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'HH24', -15)
```

The following table shows some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:30AM	Dec 31 1996 9:00:30AM
Jan 31 1997 6:24:45PM	Jan 31 1997 3:24:45AM
Oct 9 1997 2:30:12PM	Oct 8 1997 11:30:12PM
Mar 3 1996 5:12:20AM	Mar 2 1996 2:12:20PM
Mar 1 1996 5:32:12AM	Feb 29 1996 2:32:12PM *(Leap Year)*
NULL	NULL

adjust-date-to-timezone

Adjusts a date value to a specific time zone, or removes the time zone component from the date value.

Syntax

```
fn:adjust-date-to-timezone(arg, timezone)
```

The following table describes the arguments:

Argument	Required/ Optional	Description
arg	Required	The date value that is to be adjusted. The date-value is of type xs:date or is an empty sequence.
timezone	Optional	A duration that represents the time zone to which date-value is to be adjusted.

Return Value

The fn:adjust-date-to-timezone function behaves differently depending on whether the $arg date value already has a time zone, and on the value of the time zone provided as given below:

- The $timezone argument is expressed as an xs:dayTimeDuration, for example -PT5H for US Eastern Standard Time.
- If $timezone is an empty sequence, returns an xs:date without a time zone. Otherwise, returns an xs:date with a time zone.
- If $arg already has a time zone, its value is adjusted to that time zone.
- To adjust the time zone, an xs:date is treated as an xs:dateTime with time 00:00:00 .
- If $arg does not already have a time zone, its date part stays the same, and the time is associated with the specified time zone.
- If $timezone is omitted from the function call, it is assumed to be the implicit time zone.

Examples

The following table contains examples and return values for the adjust-date-to-timezone function:

Examples	Return Values

adjust-date-to-timezone(
    xs:date('2023-02-15'),
    xs:dayTimeDuration('-PT8H'))
	2023-02-15-08:00

adjust-date-to-timezone(
    xs:date('2023-02-15-03:00'),
    xs:dayTimeDuration('-PT8H'))
	2023-02-14-08:00

adjust-date-to-timezone(xs:date('2023-02-15'), ())	2023-02-15
adjust-date-to-timezone(xs:date('2023-02-15-03:00'), ())	2023-02-15

The following table contains examples and return values assuming an implicit time zone of -05:00:

Examples	Return Values
adjust-date-to-timezone(xs:date('2023-02-15'))	2023-02-15-05:00
adjust-date-to-timezone(xs:date('2023-02-15-03:00'))	2023-02-14-05:00

adjust-dateTime-to-timezone

Adjusts a dateTime value to a specific time zone, or removes the time zone component from the dateTime value.

Syntax

```
fn:adjust-dateTime-to-timezone(arg, timezone)
```

The following table describes the arguments:

Argument	Required/Optional	Description
arg	Required	The dateTime value that is to be adjusted. The dateTime-value is of type xs:dateTime or is an empty sequence.
timezone	Optional	A duration that represents the time zone to which dateTime-value is to be adjusted.

Return Value

The fn:adjust-dateTime-to-timezone function behaves differently depending on whether the $arg date value already has a time zone, and on the value of the time zone provided as given below:

- The $timezone argument is expressed as an xs:dayTimeDuration, for example -PT5H for US Eastern Standard Time.
- If $timezone is an empty sequence, returns an xs:dateTime without a time zone. Otherwise, returns an xs:dateTime with a time zone.
- If $arg already has a time zone, its value is adjusted to that time zone.
- If $arg does not already have a time zone, its date part stays the same, and the time is associated with the specified time zone.
- If $timezone is omitted from the function call, it is assumed to be the implicit time zone.

Examples

The following table contains examples and return values for the adjust-date-to-timezone function:

Examples	Return Values

`adjust-dateTime-to-timezone(`
`    xs:dateTime('2023-02-15T17:00:00'),`
`    xs:dayTimeDuration('-PT7H'))`
	`2023-02-15T17:00:00-07:00`

`adjust-dateTime-to-timezone(`
`    xs:dateTime('2023-02-15T17:00:00-03:00'),`
`    xs:dayTimeDuration('-PT7H'))`
	`2023-02-15T13:00:00-07:00`

`adjust-dateTime-to-timezone(`
`    xs:dateTime('2023-02-15T17:00:00'),`
`    ())`
	`2023-02-15T17:00:00`

`adjust-dateTime-to-timezone(`
`    xs:dateTime('2023-02-15T17:00:00-03:00'),`
`    ())`
	`2023-02-15T17:00:00`

`adjust-dateTime-to-timezone(`
`    xs:dateTime('2023-02-15T01:00:00-03:00'),`
`    xs:dayTimeDuration('-PT7H'))`
	`2023-02-14T21:00:00-07:00`

The following table contains examples and return values assuming an implicit time zone of -05:00:

Examples	Return Values
`adjust-dateTime-to-timezone(` `    xs:dateTime('2023-02-15T17:00:00'))`	`2023-02-15T17:00:00-05:00`
`adjust-dateTime-to-timezone(` `    xs:dateTime('2023-02-15T17:00:00-03:00'))`	`2023-02-15T15:00:00-05:00`

adjust-time-to-timezone

Adjusts a time value to a specific time zone, or removes the time zone component from the date value.

Syntax

```
fn:adjust-time-to-timezone(arg, timezone)
```

The following table describes the arguments:

Argument	Required/ Optional	Description
arg	Required	The time value that is to be adjusted. The time-value is of type xs:time or is an empty sequence.
timezone	Optional	A duration that represents the time zone to which time-value is to be adjusted.

Return Value

The fn:adjust-time-to-timezone function behaves differently depending on whether the $arg date value already has a time zone, and on the value of the time zone provided as given below:

- The $timezone argument is expressed as an xs:dayTimeDuration, for example -PT5H for US Eastern Standard Time.
- If $timezone is an empty sequence, returns an xs:time without a time zone. Otherwise, returns an xs:time with a time zone.
- If $arg already has a time zone, its value is adjusted to that time zone.
- If $arg does not already have a time zone, its date part stays the same, and the time is associated with the specified time zone.
- If $timezone is omitted from the function call, it is assumed to be the implicit time zone.

Examples

The following table contains examples and return values for the adjust-date-to-timezone function:

Examples	Return Values

adjust-time-to-timezone(
    xs:time('17:00:00'),
    xs:dayTimeDuration('-PT7H'))
	17:00:00-07:00

adjust-time-to-timezone(
    xs:time('17:00:00-03:00'),
    xs:dayTimeDuration('-PT7H'))
	13:00:00-07:00

adjust-time-to-timezone(xs:time('17:00:00'), ())	17:00:00
adjust-time-to-timezone(xs:time('17:00:00-03:00'), ())	17:00:00
adjust-time-to-timezone(xs:time('17:00:00'))	17:00:00-05:00

adjust-time-to-timezone(
    xs:time('17:00:00'),
    xs:dayTimeDuration('-PT20H'))
	Error FODT0003

The following table contains examples and return values assuming an implicit time zone of -05:00:

Examples	Return Values
adjust-time-to-timezone(xs:time('17:00:00-03:00'))	15:00:00-05:00
adjust-time-to-timezone(xs:time('22:00:00-08:00'))	01:00:00-05:00
adjust-time-to-timezone(xs:time('01:00:00-02:00'))	22:00:00-05:00

current-date

Returns the current date.

Syntax

```
fn:current-date()
```

Return Value

The fn:current-date function takes no parameters, and returns the current date with the time zone. The time zone is implementation-dependent.

Example

If this function was invoked on January 12, 2023, the returned value would be 2023-01-12Z.

current-dateTime

Returns the current dateTime with time zone.

Syntax

```
fn:current-dateTime()
```

Return Value

The fn:current-dateTime function takes no parameters, and returns the current date and time with the time zone. If the function is called multiple times within the same query, it returns the same value every time. The time zone is implementation-dependent.

Example

If this function was invoked on January 12, 2023 in time zone Z, the returned value would be `2023-01-12T11:25:30.125Z`.

current-time

Returns the current time.

Syntax

```
fn:current-time()
```

Return Value

The fn:current-time function takes no parameters, and returns the current time with the time zone. If the function is called multiple times within the same query, it returns the same value every time. The time zone is implementation-dependent.

Example

The following expression will return `23:17:00.000-05:00` as the result:

```
fn:current-time()
```

dateDiff

Returns the length of time between two dates. You can specify the format as years, months, days, hours, minutes, seconds, milliseconds, or microseconds. The dateDiff function subtracts the second date from the first date and returns the difference.

The dateDiff function calculates the value based on the number of months instead of the number of days. It calculates the date differences for partial months with the days selected in each month. To calculate the date difference for the partial month, dateDiff adds the days used within the month. It then divides the value with the total number of days in the selected month.

The dateDiff function gives a different value for the same period in the leap year period and a non-leap year period. The difference occurs when February is part of the dateDiff function. The dateDiff function divides the days with 29 for February for a leap year and 28 if it is not a leap year.

For example, you want to calculate the number of months from September 13 to February 19. In a leap year period, dateDiff calculates the month of February as 19/29 months or 0.655 months. In a non-leap year period, dateDiff calculates the month of February as 19/28 months or 0.678 months. The dateDiff function similarly calculates the difference in the dates for the remaining months and the dateDiff function returns the total value for the specified period.

Note: Some databases might use a different algorithm to calculate the difference in dates.

Syntax

```
date:dateDiff(xs:dateTime('date'), xs:dateTime('date'), 'format')
```

Note: You must manually add the xs:dateTime phrase and enclose the date values within single quotation marks.

The following table describes the arguments:

Argument	Required/ Optional	Description
date	Required	Date/Time data type. Passes the values for the first date that you want to compare. You can enter any valid transformation expression.
date	Required	Date/Time data type. Passes the values for the second date that you want to compare. You can enter any valid transformation expression.
format	Required	Format string that specifies the date or time measurement. You can specify years, months, days, hours, minutes, seconds, milliseconds, or microseconds. You can specify only one part of the date, such as 'mm'. Enclose the format strings within single quotation marks. The format string is not case sensitive. For example, the format string 'mm' is the same as 'MM', 'Mm', or 'mM'.

Return Value

Double value. If the first date is later than the second date, the return value is a positive number. If the first date is earlier than the second date, the return value is a negative number.

0 if the dates are the same.

NULL if one (or both) of the date values is NULL.

Examples

The following expressions return the number of hours between the DATE_PROMISED and DATE_SHIPPED columns:

```
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'HH')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'HH12')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'HH24')
```

The following table lists some sample values and return values:

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:00AM	Mar 29 1997 12:00:00PM	-2100
Mar 29 1997 12:00:00PM	Jan 1 1997 12:00:00AM	2100
NULL	Dec 10 1997 5:55:10PM	NULL
Dec 10 1997 5:55:10PM	NULL	NULL
Jun 3 1997 1:13:46PM	Aug 23 1996 4:20:16PM	6812.89166666667
Feb 19 2004 12:00:00PM	Feb 19 2005 12:00:00PM	-8784

The following expressions return the number of days between the DATE_PROMISED and the DATE_SHIPPED columns:

```
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'D')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'DD')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'DDD')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'DY')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'DAY')
```

The following table lists some sample values and return values:

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:00AM	Mar 29 1997 12:00:00PM	-87.5
Mar 29 1997 12:00:00PM	Jan 1 1997 12:00:00AM	87.5
NULL	Dec 10 1997 5:55:10PM	NULL
Dec 10 1997 5:55:10PM	NULL	NULL
Jun 3 1997 1:13:46PM	Aug 23 1996 4:20:16PM	283.870486111111
Feb 19 2004 12:00:00PM	Feb 19 2005 12:00:00PM	-366

The following expressions return the number of months between the DATE_PROMISED and DATE_SHIPPED columns:

```
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'MM')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'MON')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'MONTH')
```

The following table lists some sample values and return values:

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:00AM	Mar 29 1997 12:00:00PM	-2.91935483870968
Mar 29 1997 12:00:00PM	Jan 1 1997 12:00:00AM	2.91935483870968
NULL	Dec 10 1997 5:55:10PM	NULL
Dec 10 1997 5:55:10PM	NULL	NULL
Jun 3 1997 1:13:46PM	Aug 23 1996 4:20:16PM	9.3290162037037
Feb 19 2004 12:00:00PM	Feb 19 2005 12:00:00PM	-12

The following expressions return the number of years between the DATE_PROMISED and DATE_SHIPPED columns:

```
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'Y')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'YY')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'YYY')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'YYYY')
```

The following table lists some sample values and return values:

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:00AM	Mar 29 1997 12:00:00PM	-0.24327956989247
Mar 29 1997 12:00:00PM	Jan 1 1997 12:00:00AM	0.24327956989247
NULL	Dec 10 1997 5:55:10PM	NULL
Dec 10 1997 5:55:10PM	NULL	NULL
Jun 3 1997 1:13:46PM	Aug 23 1996 4:20:16PM	0.77741801697531
Feb 19 2004 12:00:00PM	Feb 19 2005 12:00:00PM	-1

The following expressions return the number of months between the DATE_PROMISED and DATE_SHIPPED columns:

```
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'MM')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'MON')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'MONTH')
```

The following table lists some sample values and return values:

DATE_PROMISED	DATE_SHIPPED	LEAP YEAR VALUE (in Months)	NON-LEAP YEAR VALUE (in Months)
Sept 13	Feb 19	-5.237931034	-5.260714286
NULL	Feb 19	NULL	N/A
Sept 13	NULL	NULL	N/A

dateTime

Constructs a dateTime value from a date value and a time value.

Syntax

```
fn:dateTime(arg1, arg2)
```

The `fn:dateTimeAdd (arg1 as xs:date?, arg2 as xs:time?)` is different from the `xs:dateTime` constructor, which accepts a single argument that includes the date and time. Time zone is taken into account when constructing the date/time.

Return Value

- If only one of the arguments has a time zone, or if both arguments have the same time zone, the result has this time zone.
- If neither the date nor the date argument has a time zone, the result has no time zone.
- If the two arguments have different time zones, an error is returned.
- If the combination does not result in a valid time, for example, the day is 32 or the minute is 61, an error is returned.

Example

The following expression returns `2023-06-12T11:35:29` as the result:

```
fn:dateTime((xs:date("2023-06-12")), (xs:time("11:35:29")))
```

day-from-date

Returns the day portion of a date.

Syntax

```
fn:day-from-date(arg)
```

The following table describes the argument for this command:

Argument	Required/ Optional	Description
arg	Required	The date value from which the day component is to be extracted. The date-value is of type xs:date, or is an empty sequence.

Return Value

Returns the day portion from an xs:date value in its localized form, as an integer between 1 and 31 inclusive.

If date-value is an empty sequence, the returned value is an empty sequence.

Example

The following expression returns 15 as the result:

```
fn:day-from-date(xs:date('2023-07-15'))
```

day-from-dateTime

Returns the day portion of a date/time.

Syntax

```
fn:day-from-dateTime(arg)
```

The following table describes the argument for this command:

Argument	Required/ Optional	Description
arg	Required	The dateTime value from which the day component is to be extracted. The dateTime-value is of type xs:dateTime or is an empty sequence.

Return Value

Returns the day portion from an xs:dateTime value in its localized form, as an integer between 1 and 31 inclusive.

If dateTime-value is an empty sequence, the returned value is an empty sequence.

Example

The following expression returns 15 as the result:

```
fn:day-from-dateTime(xs:dateTime('2023-07-15T18:00:00'))
```

days-from-duration

Returns the number of days in a duration.

Syntax

```
fn:days-from-duration(arg)
```

The following table describes the argument for this command:

Argument	Required/Optional	Description
arg	Required	The duration value from which the days component is to be extracted. The duration-value is an empty sequence, or is a value that has one of the following types: - xs:dayTimeDuration - xs:duration - xs:yearMonthDuration

Return Value

- If duration-value is of type xs:dayTimeDuration or is of type xs:duration, the returned value is of type xs:integer, and is the days component of duration-value cast as xs:dayTimeDuration.

- If duration-value is of type xs:yearMonthDuration, the returned value is 0.

- If duration-value is an empty sequence, the returned value is an empty sequence.

- If duration-value is negative, the returned value is negative.

Example

The following expression returns the days from the duration:

```
fn:days-from-duration(xs:dayTimeDuration('DURATION'))
```

The following table lists some sample values and return values:

DURATION	RETURN VALUE
P5D	5
-PT24H	-1
PT23H	0
P1DT36H	2
PT1440M	1

getDatePart

Returns the specified part of a date as an integer value. Therefore, if you create an expression that returns the month portion of the date, and pass a date such as Apr 1 1997 00:00:00, getDatePart returns 4.

Syntax

```
date:getDatePart(xs:dateTime('date'), 'format')
```

Note: You must manually add the xs:dateTime phrase and enclose the date values within single quotation marks.

The following table describes the arguments:

Argument	Required/Optional	Description
date	Required	Date/Time data type. You can enter any valid transformation expression.
format	Required	A format string that specifies the portion of the date value that you want to return. Enclose format strings within single quotation marks, for example, 'mm'. The format string is not case sensitive. For example, if you pass the date Apr 1 1997 to getDatePart, the format strings 'Y', 'YY', 'YYY', or 'YYYY' all return 1997.

Return Value

Integer representing the specified part of the date.

NULL if a value passed to the function is NULL.

Examples

The following expressions return the hour for each date in the DATE_SHIPPED column. 12:00:00AM returns 0 because the default date format is based on the 24 hour interval:

```
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'HH')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'HH12')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'HH24')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Mar 13 1997 12:00:00AM	0
Sep 2 1997 2:00:01AM	2
Aug 22 1997 12:00:00PM	12
June 3 1997 11:30:44PM	23
NULL	NULL

The following expressions return the day for each date in the DATE_SHIPPED column:

```
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'D')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'DD')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'DDD')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'DY')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'DAY')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Mar 13 1997 12:00:00AM	13
June 3 1997 11:30:44PM	3

DATE_SHIPPED	RETURN VALUE
Aug 22 1997 12:00:00PM	22
NULL	NULL

The following expressions return the month for each date in the DATE_SHIPPED column:

```
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'MM')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'MON')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'MONTH')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Mar 13 1997 12:00:00AM	3
June 3 1997 11:30:44PM	6
NULL	NULL

The following expressions return the year for each date in the DATE_SHIPPED column:

```
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'Y')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'YY')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'YYY')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'YYYY')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Mar 13 1997 12:00:00AM	1997
June 3 1997 11:30:44PM	1997
NULL	NULL

getLocale

Returns a string representing the current locale where the process is running.

Syntax

```
date:getLocale()
```

Use the getLocale function to retrieve the current locale setting active within the system. It identifies regional or language specific settings that are used to generate certain values such as language code and a country/region code.

Return Value

Returns the locale value as a text string.

Example

If the process is being executed in the IST time zone, the following expression returns the locale information:

```
date:getLocale()
```

The following table lists some sample properties and values:

PROPERTY	VALUE
Base locale ID	en_IN
Language	en
Country	IN
Display name	English (India)

getTimeZone

Returns a string that specifies the timezone ID where the process is running or where the user is executing it.

Syntax

```
date:getTimeZone()
```

Return Value

Returns the timezone ID as a string value.

Example

If the process is being executed in the IST time zone, the following expression returns
`2023-01-01T12:00:00+00:00` as the result:

```
date:getTimeZone()
```

hours-from-dateTime

Returns the hour portion of a date/time.

Syntax

```
fn:hours-from-dateTime(arg)
```

The following table describes the argument for this command:

Argument	Required/ Optional	Description
arg	Required	The dateTime value from which the hours component is to be extracted. The dateTime-value is of type xs:dateTime or is an empty sequence.

Return Value

Returns the hour portion of an xs:dateTime value in its localized form, as an integer between 0 and 23 inclusive.

If dateTime-value is an empty sequence, the returned value is an empty sequence.

Example

The following expression returns the hour from the date/time:

```
fn:hours-from-dateTime(xs:dateTime('TIME'))
```

The following table lists some sample values and return values:

TIME	RETURN VALUE
2023-08-15T10:30:23	10
2023-08-15T10:30:23-05:00	10

hours-from-duration

Returns the hours in a duration.

Syntax

```
fn:hours-from-duration(arg)
```

The following table describes the argument for this command:

Argument	Required/ Optional	Description
arg	Required	The duration value from which the hours component is to be extracted. The duration-value is an empty sequence or is a value that has one of the following types: - xs:dayTimeDuration - xs:duration - xs:yearMonthDuration

Return Value

- If duration-value is of type xs:dayTimeDuration or is of type xs:duration, the returned value is of type xs:integer, and is a value between -23 and 23, inclusive. The value is the hours component of duration-value cast as xs:dayTimeDuration.

- If duration-value is of type xs:yearMonthDuration, the returned value is of type xs:integer and is 0.

- If duration-value is an empty sequence, the returned value is an empty sequence.

- If duration-value is negative, the returned value is negative.

Example

The following expression returns the hours from the duration:

```
fn:hours-from-duration(xs:dayTimeDuration('DURATION'))
```

The following table lists some sample values and return values:

DURATION	RETURN VALUE
P1DT5H	5
-PT36H	-12
PT23H	23
PT1H90M	2

DURATION	RETURN VALUE
PT2H59M	2
PT3600S	1

hours-from-time

Returns the hour portion of a time.

Syntax

```
fn:hours-from-time(arg)
```

The following table describes the argument for this command:

Argument	Required/ Optional	Description
arg	Required	The time value from which the hours component is to be extracted. The time-value is of type xs:time or is an empty sequence.

Return Value

Returns the hour portion of an xs:time value in its localized form, as an integer between 0 and 23 inclusive.

If time-value is an empty sequence, the returned value is an empty sequence.

Example

The following expression returns the hour from the time:

```
fn:hours-from-time(xs:time('TIME'))
```

The following table lists some sample values and return values:

TIME	RETURN VALUE
10:30:23	10
10:30:23-05:00	10
09:30:00-08:00	9

implicit-timezone

Returns the value of the implicit timezone property.

Syntax

```
fn:implicit-timezone()
```

The implicit time zone is used in comparisons and calculations involving date and time values that do not have explicitly defined time zones. The implicit time zone is defined by the implementation.

Return Value

Returns the implicit time zone as an xs:dayTimeDuration value.

Example

If the implicit time zone is UTC minus 5 hours, the following expression returns `-PT5H` as the result:

```
fn:implicit-timezone()
```

lastDay

Returns the date of the last day of the month for each date in a column.

Syntax

```
date:lastDay(date)
```

The following table describes the argument for this command:

Argument	Required/ Optional	Description
date	Required	Date/Time data type. Passes the date for which you want to return the last day of the month. You can enter any valid transformation expression that evaluates to a date.

Return Value

Date. The last day of the month for the date value that you pass to this function.

NULL if a value in the selected column is NULL.

Example

The following expression returns the current date as the last day:

```
date:lastDay(fn:current-dateTime('DATE'))
```

The following table lists some sample values and return values:

```
DATE                                    RETURN VALUE

18-04-98 01:00                          Apr 18 1998 01:00:00 AM

20-08-99 05:00                          Aug 20 1999 05:00:00 AM
```

The following expression returns the last day of the previous month for each date in the DATE column:

```
date:lastDay(date:addToDate(fn:current-dateTime('DATE','MM',-1))
```

The following table lists some sample values and return values:

```
DATE                    RETURN VALUE

Apr 1 1998 12:00:00AM   Mar 31 1998 12:00:00AM

Jan 6 1998 12:00:00AM   Dec 31 1997 12:00:00AM

Feb 2 1996 12:00:00AM   Jan 31 1996 12:00:00AM

NULL                    NULL
```

You can nest toDate to convert string values to a date. toDate function always includes time information. If you pass a string that does not have a time value, the date returned will include the time 00:00:00.

The following example returns the last day of the month for each date in the same format as the string:

```
date:lastDay(toDate('DATE', 'MON-DD-YYYY'))
```

The following table lists some sample values and return values:

DATE	RETURN VALUE
'18-NOV-98'	Nov-30-1998 00:00:00
'28-APR-98'	Apr-30-1998 00:00:00
NULL	NULL
'18-FEB-96'	Feb-29-1996 00:00:00 *(Leap year)*

```
date:lastDay(date:toDate("DATE", "YYYY-MM-DD"))
```

The following table lists some sample values and return values:

DATE	RETURN VALUE
'18-NOV-98'	1998-Nov-30 00:00:00
'28-APR-98'	1998-Apr-30 00:00:00
NULL	NULL
'18-FEB-96'	1996-Feb-29 00:00:00 *(Leap year)*

millisToDate

Converts the current time from milliseconds.

Syntax

```
date:millisToDate(millis)
```

Argument	Required/ Optional	Description
millis	Required	Numeric data type. Passes the values in milliseconds.

Return Value

The function returns a date that is the number of milliseconds passed since January 1, 1970, 00:00:00 UTC.

Examples

The function returns the corresponding date values for the value mentioned in the MILLISECONDS column:

MILLISECONDS	RETURN VALUE
1000	1970-01-01T00:00:01Z
1900800000	1970-01-23T00:00:00Z

minutes-from-dateTime

Returns an xs:integer value between 0 and 59, both inclusive, representing the minute component in the localized value of $arg.

Syntax

```
fn:minutes-from-dateTime(arg)
```

Argument	Required/ Optional	Description
arg	Required	Numeric data type. Passes the argument as datetime that you want to convert to minutes.

Return Value

If $arg is an empty sequence, the function returns the empty sequence. Otherwise, the function returns an xs:integer value between 0 and 59, both inclusive, representing the minute component in the local value of $arg.

Examples

The function returns the corresponding date values for the value mentioned in the MILLISECONDS column:

MILLISECONDS	RETURN VALUE
1999-05-31T13:40:00-05:00	40
1999-05-31T13:30:00+05:30	30

minutes-from-duration

The function returns an xs:integer representing the minutes component in the value of $arg.

Syntax

```
fn:minutes-from-duration(arg)
```

Argument	Required/ Optional	Description
arg	Required	Numeric data type. Passes the argument as a duration that must be converted to minutes.

Return Value

The function returns an xs:integer representing the minutes component in the value of $arg. The result is obtained by casting $arg to an xs:dayTimeDuration.

Examples

The function returns the corresponding date values for the value mentioned in the DURATION column:

DURATION	RETURN VALUE
P3DT12H32M12S	32

```
DURATION                         RETURN VALUE

-P5DT12H30M                      -30
```

minutes-from-time

Returns an xs:integer value between 0 and 59, both inclusive, representing the minute component in the localized value of $arg.

Syntax

```
fn:minutes-from-time(arg)
```

Argument	Required/ Optional	Description
arg	Required	Numeric data type. Passes the argument as a time that must be converted to minutes.

Return Value

If $arg is an empty sequence, the function returns the empty sequence. Otherwise, the function returns an xs:integer value between 0 and 59, both inclusive, representing the minute component in the local value of $arg.

Examples

The function returns the corresponding date values for the value mentioned in the TIME column:

```
TIME                             RETURN VALUE

11:23:00                         23

13:00:00Z                        0
```

month-from-date

Returns an xs:integer between 1 and 12, both inclusive, representing the month component in the localized value of $arg.

Syntax

```
fn:month-from-date(arg)
```

Argument	Required/ Optional	Description
arg	Required	Numeric data type. Passes the argument as a date that must be converted to months.

Return Value

If $arg is an empty sequence, the function returns the empty sequence. Otherwise, the function returns an xs:integer value between 1 and 12, both inclusive, representing the month component in the local value of $arg.

Examples

The function returns the corresponding date values for the value mentioned in the DATE column:

```
DATE                             RETURN VALUE

1991-05-31-05:00                 05

2000-01-01+05:00                 01
```

month-from-dateTime

Returns an xs:integer between 1 and 12, both inclusive, representing the month component in the localized value of $arg.

Syntax

```
fn:month-from-dateTime(arg)
```

Argument	Required/ Optional	Description
arg	Required	Numeric data type. Passes the argument as a datetime that must be converted to months.

Return Value

If $arg is an empty sequence, the function returns the empty sequence. Otherwise, the function returns an xs:integer value between 1 and 12, both inclusive, representing the month component in the local value of $arg.

Examples

The function returns the corresponding date values for the value mentioned in the DATETIME column:

```
DATETIME                         RETURN VALUE

1995-05-31T13:20:00-05:00        05

1996-12-31T19:20:00-05:00        12
```

months-from-duration

Returns an xs:integer representing the minutes component in the value of $arg . The result is obtained by casting $arg to an xs:dayTimeDuration.

Syntax

```
fn:months-from-duration(arg)
```

Argument	Required/ Optional	Description
arg	Required	Numeric data type. Passes the argument as a duration that must be converted to months.

Return Value

If $arg is an empty sequence, the function returns the empty sequence. Otherwise, the function returns an xs:integer value between 1 and 12, both inclusive, representing the month component in the local value of $arg.

Examples

The function returns the corresponding date values for the value mentioned in the DURATION column:

DURATION	RETURN VALUE
1991-05-31-05:00	05
2000-01-01+05:00	01

seconds-from-dateTime

Returns an xs:decimal value greater than or equal to 0 and less than 60, representing the seconds and fractional seconds in the localized value of $arg.

Syntax

```
fn:seconds-from-dateTime(arg)
```

Argument	Required/ Optional	Description
arg	Required	Numeric data type. Passes the argument as a datetime that must be converted to seconds.

Return Value

If $arg is an empty sequence, the function returns the empty sequence. Otherwise, the function returns an xs:decimal value greater than or equal to 0 and less than 60, representing the seconds and fractional seconds in the local value of $arg.

Examples

The function returns the corresponding date values for the value mentioned in the DATETIME column:

DATETIME	RETURN VALUE
2006-08-15T10:30:14.5	14.5
2001-05-31T13:20:40	40

seconds-from-duration

Returns an xs:decimal representing the seconds component in the value of $arg . The result is obtained by casting $arg to an xs:dayTimeDuration.

Syntax

```
fn:seconds-from-duration(arg)
```

Argument	Required/ Optional	Description
arg	Required	Numeric data type. Passes the argument as a duration that must be converted to seconds.

Return Value

If $arg is an empty sequence, the function returns the empty sequence. Otherwise, the function returns an xs:decimal value greater than or equal to 0 and less than 60, representing the seconds and fractional seconds in the local value of $arg.

Examples

The function returns the corresponding date values for the value mentioned in the DURATION column :

DURATION	RETURN VALUE
PT30.8S	30.8

seconds-from-time

Returns an xs:decimal value greater than or equal to zero and less than 60, representing the seconds and fractional seconds in the localized value of $arg.

Syntax

```
fn:seconds-from-time(arg)
```

Argument	Required/ Optional	Description
arg	Required	Numeric data type. Passes the argument as a time that must be converted to seconds.

Return Value

If $arg is an empty sequence, the function returns the empty sequence. Otherwise, the function returns an xs:decimal value greater than or equal to 0 and less than 60, representing the seconds and fractional seconds in the localized value of $arg.

Examples

The function returns the corresponding date values for the value mentioned as arg in the TIME column :

TIME	RETURN VALUE
13:20:11.5	11.5

timezone-from-date

Returns the timezone component of $arg, if any.

Syntax

```
fn:timezone-from-date(arg)
```

Return Value

If $arg has a timezone component, the result is an xs:dayTimeDuration that indicates deviation from UTC. Its value might range from +14:00 to -14:00 hours, both inclusive. Otherwise, the result is an empty sequence.

Examples

The function returns the corresponding date values for the value mentioned in the DATE column:

DATE	RETURN VALUE
1999-05-31-05:00	-PT5H

timezone-from-dateTime

Returns the timezone component of an xs:dateTime.

Syntax

```
fn:timezone-from-dateTime(arg)
```

Argument	Required/ Optional	Description
arg	Required	Numeric data type. Passes the argument as a datetime that must be converted to a timezone.

Return Value

If $arg is an empty sequence, the function returns the empty sequence. Otherwise, the function returns the timezone component of $arg, if any. If $arg has a timezone component, the result is an xs:dayTimeDuration that indicates deviation from UTC. Its value might range from +14:00 to -14:00 hours, both inclusive. If $arg has no timezone component, the result is the empty sequence.

Examples

The function returns the corresponding date values for the value mentioned in the DATETIME column:

DATETIME	RETURN VALUE
1995-05-31T13:20:00-05:00	-PT5H

timezone-from-time

Returns the timezone component of an xs:time.

Syntax

```
fn:timezone-from-time(arg)
```

Argument	Required/ Optional	Description
arg	Required	Numeric data type. Passes the argument as a time that must be converted to a timezone.

Return Value

If $arg is an empty sequence, the function returns the empty sequence. Otherwise the fucntion returns the timezone component of the xs:time value supplied as argument. The result is an xs:dayTimeDuration that indicates deviation from UTC. Its value might range from +14:00 to -14:00 hours, both inclusive.

Examples

The function returns the corresponding date values for the value mentioned in the TIME column:

```
TIME                            RETURN VALUE

12:00:00-05:00                  -PT5H
```

toChar

Converts the date passed to a string based on the specified format string.

Syntax

```
date:toChar(date, format)
```

Argument	Required/ Optional	Description
arg	Required	Numeric data type. Passes the argument as a date that must be converted to a specified format string.

Examples

The function returns the corresponding date values for the value mentioned as arg in the DATE column:

```
DATE                                            RETURN VALUE

date:toChar(08-02-2017, DL)                     August 02, 2017
```

toDate

Converts a character string to a date data type. You use the toDate format strings to specify the format of the source strings.

The output port must be Date/Time for toDate expressions.

If you are converting two-digit years with toDate, use either the RR or YY format string. Do not use the YYYY format string.

Syntax

```
date:toDate(xs:dateTime('date'), 'format')
```

Note: You must manually add the date phrase and enclose the date values within single quotation marks.

The following table describes the arguments:

Argument	Required/ Optional	Description
date	Required	Must be a string data type. Passes the values that you want to convert to dates. You can enter any valid transformation expression.
format	Required	Enter a valid toDate format string. The format string must match the parts of the *date* argument. For example, if you pass the date 'Mar 15 1998 12:43:10AM', you must use the format string 'MON DD YYYY HH12:MI:SSAM'.

Return Value

Date.

The toDate function always returns a date and time. If you pass a string that does not have a time value, the date returned always includes the time 00:00:00.000000000. You can map the results of this function to any target column with a datetime data type.

NULL if you pass a NULL value to this function.

Warning: The format of the toDate string must match the format string including any date separators. If it does not, toDate might return inaccurate values or skip the record.

Examples

```
date:toDate(xs:dateTime('DATE_PROMISED'), 'MM/DD/YY')
```

The following table lists some sample values and return values:

DATE_PROMISED	RETURN VALUE
'01/22/98'	Jan 22 1998 00:00:00
'05/03/98'	May 3 1998 00:00:00
'11/10/98'	Nov 10 1998 00:00:00
'10/19/98'	Oct 19 1998 00:00:00
NULL	NULL

```
date:toDate(xs:dateTime('DATE_PROMISED'), 'MON DD YYYY HH12:MI:SSAM')
```

The following table lists some sample values and return values:

DATE_PROMISED	RETURN VALUE
'Jan 22 1998 02:14:56PM'	Jan 22 1998 02:14:56PM
'Mar 15 1998 11:11:11AM'	Mar 15 1998 11:11:11AM
'Jun 18 1998 10:10:10PM'	Jun 18 1998 10:10:10PM
'October 19 1998'	*Error. Integration Service skips this row.*
NULL	NULL

The following expression converts strings in the SHIP_DATE_MJD_STRING port to date values:

```
date:toDate(xs:dateTime('SHIP_DATE_MJD_STR'), 'J')
```

The following table lists some sample values and return values:

SHIP_DATE_MJD_STR	RETURN_VALUE
'2451544'	Dec 31 1999 00:00:00.000000000
'2415021'	Jan 1 1900 00:00:00.000000000

Because the J format string does not include the time portion of a date, the return values have the time set to 00:00:00.000000000.

The following expression converts a string to a four-digit year format. The current year is 1998:

```
date:toDate(xs:dateTime('DATE_STR'), 'MM/DD/RR')
```

The following table lists some sample values and return values:

DATE_STR	RETURN VALUE
'04/01/98'	04/01/1998 00:00:00.000000000
'08/17/05'	08/17/2005 00:00:00.000000000

The following expression converts a string to a four-digit year format. The current year is 1998:

```
date:toDate(xs:dateTime('DATE_STR'), 'MM/DD/YY')
```

The following table lists some sample values and return values:

DATE_STR	RETURN VALUE
'04/01/98'	04/01/1998 00:00:00.000000000
'08/17/05'	08/17/1905 00:00:00.000000000

Note: For the second row, RR returns the year 2005 and YY returns the year 1905.

The following expression converts a string to a four-digit year format. The current year is 1998:

```
date:toDate(date('DATE_STR'), 'MM/DD/Y')
```

The following table lists some sample values and return values:

DATE_STR	RETURN VALUE
'04/01/8'	04/01/1998 00:00:00.000000000
'08/17/5'	08/17/1995 00:00:00.000000000

The following expression converts a string to a four-digit year format. The current year is 1998:

```
date:toDate(xs:dateTime('DATE_STR'), 'MM/DD/YYY')
```

The following table lists some sample values and return values:

DATE_STR	RETURN VALUE
'04/01/998'	04/01/1998 00:00:00.000000000
'08/17/995'	08/17/1995 00:00:00.000000000

The following expression converts strings that includes the seconds since midnight to date values:

```
date:toDate(xs:dateTime('DATE_STR'), 'MM/DD/YYYY SSSSS')
```

The following table lists some sample values and return values:

DATE_STR	RETURN_VALUE
'12/31/1999 3783'	12/31/1999 01:02:03
'09/15/1996 86399'	09/15/1996 23:59:59

trunc (Dates)

Truncates dates to a specific year, month, day, hour, minute, second, or millisecond.

You can truncate the following date parts:

- **Year.** If you truncate the year portion of the date, the function returns Jan 1 of the input year with the time set to 00:00:00.000000000. For example, the following expression returns 1/1/1997 00:00:00.000000000:

  ```
  date:trunc(xs:dateTime('12/1/1997 3:10:15'), 'YY')
  ```

- **Month.** If you truncate the month portion of a date, the function returns the first day of the month with the time set to 00:00:00.000000000. For example, the following expression returns 4/1/1997 00:00:00.000000000:

  ```
  date:trunc(xs:dateTime('4/15/1997 12:15:00'), 'MM')
  ```

- **Day.** If you truncate the day portion of a date, the function returns the date with the time set to 00:00:00.000000000. For example, the following expression returns 6/13/1997 00:00:00.000000000:

  ```
  date:trunc(xs:dateTime('6/13/1997 2:30:45'), 'DD')
  ```

- **Hour.** If you truncate the hour portion of a date, the function returns the date with the minutes, seconds, and milliseconds set to 0. For example, the following expression returns 4/1/1997 11:00:00.000000000:

  ```
  date:trunc(xs:dateTime('4/1/1997 11:29:35'), 'HH')
  ```

- **Minute.** If you truncate the minute portion of a date, the function returns the date with the seconds and milliseconds set to 0. For example, the following expression returns 5/22/1997 10:15:00.000000000:

  ```
  date:trunc(xs:dateTime('5/22/1997 10:15:29'), 'MI')
  ```

- **Second.** If you truncate the second portion of a date, the function returns the date with the milliseconds set to 0. For example, the following expression returns 5/22/1997 10:15:29.000000000:

    ```
    date:trunc(xs:dateTime('5/22/1997 10:15:29.135'), 'SS')
    ```
- **Millisecond.** If you truncate the millisecond portion of a date, the function returns the date with the microseconds set to 0. For example, the following expression returns 5/22/1997 10:15:30.135000000:

    ```
    date:trunc(xs:dateTime('5/22/1997 10:15:30.135235'), 'MS')
    ```

Syntax

```
date:trunc(xs:dateTime('date'), 'format')
```

Note: You must manually add the xs:dateTime phrase and enclose the date values within single quotation marks.

The following table describes the arguments:

Argument	Required/ Optional	Description
date	Required	Date/Time data type. The date values that you want to truncate. You can enter any valid transformation expression that evaluates to a date. To pass a NULL value, you must specify an empty sequence in the following format: ()
format	Required	Enter a valid format string. The format string is not case sensitive. To pass a NULL value, you must specify an empty sequence in the following format: ()

Return Value

Date.

NULL if a value passed to the function is NULL.

Examples

The following expressions truncate the year portion of dates in the DATE_SHIPPED column:

```
date:trunc(xs:dateTime('DATE_SHIPPED'), 'Y')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'YY')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'YYY')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'YYYY')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 1 1998 00:00:00.000000000
Apr 19 1998 1:31:20PM	Jan 1 1998 00:00:00.000000000
Jun 20 1998 3:50:04AM	Jan 1 1998 00:00:00.000000000
Dec 20 1998 3:29:55PM	Jan 1 1998 00:00:00.000000000
NULL	NULL

The following expressions truncate the month portion of each date in the DATE_SHIPPED column:

```
date:trunc(xs:dateTime('DATE_SHIPPED'), 'MM')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'MON')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'MONTH')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 1 1998 00:00:00.000000000
Apr 19 1998 1:31:20PM	Apr 1 1998 00:00:00.000000000
Jun 20 1998 3:50:04AM	Jun 1 1998 00:00:00.000000000
Dec 20 1998 3:29:55PM	Dec 1 1998 00:00:00.000000000
NULL	NULL

The following expressions truncate the day portion of each date in the DATE_SHIPPED column:

```
date:trunc(xs:dateTime('DATE_SHIPPED'), 'D')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'DD')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'DDD')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'DY')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'DAY')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 15 1998 00:00:00.000000000
Apr 19 1998 1:31:20PM	Apr 19 1998 00:00:00.000000000
Jun 20 1998 3:50:04AM	Jun 20 1998 00:00:00.000000000
Dec 20 1998 3:29:55PM	Dec 20 1998 00:00:00.000000000
Dec 31 1998 11:59:59PM	Dec 31 1998 00:00:00.000000000
NULL	NULL

The following expressions truncate the hour portion of each date in the DATE_SHIPPED column:

```
date:trunc(xs:dateTime('DATE_SHIPPED'), 'HH')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'HH12')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'HH24')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:31AM	Jan 15 1998 02:00:00.000000000
Apr 19 1998 1:31:20PM	Apr 19 1998 13:00:00.000000000
Jun 20 1998 3:50:04AM	Jun 20 1998 03:00:00.000000000

DATE_SHIPPED	RETURN VALUE
Dec 20 1998 3:29:55PM	Dec 20 1998 15:00:00.000000000
Dec 31 1998 11:59:59PM	Dec 31 1998 23:00:00.000000000
NULL	NULL

The following expression truncates the minute portion of each date in the DATE_SHIPPED column:

```
date:trunc(xs:dateTime('DATE_SHIPPED'), 'MI')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 15 1998 02:10:00.000000000
Apr 19 1998 1:31:20PM	Apr 19 1998 13:31:00.000000000
Jun 20 1998 3:50:04AM	Jun 20 1998 03:50:00.000000000
Dec 20 1998 3:29:55PM	Dec 20 1998 15:29:00.000000000
Dec 31 1998 11:59:59PM	Dec 31 1998 23:59:00.000000000
NULL	NULL

year-from-date

Returns an xs:integer representing the year in the localized value of $arg. The value might be negative.

Syntax

```
fn:year-from-date(arg)
```

Argument	Required/ Optional	Description
arg	Required	Numeric data type. Passes the argument as a date that must be converted to a year format.

Return Value

If $arg is an empty sequence, the function returns the empty sequence. Otherwise, the function returns an xs:integer representing the year in the localized value of $arg.

Examples

The function returns the corresponding year values for the value mentioned as arg in the DATE column:

DATE	RETURN VALUE
1999-06-30	1999
2001-01-01+05:00	2001

year-from-dateTime

Returns an xs:integer representing the year component in the localized value of $arg.

Syntax

```
fn:year-from-dateTime(arg)
```

Argument	Required/ Optional	Description
arg	Required	Numeric data type. Passes the argument as a datetime that must be converted to a year format.

Return Value

If $arg is an empty sequence, the function returns the empty sequence. Otherwise, the function returns an xs:integer representing the year in the localized value of $arg.

Examples

The function returns the corresponding year values for the value mentioned as arg in the DATETIME column:

```
DATETIME                        RETURN VALUE

1999-06-30+08:00                1999

2001-01-01+05:00                2001
```

years-from-duration

Returns an xs:integer representing the years component in the value of $arg. The result is obtained by casting $arg to an xs:yearMonthDuration.

Syntax

```
fn:years-from-duration(arg)
```

Argument	Required/ Optional	Description
arg	Required	Numeric data type. Passes the argument as a duration that must be converted to years.

Return Value

If $arg is an empty sequence, the function returns the empty sequence. Otherwise, the function returns an xs:integer representing the years component in the value of $arg. Given that a duration is a ($months, $seconds) tuple, the result is the value of ($months idiv 12).

Examples

The function returns the corresponding date values for the value mentioned as arg in the DURATION column:

```
DATETIME                        RETURN VALUE

P20Y15M                         21
```

DATETIME	RETURN VALUE
-P15M	-1

Miscellaneous functions

The following functions are available in the Miscellaneous section of the Expression Editor:

aesEncryption

Encrypts the specified data with the provided key using the Advanced Encryption Standard (AES) algorithm.

Syntax

```
util:aesEncryption(key, dataToEncrypt)
```

Return Value

The function returns an encrypted value.

Examples

The following example returns encrypted values for the data and the key passed as inputs in the function:

SAMPLE FUNCTION	OUTPUT
util:aesEncryption("1234@abc", "Informatica")	QETcr60N6QBSTB0X8V4Y+GL\/K0+nt7M6ON0VommGAU4=
util:aesEncryption("abcdefghkl", "cloud")	tceVjWKW2lKWQ+ZJfHn4gUKXPDUPrEilmNoTueesPo0=

aesDecryption

Decrypts the specified data with the provided key using the Advanced Encryption Standard (AES) algorithm. You must use the same key that you had used for encrypting the data in the aesEncryption function.

Syntax

```
util:aesDecryption(key, dataToDecrypt)
```

Return Value

The function returns a decrypted value.

Examples

The following example returns decrypted values for the data and the key passed as inputs in the function:

SAMPLE FUNCTION	OUTPUT
util:aesDecryption("1234@abc", "QETcr60N6QBSTB0X8V4Y+GL\/K0+nt7M6ON0VommGAU4=")	Informatica
util:aesEncryption("abcdefghkl", "tceVjWKW2lKWQ+ZJfHn4gUKXPDUPrEilmNoTueesPo0=")	cloud

decode

Searches a column for a value that you specify. If the function finds the value, it returns a result value, which you define. You can build an unlimited number of searches within a decode function.

If you use decode to search for a value in a string column, you can either trim trailing blank characters with the rtrim function or include the blanks in the search string.

Syntax

```
util:decode(value, search1, result1, args, default)
```

The following table describes the arguments:

Argument	Required/ Optional	Description
value	Required	Passes the values that you want to search. You can enter any valid transformation expression. You can pass any data type except Binary. To pass a NULL value, you must specify an empty sequence in the following format: `()`
search1	Required	Passes the values for which you want to search. You can enter any valid transformation expression. You can pass any value with the same data type as the value argument. The search value must match the value argument. You cannot search for a portion of a value. Also, the search value is case sensitive. For example, if you want to search for the string 'Halogen Flashlight' in a particular column, you must enter 'Halogen Flashlight, not just 'Halogen'. If you enter 'Halogen', the search does not find a matching value. To pass a NULL value, you must specify an empty sequence in the following format: `()`
result1	Required	The value that you want to return if the search finds a matching value. You can enter any valid transformation expression and pass any data type except Binary. To pass a NULL value, you must specify an empty sequence in the following format: `()`
args	Required	Pairs of search values and result values separated by a comma. For example, use the following syntax: `util:decode(value, search1, result1, search2, result2, searchn, resultn, default)` To pass a NULL value, you must specify an empty sequence in the following format: `()`
default	Required	The value that you want to return if the search does not find a matching value. You can enter any valid transformation expression and pass any data type except Binary. To pass a NULL value, you must specify an empty sequence in the following format: `()`

Return Value

result1 if the search finds a matching value.

default value if the search does not find a matching value.

NULL if you omit the default argument and the search does not find a matching value.

Even if multiple conditions are met, decode returns the first matching result.

decode and data types

When you use decode, the data type of the return value is always the same as the data type of the result with the greatest precision.

For example, you have the following expression:

```
util:decode( CONST_NAME
        'Five', 5,
        'Pythagoras', 1.414213562,
        'Archimedes', 3.141592654,
        'Pi', 3.141592654 )
```

The return values in this expression are 5, 1.414213562, and 3.141592654. The first result is an integer, and the other results are decimal. The decimal data type has a greater precision than the integer data type. This expression always writes the result as a decimal value.

You cannot create a decode function with both string and numeric return values.

For example, the following expression is not valid:

```
util:decode( CONST_NAME
        'Five', 5,
        'Pythagoras', '1.414213562',
        'Archimedes', '3.141592654',
        'Pi', 3.141592654 )
```

When you validate the expression above, you receive the following error message:

```
Function cannot resolve operands of ambiguously mismatching datatypes.
```

Examples

You might use decode in an expression that searches for a particular ITEM_ID and returns the ITEM_NAME:

```
util:decode( ITEM_ID, 10, 'Flashlight',
                14, 'Regulator',
                20, 'Knife',
                40, 'Tank',
                'NONE' )
```

The following table lists some sample values and return values:

ITEM_ID	RETURN VALUE
10	Flashlight
14	Regulator
17	NONE
20	Knife
25	NONE
NULL	NONE
40	Tank

The decode function returns the default value of NONE for items 17 and 25 because the search values did not match the ITEM_ID. Also, decode returns NONE for the NULL ITEM_ID.

The following expression tests multiple columns and conditions, evaluated in a top to bottom order for TRUE or FALSE:

```
util:decode( TRUE,
        Var1 = 22, 'Variable 1 was 22!',
        Var2 = 49, 'Variable 2 was 49!',
        Var1 < 23, 'Variable 1 was less than 23.',
        Var2 > 30, 'Variable 2 was more than 30.',
        'Variables were out of desired ranges.')
```

The following table lists some sample values and return values:

Var1	Var2	RETURN VALUE
21	47	Variable 1 was less than 23.
22	49	Variable 1 was 22!
23	49	Variable 2 was 49!
24	27	Variables were out of desired ranges.
25	50	Variable 2 was more than 30.

escape-Json-String

Escapes special characters in the provided string to make it valid for usage in JSON.

Syntax

```
util:escapeJsonString(str)
```

Return Value

Encodes and returns the value as a JSON string according to http://json.org/ rules.

Examples

The function returns the following values for various strings provided as input:

STRING INPUT	RETURN VALUE
//	/\/\
Item1, Items2	Item1, Items2
\n	\\\\n

getAssetLocation

Returns the location where the process or the guide that uses the function is stored.

Note: In addition to guides, you can use the getAssetLocation function only in processes that run on the Cloud Server.

Syntax

```
util:getAssetLocation()
```

Return Value

The function returns the location of the folder where the asset exists.

getAssetName

Returns the name of the process or the guide that uses the function.

Syntax

```
util:getAssetName()
```

Return Value

The function returns the asset name as a string in the output.

getInstanceStartTime

Returns the start time of the running instance of the specified process or guide.

Syntax

```
util:getInstanceStartTime()
```

Return Value

The function returns the start time of the running instance corresponding to the specified asset. The function returns an output in the `yyyy-MM-dd'T'HH:mm:ss.SS'Z'` format.

Examples

The following sample displays the result of the function that is used in a process:
```
2023-09-20T09:02:50.774Z
```

getOrganizationId

Returns the *Informatica Intelligent Cloud Services* organization ID string in the context of the executing instance.

Syntax

```
util:getOrganizationId()
```

Return Value

When the process is deployed on the agent, the function returns the output as $public.

When the process is deployed on the cloud server, the function returns the *Informatica Intelligent Cloud Services* organization ID string as the output.

Examples

The following sample displays the result of the function:
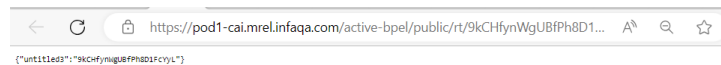

```
{"untitled3":"9kCHfynWgUBfPh8D1FcYyL"}
```

getOrganizationName

Returns the organization name in the context of the executing process, guide, service connector, or data access service connector.

Syntax

```
util:getOrganizationName()
```

Return Value

The function returns the organization name string as the output.

iif

Returns one of two values that you specify based on the results of a condition.

Syntax

```
util:iif(condition, val1 ,val2)
```

The following table describes the arguments:

Argument	Required/ Optional	Description
condition	Required	The condition that you want to evaluate. You can enter any valid transformation expression that evaluates to TRUE or FALSE.
val1	Required	The value that you want to return if the condition is TRUE. The return value is always the data type specified by this argument. You can enter any valid transformation expression, including another iif expression. You can pass any data type except Binary.
val2	Optional	The value that you want to return if the condition is FALSE. You can enter any valid transformation expression, including another iif expression. You can pass any data type except Binary.

The FALSE (*val2*) condition in the iif function is not required. If you omit *val2*, the function returns one of the following values when the condition is FALSE:

- 0 if *val1* is a Numeric data type.
- Empty string if *val1* is a String data type.
- NULL if *val1* is a Date/Time data type.

For example, the following expression does not include a FALSE condition and *val1* is a string data type so decode returns an empty string for each row that evaluates to FALSE:

```
util:iif(SALES > 100, EMP_NAME)
```

The following table lists some sample values and return values:

SALES	EMP_NAME	RETURN VALUE
150	John Smith	John Smith
50	Pierre Bleu	'' *(empty string)*
120	Sally Green	Sally Green
NULL	Greg Jones	'' *(empty string)*

Return Value

val1 if the condition is TRUE.

val2 if the condition is FALSE.

For example, the following expression includes the FALSE condition NULL so decode returns NULL for each row that evaluates to FALSE:

```
util:iif(SALES > 100, EMP_NAME, NULL)
```

The following table lists some sample values and return values:

SALES	EMP_NAME	RETURN VALUE
150	John Smith	John Smith

SALES	EMP_NAME	RETURN VALUE
50	Pierre Bleu	NULL
120	Sally Green	Sally Green
NULL	Greg Jones	NULL

iif and data types

When you use iif, the data type of the return value is the same as the data type of the result with the greatest precision.

For example, you have the following expression:

```
util:iif(SALES < 100, 1, .3333)
```

The TRUE result (1) is an integer and the FALSE result (.3333) is a decimal. The Decimal data type has a greater precision than the Integer data type. Therefore, the data type of the return value is always a decimal value.

Special uses of iif

Use nested iif statements to test multiple conditions. The following example tests for various conditions and returns 0 if sales is 0 or negative:

```
util:iif(SALES > 0, util:iif(SALES < 50, SALARY1, util:iif(SALES < 100, SALARY2,
util:iif( SALES < 200, SALARY3, BONUS))), 0 )
```

in

Matches input data to a list of values. By default, the match is case sensitive.

Syntax

```
util:in(valueToSearch, values, caseFlag)
```

The following table describes the arguments:

Argument	Required/ Optional	Description
valueToSearch	Required	Can be a string, date, or numeric value. Input value that you want to match against a comma-separated list of values. To pass a NULL value, you must specify an empty sequence in the following format: ()
values	Required	Can be a string, date, or numeric value. Comma-separated list of values that you want to search for. Values can be columns. There is no limit to the maximum number of values that you can list. To pass a NULL value, you must specify an empty sequence in the following format: ()
caseFlag	Optional	Must be an integer. Determines whether the arguments in this function are case sensitive. You can enter any valid transformation expression. When *caseFlag* is a number other than 0, the function is case sensitive. When *caseFlag* is a null value or 0, the function is not case sensitive.

Return Value

TRUE (1) if the input value matches the list of values.

FALSE (0) if the input value does not match the list of values.

NULL if the input is a null value.

Example

The following expression determines if the input value is a safety knife, chisel point knife, or medium titanium knife. The input values do not have to match the case of the values in the comma-separated list:

```
util:in(ITEM_NAME, 'Chisel Point Knife', 'Medium Titanium Knife', 'Safety Knife', 0)
```

The following table lists some sample values and return values:

ITEM_NAME	RETURN VALUE
Stabilizing Vest	0 (FALSE)
Safety knife	1 (TRUE)
Medium Titanium knife	1 (TRUE)
	NULL

isNull

Returns whether a value is NULL.

Note: The isNull function evaluates an empty string as FALSE.

Syntax

```
util:isNull(value)
```

The following table describes the argument for this command:

Argument	Required/Optional	Description
value	Required	Passes the rows that you want to evaluate. You can enter any valid transformation expression. You can pass a value of any data type except Binary.

Return Value

TRUE (1) if the value is NULL.

FALSE (0) if the value is not NULL.

Example

The following example checks for NULL values in the items table:

```
util:isNull( ITEM_NAME )
```

The following table lists some sample values and return values:

ITEM_NAME	RETURN VALUE
Flashlight	0 (FALSE)
NULL	1 (TRUE)
Regulator system	0 (FALSE)
''	0 (FALSE) *Empty string is not NULL*

toDecimal

Converts a string or numeric value to a decimal value. The toDecimal function ignores leading spaces.

Syntax

```
util:toDecimal(value, scale)
```

The following table describes the arguments:

Argument	Required/Optional	Description
value	Required	Must be a string or numeric data type. Passes the value that you want to convert to decimals. You can enter any valid transformation expression.
scale	Optional	Must be an integer literal between 0 and 28, inclusive. Specifies the number of digits allowed after the decimal point. If you omit this argument, the function returns a value with the same scale as the input value. This argument is required in advanced mode.

Return Value

Decimal of precision and scale between 0 and 28, inclusive.

0 if the value in the selected column is an empty string or a non-numeric character.

NULL if a value passed to the function is NULL.

Example

This expression uses values from the column IN_TAX. The data type is decimal with precision of 10 and scale of 3:

```
util:toDecimal(IN_TAX, 3)
```

The following table lists some sample values and return values:

IN_TAX	RETURN VALUE
'15.6789'	15.679
'60.2'	60.200
'118.348'	118.348
NULL	NULL
'A12.3Grove'	0

This expression uses values from the column Sales. The data type is decimal with precision of 10 and scale of 2:

```
util:toDecimal(Sales, 2)
```

The following table lists some sample values and return values:

Sales	RETURN VALUE
'1234'	1234
'1234.01'	1234.01

If you want the return value in 1234.00 format, you can use the following expression:

```
format-number(util:toDecimal('1234', 2), '0.00')
```

The toDecimal function support up to 28 precision. If you pass a value with precision greater than 28, such as 32, you will encounter an issue even after enabling high precision.

If you want a 32-digit value intact from the source to the target, you must define the value as string from source to target with 32 precision. You will be able to load the value to the target database even if that database has a numeric data type and the target definition has a string data type. However, you must ensure that the data type of the column in the target database accepts a precision of 32.

Decimal overflow

If the size of the number on the left hand side of the decimal point exceeds the precision, the decimal operation overflows.

To resolve this, modify the scale and/or precision of the expression port and connect downstream ports of the mapping to accommodate the size of the input data in the expression.

For example:

If a numeric field is defined to be size 28 with scale of 15, this accepts 13 numbers on the left side of the decimal and 15 on the right. So the following numbers would be valid for a number of 28,15:

1234567890123.11143

13575.123451234567891

However, these numbers would cause a decimal overflow error:

111112222233333.4444

123.1111122222333334

toInteger

Converts a string or numeric value to an integer. The toInteger syntax contains an optional argument that you can choose to round the number to the nearest integer or truncate the decimal portion. The toInteger function ignores leading spaces.

Syntax

```
util:toInteger(value, flag)
```

The following table describes the arguments:

Argument	Required/ Optional	Description
value	Required	Must be a string or numeric data type. Passes the value that you want to convert to an integer. You can enter any valid transformation expression.
flag	Optional	Specifies whether to truncate or round the decimal portion. The flag must be an integer literal or the constants TRUE or FALSE: - toInteger truncates the decimal portion when the flag is TRUE or a number other than 0. - toInteger rounds the value to the nearest integer if the flag is FALSE or 0 or if you omit this argument.

Return Value

Integer.

NULL if a value passed to the function is NULL.

0 if a value passed to the function contains alphanumeric characters.

Example

The following expressions use values from the column IN_TAX:

```
util:toInteger(IN_TAX, fn:boolean(1))
```

The following table lists some sample values and return values:

IN_TAX	RETURN VALUE
'15.6789'	15
'60.2'	60
'118.348'	118
NULL	NULL
'A12.3Grove'	0
' 123.87'	123

```
IN_TAX                                RETURN VALUE

'-15.6789'                            -15

'-15.23'                              -15
```

If a bigint column is mapped to a column with integer data type, an issue occurs.

To avoid this issue, you must have the data type of the column same throughout the mapping. If you need to assign bigint to the integer column, ensure that the data being passed does not exceed the range of integer.

Bigint range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Integer range: -2,147,483,648 to 2,147,483,647

trace

Provides an execution trace to be used in debugging queries.

The `fn:trace` function accepts items and a label for those items and returns the items unchanged. The exact behavior of the function is implementation-dependent, but generally the processor puts the label and the value of the items in a log file or user console.

Syntax

```
fn:trace(value, label)
```

The following table describes the arguments:

Argument	Required/ Optional	Description
value	Required	The items to trace.
label	Optional	A label to display with the trace information.

Return Value

Based on the items and a label for those items, it returns the items unchanged.

Example

Assume that you have the following variable definition:

let $var1 := 5

The following expression returns `The correct value is: 5` as the result:

```
fn:trace($var1, 'The correct value is: ')
```

trunc (Numbers)

Truncates numbers to a specific digit based on the number of decimal places specified by the precision.

Syntax

```
util:trunc(arg, precision)
```

Argument	Required/ Optional	Description
arg	Required	Numeric data type. Passes the argument that you want to truncate. You can enter any valid expression that evaluates to a numeric data type.
precision	Optional	Can be a positive or negative integer. You can enter any valid expression that evaluates to an integer. The integer specifies the number of digits to truncate.

If *precision* is a positive integer, trunc returns *arg* with the number of decimal places specified by *precision*. If *precision* is a negative integer, trunc changes the specified digits to the left of the decimal point to zeros. If you omit the *precision* argument, trunc truncates the decimal portion of *arg* and returns an integer.

All trailing zeros after the decimal point in an argument will be truncated. For example, the following expression returns 2345.7535 as the result:

```
util:trunc(2345.75350000, 6)
```

If a number contains more than 16-digits after the decimal point, the result returns an exponential value due to an XQuery limitation.
For example, the following expression returns 1.234567812345679E7 as the result:

```
util:trunc(12345678.12345678901234567890, 15)
```

If you pass a decimal *precision* value, the *arg* is rounded to the nearest integer before evaluating the expression.

Return Value

Numeric or integer value based on the parameters provided in the function.

Examples

The following expressions truncate the values in the PRICE column:

```
util:trunc(PRICE, 3)
```

PRICE	RETURN VALUE
12.9995	12.999
-18.8652	-18.865
56.9563	56.956
15.9928	15.992

```
util:trunc(PRICE, -1)
```

PRICE	RETURN VALUE
12.99	10
-187.86	-180

PRICE	RETURN VALUE
56.95	50
1235.99	1230

```
util:trunc(PRICE )
```

PRICE	RETURN VALUE
12.99	12
-18.99	-18
56.95	56
15.99	15

simplifyXml

Rearranges data within XML so that it can be used by the process objects.

Syntax

```
util:simplifyXml(undefined)
```

Return Value

When the process is deployed on the cloud server or agent, the function returns an output without name spaces and convert attributes into child elements.

Examples

The function returns the following output for the XML data provided as input:

INPUT	OUTPUT

`<ns:root xmlns:ns="https://qwerty" attrKey="attrValue">`
 ` data`
 `</ns:root>`
	`<root><attrKey>attrValue</attrKey>`
	 `<text>data</text>`
	 `</root>`

XML Nodes functions

The following XML functions are available in the XML Nodes section of the Expression Editor:

data

The atomic value of a node.

The `fn:data` function takes a sequence of items and returns a sequence of atomic values.

Syntax

```
fn:data(arg)
```

The following table describes the argument:

Argument	Required/ Optional	Description
arg	Required	The items whose typed values are to be returned.

Return Value

For atomic values, it returns the unchanged value. For nodes, it extracts the typed value of the node.

Example

The following expression returns 123, 456 as the result:

```
fn:data((123, 456))
```

Assume that you have the following variable definition:

```
let $x := <hello>Hello

    let $x := <hello>Hello
              <world>World</world>
              </hello>
```

The expression `fn:data($x)` returns the following result:

```
Hello World
```

deep-equal

Assesses whether two nodes have the same content and attributes.

To be deep-equal, they must contain items that are pairwise deep-equal; and for two items to be deep-equal, they must either be atomic values that compare equal, or nodes of the same kind, with the same name, whose children are deep-equal. The $collation argument identifies a collation which is used at all levels of recursion when strings are compared, but not when names are compared.

Syntax

```
fn:deep-equal(parameter1, parameter2, collation)
```

The following table describes the arguments:

Argument	Required/ Optional	Description
parameter1	Required	The first sequence to compare.
parameter2	Required	The second sequence to compare.
collation	Optional	The collation to use for comparing strings. The collation argument identifies the collation to be used at all levels of recursion when strings are compared, but not when names are compared.

Return Value

Returns `true` if both the $parameter1 and $parameter2 sequences contain the same values, in the same order.

Atomic values are compared based on their typed values, using the `eq` operator. If two atomic values cannot be compared, for example, if one is a number, and the other is a string, the function returns `false` rather than raising an error.

Examples

The following expression returns `true` as the result:

```
fn:deep-equal (1, 1)
```

The following expression returns `false` as the result:

```
fn:deep-equal ((1,2), (2,1))
```

The following expression returns `false` as the result:

```
fn:deep-equal (123, John)
```

root

Returns the root of the tree that contains the argument.

Syntax

```
fn:root(arg)
```

The following table describes the argument:

Argument	Required/ Optional	Description
arg	Required	The node whose root node will be returned.

Return Value

The `fn:root` function returns a document node if the `$arg` node is part of a document. However, it returns an element if the `$arg` node is not part of a document.

If `$arg` is the empty sequence, the empty sequence is returned.

If the function is called without an argument, the context item is used as the default argument. If the context item is undefined or is not a node, an error occurs.

Example

Assume that you have the following variable definition:

```
let $in-xml := <a><x>123</x></a>
```

The expression `root($in-xml/x)` returns the following result:

```
<a>
  <x>123</x>
</a>
```

Specifying Functions and Variables

To configure service connectors, you may need to specify variables and functions to define bindings, output fields, and other properties.

Built-in Variables

You can reference these variables in a service connector using an XQuery expression:

Variable	Variable Type	Description
$VariableName	All connection properties	Properties, input and Other Parameters can be specified using this format.
$ResponseStatusCode	Output field mapping	HTTP response code.
$ResponseHeaders	Output field mapping	Contains the HTTP response header in an element list where each item is: <header name="Content-Type">text/plain</header> For example: $ResponseHeaders[@name = "Content-Type"]/text()
$RESTResponse	Output field mapping	Contains the RESTResponse XML data, including the headers and code. Visible in the Test Results for the Service Connector.

Output Field Mapping Functions

When you define bindings in the service connector, you can use many functions in the Expression Editor.

For information on all the available functions, see "Using Functions" on page 29 and "Using the Expression Editor" on page 23.

The following table describes functions available for service connector output field mappings:

Function	Syntax	Description
responseHeaderExists	svc:responseHeaderExists($ResponseHeaders, headerName) : boolean	Returns a Boolean value that indicates if a header parameter exists within the response.
getResponseHeader	svc:getResponseHeader($ResponseHeaders, headerName, defaultValue) : string svc:getResponseHeader($ResponseHeaders, headerName): string	Returns a response header value. If the header parameter is not defined, this function returns the optional default value.
getResponseHeaderNames	svc:getResponseHeaderNames($ResponseHeaders) : list of strings	Returns a list that contains the names of all the parameters within a response header.

Digital Signature Functions Overview

Using the digital signature functions available for Process Designer, you can use XQuery to create processes and service connectors that handle signing of content. The supported functions are either:

- Symmetric, signed using Hash-based Message Authentication Code (HMAC), which is based on a shared password and key, and allows you to use multiple key formats.

- Asymmetric, signed using private keys (for example, PKCS8, PKCS12, Java KeyStore).

One common use for signed digital content is to allow for API authentication in a service connector, which allows you to use hashing and related functions on string-based content. For example, you might need to use signed headers as part of the payload

For more information on each of the functions, see "Using Functions" on page 29.

Note: There are currently no supported functions to verify signed content, or to sign binary content.

HMAC

The HMAC signing method is used with many services, including:

- Amazon Web Services (AWS)
- Twitter OAuth

Process Designer supports HMAC SHA1, HMAC SHA256, and HMAC SHA512 with Base64, Hex, Hex64 or Base64Url encoding. For example, you might use one of these functions to sign content for AWS:

- ```
  dsig:hmacSignature( $data as xs:string, $key as xs:string, $algorithm as xs:string,
  $encoding as xs:string? ) as xs:string
  ```

- ```
  dsig:hmacSHA1signature( $data as xs:string, $key as xs:string, $encoding as xs:string? )
  as xs:string
  ```

- ```
  dsig:hmacSHA256signature( $data as xs:string, $key as xs:string, $encoding as xs:string? )
  as xs:string
  ```

- ```
  dsig:hmacSHA512signature ( $data as xs:string, $key as xs:string, $encoding as xs:string?)
  as xs:string
  ```

AWS Authentication

The AWS REST API requires the authentication header in this format:

Authorization: AWS AWSAccessKeyId:Signature

where:

Signature = dsig:hmacSHA1signature($strToSign, "*AwsSecretKeyId*")

The AWSAccessKey and AwsSecretKeyId are generated by AWS and accessible through the AWS Developer Console.

Note: $strToSign is combination of HTTP header values and other parameters.

For more information on REST authentication for AWS, see:
http://docs.aws.amazon.com/AmazonS3/latest/dev/RESTAuthentication.html

Asymmetric Private Key-Based Signing

The following private key methods are supported:

- PKCS8
- PKCS12
- Java KeyStore

Note: If using a private key to sign content, the keystore file must reside on a Secure Agent and the process or service connector must run on the Secure Agent in order to access the key. The file path to the key must be specified in the object properties (for example, "C:/certs/mykey.p12").

You can also:

- Deploy the PKCS8/12 artifact on an agent contribution and specify its location using the 'project:/' scheme.
- Supply the key using encoded key content instead of a file location, which may be useful during development of a service connector. For example, you can paste the encoded content into a text area for test purposes.

PKCS8 Key File

To sign with the PKCS8 key file, use this function:

dsig:signWithKeyFile($messageToSign as xs:string, $pathToKey as xs:string, $encryptionAlgorithm as xs:string, $digestAlgorithm as xs:string) as xs:string

The key file can be binary or in Base64-encoded format.

P12 and Keystore Based Files

To sign with a P12 or KeyStore-based file, use this function:

dsig:signWithCertificate($messageToSign as xs:string, $pathToCertificate as xs:string, $keyRecoveryPassword as xs:string, $encryptionAlgorithm as xs:string, $digestAlgorithm as xs:string) as xs:string

For example, you can use this method for Google JWT (JSON Web Token) Authentication for service accounts. After you obtain a Google private key for access and save it on your Secure Agent, you create a JWT request and sign it with your private key.

For more information on JWT authentication, see: https://console.developers.google.com and https://developers.google.com/identity/protocols/OAuth2ServiceAccount.

hmacSHA256signatureForList

Calculates a hash-based message authentication code (HMAC) using the SHA256 algorithm and the optional encoding where $encoding is one of the following values: Base64 (default), Base64Url, Hex, or Hex64.

Syntax

```
dsig:hmacSHA256signatureForList(data, delimiter, secretKey, encoding)
```

The following table describes the arguments:

Argument	Required/ Optional	Description
data	Required	Plain text to compute hash.
delimiter	Required	A sequence of one or more characters based on which you want to split the string.

Argument	Required/ Optional	Description
secretKey	Required	The secret key for HMAC computation. The key can be of any length. However, the recommended size is 64 bytes. If the key is more than 64 bytes long, it is hashed using SHA-256 to derive a 32-byte key.
encoding	Required	Type of encoding used for the output. For example, you can calculate HMAC using the SHA256 algorithm where encoding is one of the following values: Base64 (default), Base64Url, Hex, or Hex64.

Attachments

Process Designer exposes an endpoint so you can access attachments in a process using an event handler, internal API calls, custom service connectors or built-in connectors like the Amazon S3 Connector. This enables you to pass through an attachment and extract data about the attachment like the file size and file name using functions on the response (where supported).

When you design a process, you can pass attachments into a subprocess, service connector, or Service steps or return attachments (where the output field is defined using the Attachment type).

In a process, you can allow the user to add one or more attachments as inputs where the attachment is coming directly from the screen (not carried across steps using process fields). For example, you could use a process to capture real property data and allow the user to attach photos of the associated property. However, you cannot access the attachment data within a process.

Note: If you run a process on the cloud server, do not use an attachment whose size is more than the default 5,242,880 bytes. The cloud server cannot process attachments that are greater than 5,242,880 bytes.

Encoding

Attachments can be encoded as:

- multipart/form-data (for browser-based interactions)
- multipart/mixed

Data Types

You can specify named input parameters as attachments using two Simple data types:

- **Attachment** supports single attachments and allows you to define the maximum file size as a field property.
- **Attachments** supports multiple attachments and allows you to define the maximum file size and maximum number of files as field properties.

From within the process, you can access metadata about the attachment using an XQuery formula and built-in functions. For example, the following returns the size of the 'photo' input:

```
sff:getAttachmentSize($input.photo)
```

You can assign input attachment fields to outputs or temp fields using an Assignment step. In that case, be sure that the target temp or output field is also of type Attachment.

In this example, the second parameter ($encodedContent) is a base64-encoded string:

```
sff:createAttachmentFromBase64("base64", "UOpGUIRA", "ascii")
```

Note: The attachment functions are available in processes but not in service connectors. See below for more information on handling attachments in service connectors.

Refer to "Using Functions" on page 29 for more information on the functions available for use with attachments.

Attachment Payload

When using simple attachments, be sure that:

- Each named part has the same name (name and Content-Disposition) as the Input parameter in the process.

- The Content-Type of the part must also match the content type of the Input parameter. For all data types except attachments, Content-Type= text/plain (for example, text, checkbox, date).

- If the part represents a process object, its contents should be valid, serialized JSON content (with a Content-Type of application/json).

- You define process output field (s) of type Attachment (or Attachments). These fields are returned as attachments to the caller.
  For example, if the process has input fields "first" (Text), "last" (Text), and "inputFile" (Attachment), the payload would be similar to the following:

```
POST /active-bpel/public/rt/Attachments_Test HTTP/1.1
Content-Type: multipart/form-data; boundary=----TheBoundary1234
------TheBoundary1234
Content-Disposition: form-data; name="first"
John
------TheBoundary1234
Content-Disposition: form-data; name="last"
Smith
 ------TheBoundary1234
Content-Disposition: form-data; name="inputFile"; filename="filename.png"
Content-Type: image/png
 ... binary image data ...
------TheBoundary1234--
```

The first part of the response should be a JSON document describing the response output fields. The JSON content will have the output field values (standard response).

If the output field is of type Attachment, the value will be a string containing the content-id (cid) of the part containing the attachment. For example, if the output fields are the "first" (Text), "last" (Text), and "inputFile" (Attachment), the response should look similar to:

```
Content-Type: multipart/mixed; boundary=----TheBoundary1234

------TheBoundary1234
Content-Type: application/json
{
"first" : "John",
"last" : "Smith",
"inputFile" : "cid:fc1c6030-4b90-4981-b9d8-ab1d0ba0e84e"
}

------TheBoundary1234
Content-Type: image/png
Content-Name: photo
Attachment-Created-At:1435182142837
Content-Id: fc1c6030-4b90-4981-b9d8-ab1d0ba0e84e
Content-Length: 1073
... binary image data ...

------TheBoundary1234--
```

- If you do not enable **Output field is whole payload** in the process properties and you have a single output field of type attachment, the response contains both the attachment data and the metadata, as shown here:

```
--Boundary
content-type: ...
```

```
cid:outputFile

--Boundary
content-type: image/jpeg

[binary data]
```

If you enable **Output field is whole payload** and you have exactly one output field of type Attachment, you can receive a single attachment (of type text or binary) without the metadata. If a third-party wants to access only the attachment, it is easier to consume. For example:

```
HTTP/1.1 200 OK
Content-Type: image/jpeg
Content-Length: length

[.. JPEG data ...]
```

Using Attachments in Service Connectors

You can define a service connector in order to submit an attachment from a process to a service connector's input or obtain an attachment from a service connector and submit it to a process as an attachment.

Attachments are sent directly to the service input (including Salesforce-based services) from processes and service connectors.

Note: You upload or download content from services like Amazon S3 and propagate the attachment back to the main process under the name defined in the output mapping. With the Amazon S3 Connector, you can read or write the contents of an S3 object both as a Base64-encoded string and as an attachment. For more information, see the *Amazon S3 Connector Guide*.

Using Multiple Output Attachments in Service Connectors

A service connector can handle a multipart response appropriately and create multiple attachments for different response parts. The part related metadata information is available in the attachment properties.

A service connector can also recognize the payload part. The first JSON/XML/Text part is used as the payload. You can extract data from the payload in the output field expressions of the action as you would normally do when you work with non-multipart responses. Note that the selected payload will not be available as an attachment.

In the **Output** tab, you can use the **Get from Attachments** option to work with multiple attachments and pass the entire list of attachments to the selected variable except the payload.

You can also use an expression to work with attachments in service connectors. When you open the Expression Editor, you can find the new functions in the **Functions** tab, under the **Output Attachments** section.

Using GET, POST, and PUT

You can define a service connector to use GET, POST, or PUT operations on a resource that does not support JSON or XML.

Note that:

- The input parameters (Attachment type) must be uploaded to the destination from the binding URL (POST/PUT). Use a payload format based on the binding type and number of attachments.
- If there are multiple attachment parameters and the binding is FORM, send all the parameters (including non-file parameters) as multipart/form-data.
- If there are one or more attachment parameters and the binding is JSON, send files as multipart/form-data and combine non-file parameters into a single JSON element.
- If there is exactly one file parameter, the binding is CUSTOM, and the binding payload is empty (no XML or JSON in a custom textarea), POST or PUT the raw file data as is.

- If there are multiple attachment parameters and the binding is CUSTOM and the payload is XQuery, the HTTP outbound payload should be multipart/related. The first part is the CUSTOM (XQuery) content, followed by parts for each of the file/attachment input parameters.
- You can assign input attachment fields to outputs or temp fields using an Assignment step. In that case, be sure that the target temp or output field is also of type Attachment.
- Downloading multiple files (using multipart/mime attachment) is not supported.

Note: The "sff" functions are available only in processes.

HTTP and JMS Headers

Use header functions to get details about headers that you pass when you invoke a process. You can get details about process headers and message event headers.

Use header functions to get information that is only available in the header that you pass when you invoke a process. Examples include dates, JSON Web Tokens, and security IDs.

You can assign the header value to a field and, optionally, perform XQuery operations on the value.

The following image shows the header functions available under the **Request Context** section of the Expression Editor:
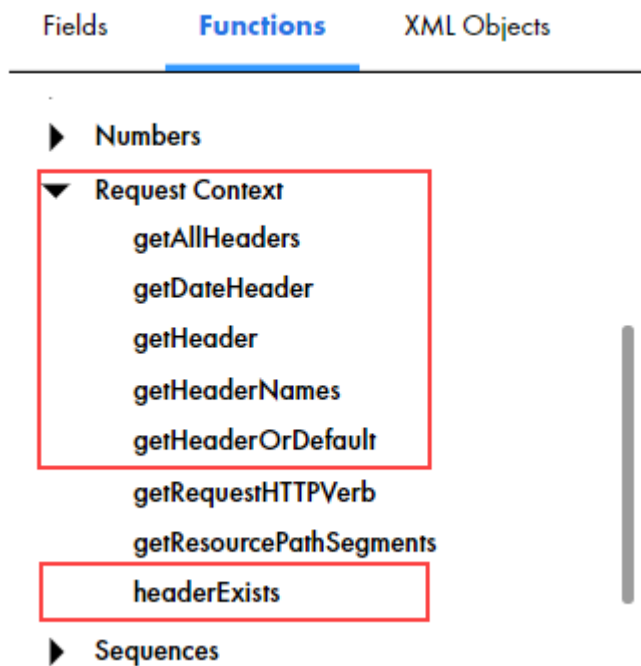
To understand how each function works, consider a process invoked with the following headers and values:

Header	Value
accept-language	en-US;q=0.9
client-ip	192.0.2.1

Header	Value
accept	text/xml
postman-token	0023-a5dfd-8vdg3-n2b3
CustomHeader	This is a custom header
ExecutionDate	Wed, July 25 2018 06:25:24 GMT

You can use the following header functions on their own, or along with other functions:

Get All Header Values

Use the **getAllHeaders** function to get a list of all headers and their values.

For example, if you use the following expression, you get a specific output:

Expression:

```
fn:getAllHeaders()
```

Output:

```
<headers>
 <header name="accept-language">en-US;q=0.9</header>
 <header name="client-ip">192.0.2.1</header>
 <header name="accept">text/xml</header>
 <header name="postman-token">0023-a5dfd-8vdg3-n2b3</header>
 <header name="CustomHeader">This is a custom header</header>
 <header name="ExecutionDate">Wed, July 25 2018 06:25:24 GMT</header>
</headers>
```

Get Date Headers

Use the `getDateHeader` function to get the value of a Date header. The function returns data in the dateTime type.

You can use other Date and Time functions to parse the value.

For example, if you use the following expression, you get a specific output:

Expression:

```
fn:year-from-dateTime(request:getDateHeader("ExecutionDate")
```

Output:

```
2018
```

Get a Specific Header Value

Use the `getHeader` function to get the value of a specific header.

For example, if you use the following expression, you get a specific output:

Expression:

```
request:getHeader("accept")
```

Output:

```
text/xml
```

Get a list of Headers Names

Use the `getHeaderNames` function to get a sequence of header names without values.

For example, if you use the following expression, you get a specific output:

Expression:

```
request:getHeaderNames()
```

Output:

```
accept-language,client-ip,accept,postman-token,CustomHeader,ExecutionDate
```

Get a Header value or 'Default'

Use the `getHeaderOrDefault` to get a header value if the header exists, or some default value if the header does not exist.

To use this function, you must give the header name and a default value as expression parameters.

Let the default value be `Not Available`.

For example, if you use the following expression, you get a valid output:

Expression:

```
concat ("The header I want is",
                    request:getHeaderOrDefault("postman-token","Not Avaialble"    )
```

Output:

```
The header I want is 0023-a5dfd-8vdg3-n2b3
```

If you use the following expression, you get a Not Available output:

Expression:

```
concat ("The header I want is",
                    request:getHeaderOrDefault("special-token","Not Avaialble"    )
```

Output:

```
The header I want is Not Available
```

This is because the request does not contain a header with the name `special-token`.

Check if a Specific Header Exists

Use the `headerExists` function to check if a header with a specific name exists.

You get either `True` or `False` in the output.

For example, if you use the following expression, you get a specific output:

Expression:

```
request:headerExists("CustomHeader")
```

Output:

```
True
```

Headers from Message Events

If a process has message events, you can use any header functions to get headers from the message events. To do this, you must add the case-sensitive message event name as an expression parameter.

For example, if you pass a header with the name `CustomHeader` to the message event receive step, use the following expression to get the value of `CustomHeader` from the message event `ME1`.

```
request:getHeader("CustomHeader","ME1")
```

HTTP verb functions

Use HTTP verb functions to retrieve the HTTP verb and the resource path segments that are used in a request. The HTTP verb functions are available under the **Request Context** section of the Expression Editor.

You can use the following HTTP verb functions:

getRequestHTTPVerb

Use the `getRequestHTTPVerb` function to determine the HTTP verb that is used in a request. The function retrieves the HTTP verb from the request and returns one of the following responses in a string format:

- GET
- POST
- PATCH
- PUT
- DELETE

getResourcePathSegments

Use the `getResourcePathSegments` function to retrieve all or specific resource path segments of REST requests. The function returns the values as a string of tokens.

For example, consider the following request URL:

```
https://na1.ai.dm-us.informaticacloud.com/active-bpel/rt/InitiateOrder/Orders/
OrderID_100/quantity/5
```

If you use the `request:getResourcePathSegments()` expression, you see the following response:

```
[Orders, OrderID_100, quantity, 5]
```

To retrieve a specific resource path segment, use a numeric qualifier to denote the position of the resource path segment from the process name.

For example, to retrieve the `OrderID_100` segment alone in the response, use the following expression:

```
request:getResourcePathSegments()[2]
```

Note: You cannot use the `getResourcePathSegments` function to fetch the resource path segments of SOAP requests and message events. You cannot use the `getResourcePathSegments` function to fetch the host context.

Human task XQuery utilities

If you use a Human Task step in a process, you can use human task utility functions when working with XQuery in Process Designer or in any field that uses a formula to set a value.

The human task utility functions are available under the **Human Task Utilities** section of the Expression Editor.

The following table describes the human task utility functions:

Function	Syntax	Description
getHumanTaskId	`task:getHumanTaskId(humanTaskStepName)`	Returns the ID of the human task associated with the specified Human Task step name. You can use the ID for other actions such as getting information about a human task, reassigning a task, or canceling a task.
getHumanTaskInfo	`task:getHumanTaskInfo(humanTaskId)`	Returns the following information about the human task based on the specified human task ID: - Task ID - Task name - Status - Priority - Task initiator - Stakeholders - Potential owners - Administrators - Task owner - Task creation date - User who created the task - Task activation time - Task expiration time - Skippable option value

reassignHumanTask	`task:reassignHumanTask(humanTaskId, users, groups)`	Reassigns a human task to the specified users and groups, and returns the task ID.
		
		If you reassign the task to a single user, the user becomes the task owner. If you reassign the task to a group or multiple users, a user must claim the task to become the task owner.
		
		To specify multiple users or groups, use a comma character as the delimiter. For example, use the following syntax:
		
		`task:reassignHumanTask(humanTaskId, ('<user1>','<user2>'), ('<group1>','<group2>'))`
		
		If you want to reassign the task to a specific user and not a group, use the following syntax:
		
		`task:reassignHumanTask(humanTaskId, '<user1>','')`

cancelHumanTask	`task:cancelHumanTask(humanTaskId)`	Cancels a human task and makes the task obsolete. Returns the task ID. The process continues to the next step after a human task is canceled.

Creating a field with a list of simple type

In Application Integration assets such as process, process object, service connector, and guide, you can use the following data types for input fields, output fields, temp fields, and messages:

- List of checkbox
- List of date
- List of date time
- List of integer
- List of number
- List of text
- List of time

When you create a process, process object, service connector, or guide, you can define a comma-separated list of values.

Note: Messages are applicable only for processes.

To create a field with a list of simple type, perform the following steps:

1. On the **Explore** page, navigate to the asset for which you want to create a field with a list of simple type.
2. Add steps to the asset.
3. Click the **Start** step.
4. Click the **Input Fields**, **Output Fields**, **Temp Fields**, or **Messages** tab as required.
5. Click the **Add** icon.
6. In the **Type** column, select **More types**.

   The **Edit Type** dialog box appears.
   The following image shows the **Edit Type** dialog box:
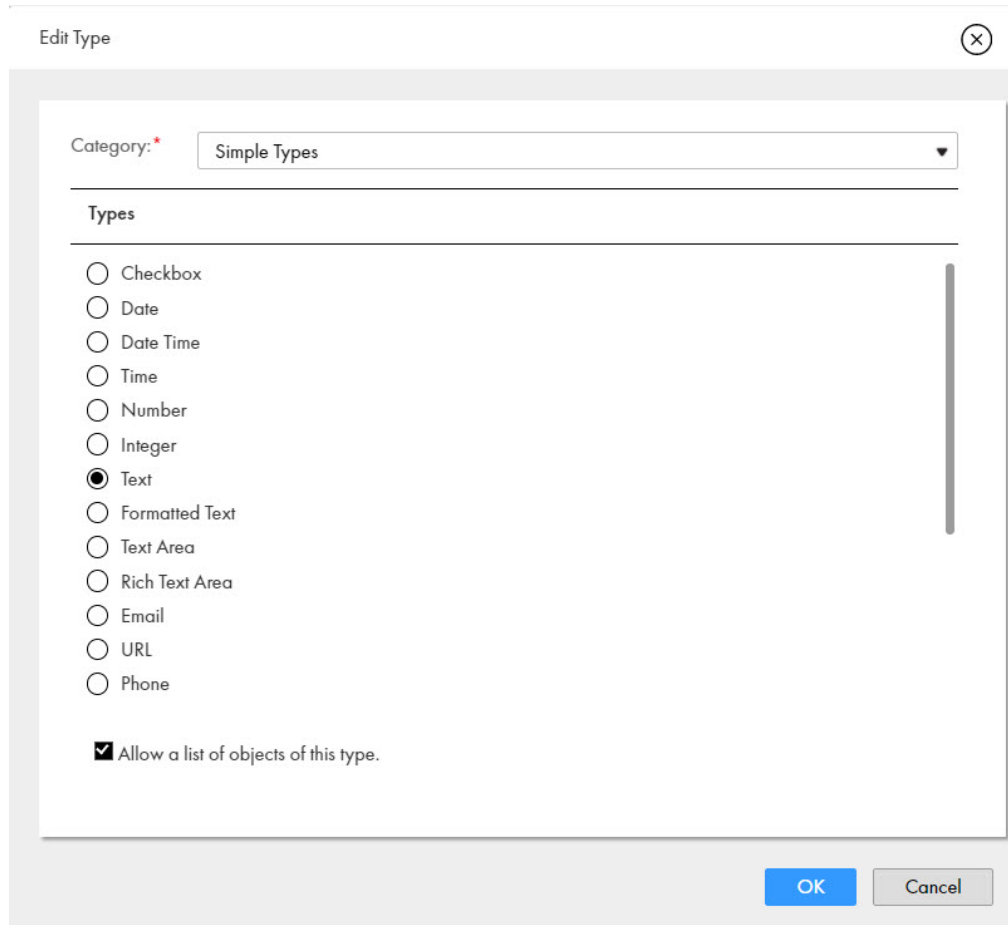
7. From the **Category** list, select **Simple Types**.

   A list of data types appear in the **Types** section.

8. Select the **Checkbox**, **Date**, **Date Time**, **Integer**, **Number**, **Text**, or **Time** type as required, and then select **Allow a list of objects of this type**.

9. Click **OK**.

   Based on the selected type, the field is set to the corresponding list of simple type.

Invoking processes with a list of simple type

When you invoke a process, the format to specify the list of values differs based on the type of HTTP verb and the payload format. You can provide a list of checkbox, date, date time, time, number, integer, and text values.

In a GET request, provide a list of check box, date, date time, time, number, integer, and text values separated by an ampersand (&) in the URL.

For example:

```
https://na1.ai.dm-us.informaticacloud.com/active-bpel/public/rt/3TbGuTUy0cchqLy0IXoXsn/
OrderManagementProcess?
```

```
taskIds=TASK-123&taskIds=TASK-456&itemcount=5&itemcount=6&listdatetime=2022-08-24T21:32:52&li
stboolean=true&listboolean=false
```

In a POST request, if you use JSON, provide an array of comma-separated values.

For example, use the following text for JSON:

```
{
    "taskIds": ["TASK-123", "TASK-456", "TASK-789" ]
    "itemcount": [5,6]
    "listdatetime": ["2022-08-24T21:32:52", "2022-08-25T22:32:52"]
    "listboolean": [true,false]
}
```

In a POST request, if you use XML, provide text in the XML format.

For example, use the following text for XML:

```
<taskIds>TASK-123</taskIds>
<taskIds>TASK-456</taskIds>
<itemcount>5</itemcount>
<itemcount>6</itemcount>
<listdatetime>2022-08-24T21:32:52</listdatetime>
<listdatetime>2022-08-25T22:32:52</listdatetime>
<listboolean>true</listboolean>
```

In a SOAP request, provide values in the XML format similar to a POST request.

Creating a field with the custom type

You can use a process object to create a field with the custom type. You can also create a list of process object types. When you create a process or guide, you can use the process object data type for input fields, output fields, temp fields, and messages.

Note: Messages are applicable only for processes.

To create a field with the custom type, perform the following steps:

1. On the **Explore** page, navigate to the process or guide for which you want to create a field with the custom type.

2. Add steps to the process or guide.

3. Click the **Start** step.

4. Click the **Input Fields**, **Output Fields**, **Temp Fields**, or **Messages** tab as required.

5. Click the **Add** icon.

6. In the **Type** column, select **More types**.

    The **Edit Type** dialog box appears.

7. From the **Category** list, select **Custom Types**.

    A list of process objects appears in the **Types** section.
    You can enter the process object name to search for the process object.

    The following image shows a list of available process objects:
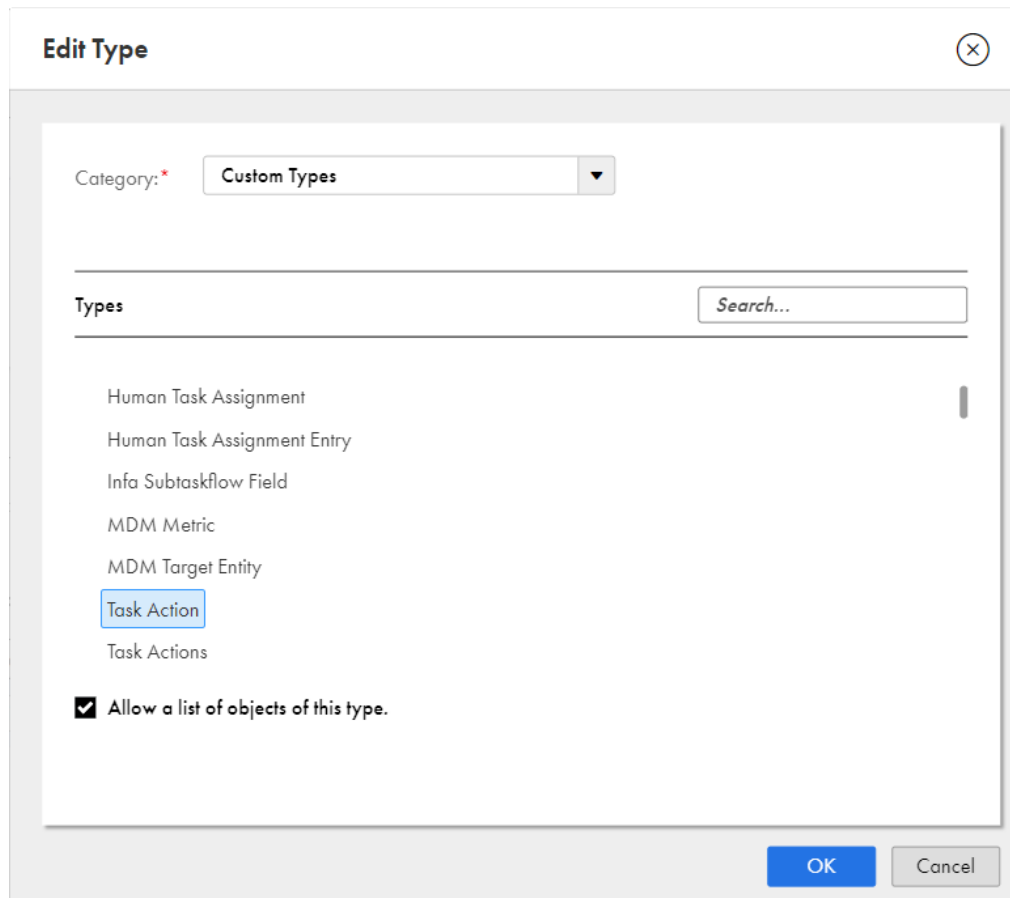
8. Select the process object as required. To use a list of process objects, select **Allow a list of objects of this type**.

9. Click **OK**.

   Based on the selected process object, the field is set to the process object type or a list of process object type.

Creating a field with the connection defined type

You can use an object associated with an app connection to create a field using the connection defined type. You can also create a list of object types. When you create a process or guide, you can use the object data type for input fields, output fields, temp fields, and messages.

Note: Messages are applicable only for processes.

To create a field with the connection defined type, perform the following steps:

1. On the **Explore** page, navigate to the process or guide for which you want to create a field with the connection defined type.

2. Add steps to the process or guide.

3. Click the **Start** step.

4. Click the **Input Fields**, **Output Fields**, **Temp Fields**, or **Messages** tab as required.

5. Click the **Add** icon.

6. In the **Type** column, select **More types**.

   The **Edit Type** dialog box appears.

7. From the **Category** list, select **Connection defined Types**.

8. From the **Connection** list, select the connection.

   A list of objects available in the connection appears in the **Types** section.
   The following image shows the objects associated with a connection:

9. Select the required object. To use a list of objects, select **Allow a list of objects of this type**.

10. Click **OK**.

    Based on the selected object, the field is set to the object type or a list of object type.

    Note: The Field can be one of several types option is reserved for future use.

CHAPTER 2

Designing Processes

When you use Process Designer to integrate applications, you can:

- Build a process from a variety of step types, including data decision, subprocess, parallel path, receive, and wait.
- Automatically link steps in a process as you select data and define actions, bindings, and other process properties.
- Test your process and necessary connections during process design.
- Publish the process so it is automatically deployed and available to invoke as a REST/XML or JSON service.

For example, you might define a process to streamline Salesforce order processing as follows:

- Invoke services to obtain shipping information.
- Lookup current pricing from an inventory database that resides on-premises.
- Submit order details to SAP.

The following image shows how to design a process:
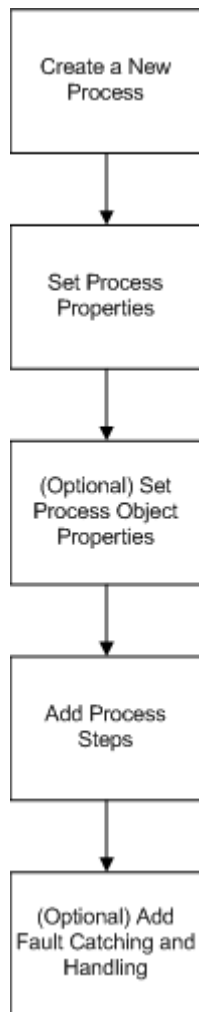
Creating a Process

1. In Application Integration, select **New**.
2. In the **New Asset** dialog box, select **Processes** > **Process**, and then click **Create**.

Setting Process Properties

1. In Application Integration, click **New** to open the **New Asset** window.
2. Select **Processes** > **Process**, and then click **Create**.

   The Process Designer opens.

   The following image shows the **Start Properties** panel:

3. Enter the required properties.

4. Optionally, to view or edit the process properties, click the **Start** step.

5. Click **Save**.

General Properties

You can specify the following General properties for processes:

Property	Description
Name	Required. A descriptive name to identify the process in Process Designer and the name that appears when the process is available for use in other objects such as subprocess steps. The name can contain multibyte characters and must not exceed 80 characters. To change the name of a published process, you must first unpublish the process. Then, change the name and republish the process.
Override API Name	Optional. Overrides the API name that is auto generated when you publish the process with a name that you specify. When you select this option, the **API Name** field becomes available.
API Name	Required if you select the **Override API Name** option. A unique API name to override the auto-generated API name for the process. The API name that you specify in this field is used in the generated service URLs. The API name can contain only alphanumeric characters, underscores (_), and hyphens (-), and must not exceed 80 characters. To change the API name of a published process, you must first unpublish the process. Then, change the API name and republish the process. If you override the API name and import the process, the imported process uses the API name that you specified. However, if there is an existing process with the same API name that you specified, Application Integration uses an auto-generated API name for the imported process to avoid duplication. Application Integration also disables the **Override API Name** field. Similarly, when you copy a process, Application Integration uses an auto-generated API name for the copied process to avoid duplication and also disables the **Override API Name** field. If you override the API name and move the process to a different location, Application Integration retains the API name that you specified.

Property	Description
Location	Optional. The location of the project or folder you want the process to reside in. To select a location for the process, browse to the appropriate project or folder, or use the default location.
Description	Optional. A description of the process.

Start Properties

Use the **Start** tab to define the binding type, access details, and the runtime environment details.

You can define the following start properties for a process:

Binding

The **Binding** property defines how a process is invoked and run.

You can select one of the following values:

- **REST/SOAP**: Select the **REST/SOAP** binding type if you want to run the process by using a service URL. When you select this option, you can use the **Allowed Groups** and the **Allowed Users** fields to define the user groups and users who can run a published process as an API. If you do not want to assign any user, you can select the **Allow anonymous access** option.

- **Event**: Select the **Event** binding type if you want the process to be invoked when the specified event occurs. For example, a process can be invoked upon an event such as arrival of a file in a file system. The **Event Source Name** field is available where you can select an event source.

Default is REST/SOAP.

Allowed Groups

The groups that have access to the process service URL at run time. If the users in the group also have the Run privilege, they can invoke the process service URL.

Use the **Allowed Groups** field when you want a group of users to have access to a service URL. For example, you have a group called 'Order Approvers'. If you enter **Order Approvers** in the **Allowed Groups** field, all users within the group will have access to the service URL.

The following image shows the Order Approvers group added to the **Allowed Groups** field:

You can enter more than one group in the field.

You can configure the **Allowed Groups** and the **Allowed Users** fields. If a user falls into either the **Allowed Groups** or the **Allowed Users** category, the user will have access to the service URL.

If you configure the **Allowed Groups** or the **Allowed Users** field, and select the **Allow anonymous access** option, the entered group or user details disappear.

If you select the **Allow anonymous access** option, the **Allowed Groups** and the **Allowed Users** fields are not available for edit.

If you do not configure the **Allowed Groups** or the **Allowed Users** field, or do not select the **Allow anonymous access** option, no service URL is available for the process. However, you can call the process as a subprocess.

If you want the process to appear in API Manager, you must either select the **Allow anonymous access** option, or configure the **Allowed Groups** or the **Allowed Users** field. For more information about managing APIs, see the API Manager help.

If a process uses the **Allowed Groups** or the **Allowed Users** fields, you can use one of the following ways to invoke the process through a REST client such as Postman:

- Use basic authentication, and enter the user name and password.

- Use session ID in the HTTP header to invoke the process without providing the user name and password.

For example, to invoke a process service URL using Postman, authenticate the GET request in one of the following ways:

- Use basic authorization and specify the Informatica Intelligent Cloud Services user name and password.
  For example:

      GET <Informatica Intelligent Cloud Services URL>/active-bpel/rt/<API unique name>
      Accept: application/json
      Authorization: Basic Auth
      username: <Informatica Intelligent Cloud Services user name>
      password: <Informatica Intelligent Cloud Services password>

- Use the IDS-SESSION-ID in the HTTP header.
  For example:

      GET <Informatica Intelligent Cloud Services URL>/active-bpel/rt/<API unique name>
      Accept: application/json
      IDS-SESSION-ID: <sessionId>

  To get the SESSION-ID, use the Platform REST API version 3 login resource. For more information about the login resource, see *REST API Reference*.

Allowed Users

The users that have access to the process service URL at run time. If the users also have the Run privilege, they can invoke the process service URL.

Use the **Allowed Users** field when you want a specific user to have access to the service URL.

The following image shows the user **jsmith** in the **Allowed Users** field and the group **Order Approvers** in the **Allowed Groups** field:

In this process, users in the Order Approvers group and the user jsmith will have access to the service URL.

You can enter more than one user in the field.

If you configure the **Allowed Groups** or the **Allowed Users** field, and select the **Allow anonymous access** option, the entered group or user details disappear.

If you select the **Allow anonymous access** option, the **Allowed Groups** and the **Allowed Users** fields are not available for edit.

If you do not configure the **Allowed Groups** or the **Allowed Users** field, or do not select the **Allow anonymous access** option, no service URL is available for the process. However, you can call the process as a subprocess.

If you want the process to appear in API Manager, you must either select the **Allow anonymous access** option, or configure the **Allowed Groups** or the **Allowed Users** field. For more information about managing APIs, see the API Manager help.

If a process uses the **Allowed Groups** or the **Allowed Users** fields, you can use one of the following ways to invoke the process through a REST client such as Postman:

- Use basic authentication, and enter the user name and password.
- Use session ID in the HTTP header to invoke the process without providing the user name and password.

For example, to invoke a process service URL using Postman, authenticate the GET request in one of the following ways:

- Use basic authorization and specify the Informatica Intelligent Cloud Services user name and password.
  For example:

      GET <Informatica Intelligent Cloud Services URL>/active-bpel/rt/<API unique name>
      Accept: application/json
      Authorization: Basic Auth
      username: <Informatica Intelligent Cloud Services user name>
      password: <Informatica Intelligent Cloud Services password>

- Use the IDS-SESSION-ID in the HTTP header.
  For example:

      GET <Informatica Intelligent Cloud Services URL>/active-bpel/rt/<API unique name>
      Accept: application/json
      IDS-SESSION-ID: <sessionId>

  To get the SESSION-ID, use the Platform REST API version 3 login resource. For more information about the login resource, see *REST API Reference*.

Only accepts HTTP authorization requests from the API Gateway

Select this option to enable OAuth2 authentication for the process and configure the API to accept only HTTP authorization requests from the API Gateway.

To invoke the API from the API Gateway, the following users must complete configuration tasks:

API provider

The API provider must perform the following tasks:

1. Create a managed API for the process in API Manager and enable OAuth2 authentication for the managed API.
2. Create an OAuth2 client in API Manager, and generate the client ID and client secret.
3. Send the Informatica Intelligent Cloud Services OAuth 2.0 server URL and the client credentials to the API consumers.

API consumers

API consumers must perform the following tasks:

1. Authenticate against the Informatica Intelligent Cloud Services OAuth 2.0 server and use the OAuth 2.0 client credentials to generate an OAuth 2.0 authorization token.
2. Use the OAuth 2.0 authorization token to invoke the API.

For more information about creating an OAuth2 client and invoking an API with OAuth2 authentication, see the API Manager help.

Note: You can use the **Only accepts HTTP authorization requests from the API Gateway** option only for processes that run on the Cloud Server. Processes that use the **Only accepts HTTP authorization requests from the API Gateway** option can also be scheduled and run with one or more process inputs.

Allow anonymous access

When selected, Process Designer lets anyone use the process. This means that access to the process is not tied to a specific user or to any user group.

If you select the **Allow anonymous access** option, the **Allowed Groups** and the **Allowed Users** fields are not available for edit.

If you configure the **Allowed Groups** or the **Allowed Users** field, and select the **Allow anonymous access** option, the entered group or user details disappear.

If you do not configure the **Allowed Groups** or the **Allowed Users** field, or do not select the **Allow anonymous access** option, no service URL is available for the process. However, you can call the process as a subprocess.

If you want the process to appear in API Manager, you must either select the **Allow anonymous access** option, or use the **Allowed Groups** or **Allowed Users** fields. For more information about managing APIs, see the API Manager help.

Applies To

The type of object associated with the process. The objects that display in this list are the process objects you defined separately or generated in a defined connection. Drill down the list to see process objects and connections, and then select one of these items. Select a process object only if the process will be embedded in another process.

Note: To change the **Applies To** property of a published process, you must first unpublish the process. Then, change the **Applies To** property and republish the process

There are two special values:

- **Any**: Choose Any if you do not want to associate the process with a specific object. When you select this option, the process does not automatically have access to an object's fields.
  Use this object type when a process creates a new object and does not access information within existing objects.
- **Home (Salesforce)**: Select Home if you want a process to be visible from Home or any initial location where users launch all processes, regardless of the object type.

Note: If you directly invoke a process, the **Applies To** setting must be either **Any** or a specific object in a service, but not a process object.

Run On

Use this list to specify where the process must run. You can select **Cloud Server**, a specific agent, or a Secure Agent group.

If a process uses a connector that is event based, ensure that you run the process on the same agent that the connector runs on.

When the process is in the published state, the **Run On** list is disabled. To enable the **Run On** list, you must first unpublish the process. Then, change the **Run On** field value, and republish the process.

If you include a Human Task step in a process, you must run the process on Cloud Server.

Input Field Properties

The input fields available for a process are those contained in the object(s) to which the process applies. This is specified in the Applies To property (on the **Start** tab).

The Input field values are available in all steps of the process. You must define a name and type for the input field category for use in a process:

Property	Description

Input Format	Determines whether the input field can represent one or more fields, or the entire contents of the request. When you select **Whole Payload**, the input field represents the entire content of the request. For example, if you have a customer process object, its contents might be:
	
	`"customer": {"name" :"Joe Smith", "street":"3 Main St",`
	`    "city":"Shelton", "state":"CT"}}`
	
	If you check this option, the following would be posted:
	
	`{"name" :"Joe Smith", "street":"3 Main St",`
	`    "city":"Shelton", "state":"CT"}`

Name	The name of the input field.
Type	The type of the input field. For example, select **Date**, **Date Time**, **Time**, **Integer**, or **Text**. You can also add an input field of a simple type, custom type, or connection defined type. To do this, select **More types**, and then in the **Edit Type** dialog box, select the **Category** as **Simple Types**, **Custom Types**, or **Connection defined Types**.
Description	Description for the input field.
Required	If the field is required for a process to execute, check **Required**.

For more information on fields, see " Introduction to Data Types and Field Properties" on page 10.

XML Payload

In JSON syntax, a root container is not required. If the payload is XML, however, the root element is required. When it sends XML, Process Designer always adds a root node. If the payload is XML, Process Designer always adds a root node when it sends the request.

For example, if the XML is wrapped:

```
<root>
  <oAddressPO>
    <Street>3 Enterprise Drive</Street>
    <State>CT</State>
    <Zipcode>6484.0</Zipcode>
    <City>Shelton</City>
  </oAddressPO>
  <oAddressPO>
    <Street>31 Enterprise Drive</Street>
    <State>CT1</State>
    <Zipcode>106484.0</Zipcode>
    <City>Shelton1</City>
  </oAddressPO>
</root>
```

If the XML is unwrapped:

```
<root>
  < _1>
    <Street>3 Enterprise Drive</Street>
    <State>CT</State>
    <Zipcode>6484.0</Zipcode>
```

```
        <City>Shelton</City>
    </_1>
    <_2>
        <Street>31 Enterprise Drive</Street>
        <State>CT1</State>
        <Zipcode>106484.0</Zipcode>
        <City>Shelton1</City>
    </_2>
</root>
```

Output Field Properties

The output fields available for a process are those contained in the object(s) to which the process applies. This is specified in the Applies To property (on the **Start** tab).

The Output field values are set when the process executes and are then available in the steps that follow. Define output fields only for use in an embedded process. These output fields do not appear in lists until the values are returned from an embedded process. You define a name and type for the output field category for use in a process.

Property	Description
Output Format	Determines whether the output field can represent one or more fields of the response, or the entire contents of the response. When you select **Whole Payload**, the output field represents the entire content of the response.
Name	The name of the output field.
Type	The type of the output field. For example, select **Date**, **Date Time**, **Time**, **Integer**, or **Text**.
	You can also add an output field of a simple type, custom type, or connection defined type. To do this, select **More types**, and then in the **Edit Type** dialog box, select the **Category** as **Simple Types**, **Custom Types**, or **Connection defined Types**.
Description	Description for the output field.
Initial Value	The initial value for the output field.

For more information on fields, see " Introduction to Data Types and Field Properties" on page 10.

Temporary Field Properties

The temporary fields available for a process are those contained in the object(s) to which the process applies. This is specified in the Applies To property (on the **Start** tab).

The temporary field values will be used in all of the current process's steps. However, a temporary field's value is not available to an embedded process. You define a name and type for the temporary field category for use in a process:

Property	Description
Name	The name of the temp field.
Type	The type of the temp field. For example, select **Date**, **Date Time**, **Number**, **Percent**, **Time**, **Currency**, **Integer**, or **Text**. You can also add a temp field of a simple type, custom type, or connection defined type. To do this, select **More types**, and then in the **Edit Type** dialog box, select the **Category** as **Simple Types**, **Custom Types**, or **Connection defined Types**.
Description	Description for the temp field.
Initial Value	The initial value for the temp field. Default is 0 if the data type is Number, Percent, Currency, or Integer.

For more information on fields, see " Introduction to Data Types and Field Properties" on page 10.

Messages Properties

If needed, you can define one or more message events for this process.

For each message, you can define the following properties (similar to Start events):

Property	Description
Input Fields	Specify the field name. Select the type for each input field. For example, select **Date**, **Date Time**, **Time**, **Integer**, or **Text**. You can also add an input field of a simple type, custom type, or connection defined type. To do this, select **More types**, and then in the **Edit Type** dialog box, select the **Category** as **Simple Types**, **Custom Types**, or **Connection defined Types**. If the field is required for the message to be valid, select **Required**. If this field is used for a correlated message receive event, select **Use for Correlation**.
Output Fields	Specify the field name. Select the type for each output field. For example, select **Date**, **Date Time**, **Time**, **Integer**, or **Text**. You can also add an output field of a simple type, custom type, or connection defined type. To do this, select **More types**, and then in the **Edit Type** dialog box, select the **Category** as **Simple Types**, **Custom Types**, or **Connection defined Types**.
Binding	Select REST/SOAP or Event, depending on how this message is called.
Allowed Users/ Groups	Enter the users and groups who can access this message.
Allow Anonymous Access	When checked, anyone can use this message without any authentication. **Note:** If you enable this option, Process Designer ignores any limits you set with Allowed Users/Groups.

Property	Description
Input format is Whole payload	The input field can represent the entire contents of the request contained in this message.
Output format is Whole payload	The output field can represent the entire contents of the response. The way in which this changes a payload is the same as for an input field.

For more information on fields, see " Introduction to Data Types and Field Properties" on page 10.

Correlated Message Events

When you design a process, you can use message events in connection with the Receive step to interact with a process after the process has started execution. For example, you might want to:

- Query the status of a process
- Update data or provide control to the process

Based on one or more input parameters that uniquely identify the process, the process engine matches the correlated message events with their intended business process instances.

For example, a purchase ordering process might have an OrderId property. Messages can store the value of OrderId in different parts of the order process using different names and you associate a specific process instance with a unique OrderId. When a message arrives with the correlated OrderId, it is dispatched to the correct process instance. You can create as many correlation sets as you need.

Correlation matches incoming messages with their intended business process instances, so business processes can support asynchronous request/response patterns. (By comparison, synchronous calls need not rely on correlation because the message context is maintained on the stack or across a TCP connection.)

You define message events at the process level. A message event can be either interrupting or non-interrupting.

Before You Use Correlation

Take these steps before you define a set of correlated message receive events:

- Identify the Process Designer steps that need to be correlated. They should share one or more pieces of common data.
- Define a property that identifies the piece of common data.
- Define a correlation set of data so you can implement this in Process Designer.

Message Event Design Guidelines

When you define correlated message events, note that:

- For each message event you define in the process properties, at least one input field should be designated as **Use for Correlation**. If this Input field is a Reference type (a process object), also specify a path to a simple-typed field (with a choice of "field" or "formula").
- You can use event inputs and outputs as fields in the process but they are available only downstream of the Receive step (similar to outputs of Service steps).
- If you use multiple message event definitions and two input parameters in different message events have the same name, they should have the same type and same correlation path, if the input parameters are used for correlation. This ensures that you can have a single correlation value for a step.

You also specify whether it is interrupting or non-interrupting.

With an *interrupting* message event:

- The event terminates the step.
- The event path can merge with main path.
- The event path must have a milestone, which serves as the reply to the message event receive step.

With a *non-interrupting* message event:

- The process does not wait for the message event.
- The event path must end with an end step, which serves as the reply to the message event receive step.

Client Interaction to Invoke a Message Event

You can invoke a message event similar to a process, but using a different endpoint. For example, the endpoint to start a process might be:

```
…/active-bpel/public/rt/00000I/OrderProcessor
```

To invoke a specific message event, you would use an endpoint similar to:

```
…/active-bpel/public/rt/00000I/OrderProcessor/event/CheckOrderStatus
```

To learn more about concepts related to correlated message events, also see the *Process Developer Guide*.

Advanced Properties

You can configure advanced properties to define fault handling, tracing, and Cross-Origin Resource Sharing (CORS) support for a process.

You can configure the following advanced properties:

Suspend on Fault

By default, Process Server terminates a process when an uncaught fault occurs. You can configure Process Designer to suspend individual processes on uncaught faults.

If you select the **Suspend on Fault** option, Process Server suspends the process when an uncaught fault occurs. If you do not select the **Suspend on Fault** option, Process Server terminates the process if a fault occurs and the process does not explicitly handle the fault.

If the process is in the suspended state, review and identify error conditions. Then, you can then retry or complete the activity. For example, you can catch error conditions during employee or customer onboarding. After you resolve all the errors, the process moves out of the suspended state and continues from the faulted step.

If you select the **Suspend on Fault** option, Process Designer uses full persistence.

Note: These process-specific properties work in conjunction with other exception management settings that you specify in the Application Integration Console.

For more information about managing and correcting faults, see *Monitor*.

Tracing Level

You can configure one of the following tracing levels to determine the type of run-time logging for a process:

- None
- Terse
- Normal
- Verbose

The tracing level that you configure in Process Designer maps to the persistence level and logging level in Process Developer as listed in the following table:

Tracing Level	Persistence Level	Logging Level
None	Brief	None
Terse	Brief	Fault
Normal	Brief	Execution with Service Data
Verbose	Final	Execution with Data

Persistence Level

Regardless of the tracing level that you configure, Process Server uses full persistence in the following situations:

- When you select the **Suspend on Uncaught Fault** option
- When a process includes a timer such as a Wait step, a message event, or a receive event
- When one process invokes another process to provide subprocess coordination.

For more information about different persistence levels, see *Monitor*.

Logging Level

The Process Server logging level might be lower than the logging level that you set for an individual process. Regardless of the logging level that you set at the process level, the logging will not exceed the maximum process logging level specified for Process Server. The tracing level table illustrates the mappings between the logging level and the persistence level.

For more information about different logging levels, see the *Administrator* online help.

Automatic reset of tracing levels

In a process that is configured to run on the Cloud Server, if you set the tracing level to **Verbose** or **Normal**, the tracing level will be reset to **Fault** by the server, if necessary. This automatic reset feature helps in reducing the process execution time caused by the run-time logging.

Note: The automatic reset of tracing levels feature is in technical preview. Technical preview functionality is supported but is unwarranted and is not production-ready. Informatica recommends that you use in non-production environments only. Informatica intends to include the preview functionality in an upcoming GA release for production use, but might choose not to in accordance with changing market or technical circumstances. For more information, contact Informatica Global Customer Support.

Enable CORS

By default, API consumers can invoke a process from the domain that is part of the service URL. API consumers can use the GET, POST, PATCH, PUT, and DELETE verbs to invoke a process.

Select the **Enable CORS** option to enable Cross-Origin Resource Sharing (CORS) support for a process. You can also specify the allowed domains and allowed verbs for CORS support.

In the **For Domains** field, specify additional allowed domains from which API consumers can invoke the process. Enter the wildcard character (*) to enable CORS for all domains. Enter a comma-separated list of domains to enable CORS for specific domains. For example, enter the following phrase to enable CORS for the abc.domain.com and xyz.domain.com domains:

```
abc.domain.com,xyz.domain.com
```

If you enable CORS but leave the **For Domains** field blank, you can invoke the process only from the domain that is part of the service URL.

In the **For Verbs** field, specify the allowed verbs that API consumers can use to invoke a process from the domains that you specified in the **For Domains** field. Enter a comma-separated list of verbs to enable CORS for specific verbs. For example, enter the following phrase to enable CORS for the POST, PUT, and GET verbs:

```
POST,PUT,GET
```

Default is all verbs.

API consumers can invoke the process with all verbs from the domain that is part of the service URL.

Notes

Use the **Notes** tab to add information that you or other developers might need. The Notes you enter display only at design time. The guide or process users or consumers do not see the notes. For example, you might add a reminder about something that needs to be done before you deploy a process or guide.

Updating field values for processes

You can use the update resource to update the values for the following fields for existing processes:

- Allowed Users/Allowed Groups/Allow anonymous access
- HTTP authorization requests from the API Gateway
- Run On
- Tracing Level

You can update fields for a single process or multiple processes. To update the processes using the update resource, you must be assigned the Admin, Deployer, Designer, or Operator role.

You can add the process and fields that you want to update as input data in a REST client such as Postman. The input data consists of the Global Identifier/Global Unique Identifier (GUID) of the process and a key-value pair of deployments to be updated. For more information about GUID, see the lookup resource and finding assets resource in *REST API Reference* in the Data Integration help.

To update the field values, use the following URL:

```
<Informatica Intelligent Cloud Services URL>/active-bpel/asset/v1/update?assetType=Process
```

You can add the input data as shown in the following sample:

```
PUT <baseApiUrl>/active-bpel/asset/v1/update?assetType=Process
Content-Type: application/json
Accept: application/json
INFA-SESSION-ID: <Infa-Session-ID>
{
  "processes": [
   {
      "guid": {
        "$t": "<guidvalue>"
      },
      "deployment": [
        {
          "key": {
            "$t": "Allow Anonymous Access"
          },
          "value": {
            "$t": "true"
          }
        },
        {
          "key": {
```

```
            "$t": "Run On"
          },
          "value": {
            "$t": "cloud server"
          }
        },
        {
          "key": {
            "$t": "Tracing Level"
          },
          "value": {
            "$t": "normal"
          }
        }
      ]
    }
  ]
}
```

You can use the login resource to get the `INFA-SESSION-ID`. For more information about the login resource, see *REST API Reference* in the Data Integration help.

If the fields are successfully updated, you receive the following response:

```
{
    "status": {
      "$t": "SUCCESS"
    }
}
```

If you enter an invalid Secure Agent, you receive the following response:

```
{
  "status": {
    "$t": "FAILED"
  },
  "errorDetails": {
    "error": {
      "$t": "Update failed for assets"
    },
    "failedAssets": {
      "updateAssetFaultInfo": {
        "xmlns": "https://com.informatica.icrt/2021/08/updateAssets.xsd",
        "xmlns$aetgt": "https://com.informatica.icrt/2021/08/updateAssets.xsd",
        "name": {},
        "GUID": {
          "$t": "<Incorrect guid value>"
        },
        "fault": {
          "$t": "The Secure Agent is not valid. Specify a valid Secure Agent."
        }
      }
    }
  }
}
```

Rules and guidelines for updating fields in a process

Consider the following rules and guidelines when you use the update resource to update the fields in a process:

*   You must set either the Allowed Users/Allowed Groups or Allow anonymous access field at a time. Otherwise, an error occurs.
*   You must set either the HTTP authorization request from the API Gateway or the Allow anonymous access field at a time. Otherwise, an error occurs.
*   In the HTTP authorization request from the API Gateway and Allow anonymous access field, you must specify the value as true or false. If you specify any other value such as yes or no, an error occurs.

- In the Tracing Level field, you must specify one of the following values: None, Terse, Normal, or Verbose. Otherwise, an error occurs.
- If you set the HTTP authorization request from the API Gateway value to true, you must specify the value for the Run On (runtime environment) field as Cloud Server. If you specify the run on value as Secure Agent, an error occurs.
- If you want to update the runtime environment value for a published process, you must first unpublish the process.

Adding Process Steps

Add steps to access data, services, and perform related orchestration activities.

When you add a step, you also define properties for that step.

Perform one of the following tasks to add a step to a process:

- Drag a step from the pallette on the left onto the canvas.
- Double click on the canvas and select a step from the **Step Type** list.

Assignment Step

Use the Assignment step to set a value to a field. To create an Assignment step, click a process, and then select **Assignment** from the Design canvas. Then, from the Assignments properties panel, click the **Add** icon to assign Name and Assignment values.

The following image shows an Assignment step:

To add a field, click the **Add** icon, and then add the following information for each field:

Name

This is the fully-qualified field name. The name can contain only alphanumeric characters, underscores (_), spaces, and Unicode characters.

Assignments

This is the source from which the field takes values. The fields you see depend on the data type you defined under **Start** > **Properties** > **Input Fields or Temporary Fields**.

For example, if you define the data type in the Input Fields as Date or Time, the following options appear for that field in the Assignment step:

- Specific date
- Days from today
- Days before/after
- Field
- Formula

If you define the data type as Integer or Text, you see the following options to specify the value of the field:

- Content
- Field
- Formula

For more information about data types, see "Using the Field Properties Dialog " on page 20.

Service Step

When you add a Service step, you set some properties.

The following sections describes the Service step properties:

General

Property	Description
Step Type	The Service step.
Name	A descriptive name for the service. The name can contain only alphanumeric characters, underscores (_), spaces, and Unicode characters.

Service

Property	Description
Service Type	The connection, process, or system service you add to the process. Select from a list of existing tasks. **Note:** You must have an existing item to add to a process. You cannot create a task in when you create a process.

For more information about the list of supported actions for system service, see "System Service Actions" on page 140.

When you add a service to a Service step, corresponding Input Fields are created.

Input Fields

Input Fields section shows the names of input fields that are most often used when using this kind of step. For Service steps that create objects, the input fields shown are the fields that are most frequently needed when the object is created. (If you do not need a field and it is optional, you can delete it.) You can choose additional (optional) input fields from the list.

Use the delete icon to remove a field. This removes input fields that you do not want to pass to the Service step. Some fields are required and cannot be deleted.

Fault Handling

Fault handling in Process Designer is based on the Business Process Model and Notation (BPMN) 2.0 specification, which defines the concept of a boundary event. The boundary events catch faults associated with specific steps, rather than the overall process scope. This means you can handle faults at the step level, not the process level. (Developers can also use Process Developer to define a fault handler on the process scope.)

Visibility of Faults in Application Integration Console

When fault handling is enabled in a process, you can view the error marker on the step and some basic fault information in the Application Integration Console Processes list.

Timer Events

Use Timer Events to perform an action based on a schedule. You can specify whether you want the event to run **At specific time** or **After a delay**.

The following image shows a configured timer event:
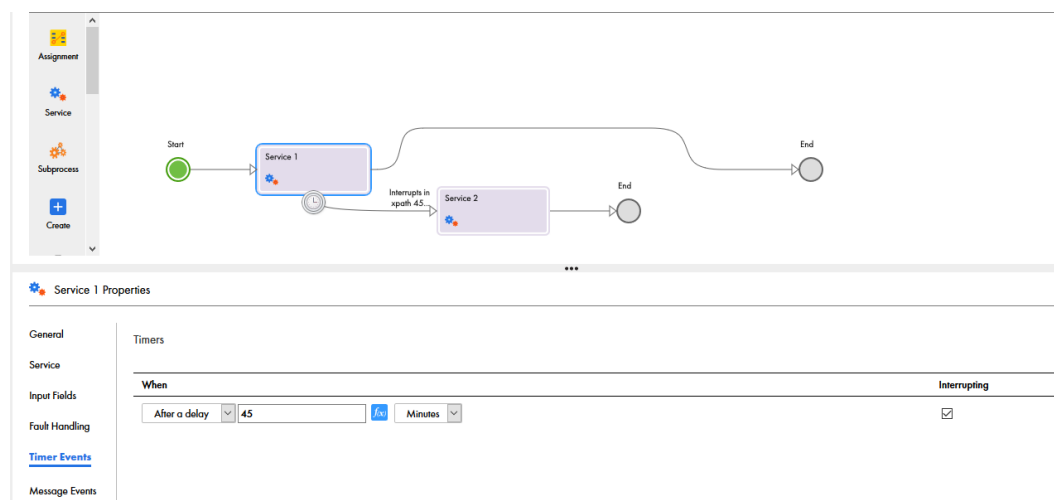
Select **Interrupting** if you want the timer to interrupt the Service step. When you set an interrupting timer, the Service Step task is interrupted and the process only runs the task on the timer set

Message Events

Specify messages for use with message providers and correlation. You can add multiple instances of the same message event to the Service step.

Subprocess Step

A Subprocess step embeds one process within another process. When the Subprocess step executes, the embedded process executes.

When you have a process that contains numerous steps, consider splitting the orchestration logic across multiple smaller processes. You can then simplify the design by using the Subprocess step to embed the smaller processes in the parent process. This not only leads to modular design, but also helps with faster loading when you open the process for editing.

The Subprocess step name can contain only alphanumeric characters, underscores (_), spaces, and Unicode characters.

You can define the Input and Output associated fields for the Subprocess step, if any, based on the embedded process.

The following image shows the JMS Dequeue subprocess properties:

Iteration Rule

The iteration rule applies when the selected process in the Subprocess step has the **Applies To** field set to an object type.

You can configure the subprocess to run on a single object or on all objects in a specified list.

If you select **Run on a single object**, the subprocess runs on the specific object it receives as input. When execution completes, the subprocess passes control to the step that follows in the subprocess.

If you select **Run on each object in list until**, you also choose an event that controls the subprocess execution.

You can select one of the following values to define the event that controls the subprocess execution at run time:

- **Field**: Available for both the options, **Run on a single object** and **Run on each object in list until**, that is, when a single object or a list of objects is passed to the subprocess based on the selected Run choice. It contains the process objects or the salesforce connection fields available in the Connection Properties and Input Parameters.

- **Query**: Available only when you select the **Run on a single object** option. You can define a WHERE condition that selects the object passed to the subprocess.

- **List**: Available only when you select the **Run on each object in list until** option. You can define a WHERE condition that selects the objects passed to the subprocess.

The following image shows the iteration rule properties:
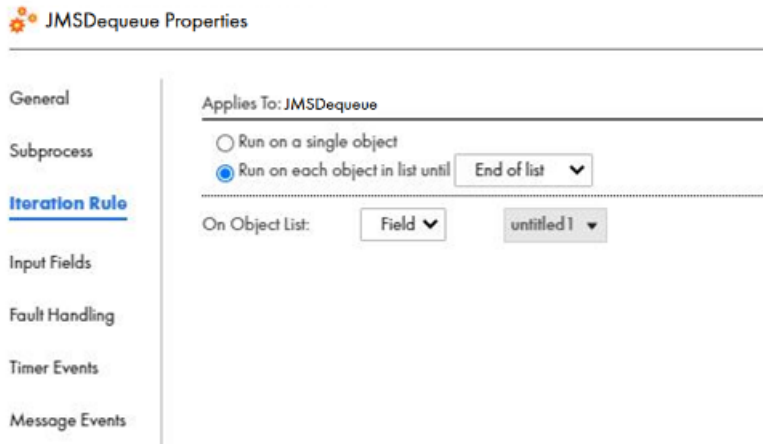
Human Task Step

Use the Human Task step to include a human decision within an Application Integration process. A human action is required for approvals and exception management. For example, you might want to add a Human Task step to a loan approval process.

To include a human task into a process, drag and drop the Human Task step from the palette to the canvas. Double-click the Human Task step and select the human task that was created in the human task asset.

The following image shows a Human Task step within an Application Integration process:
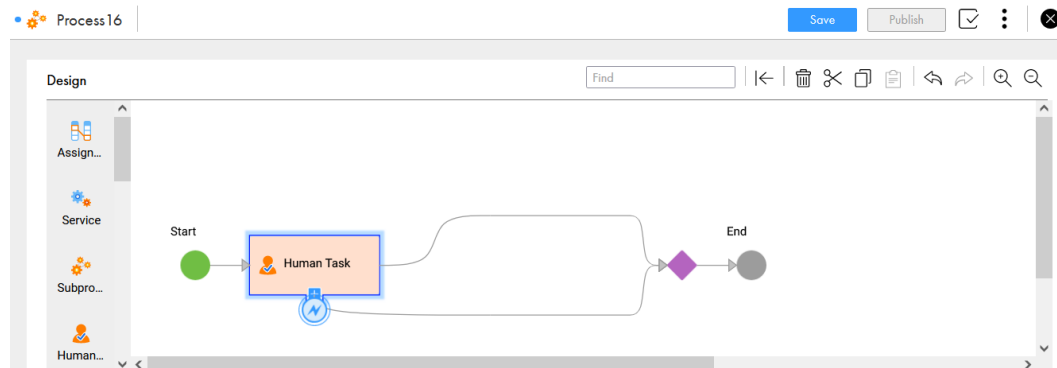
The following table describes the properties that you can configure in a Human Task step:

Property	Description
General	The name of the Human Task step.
Human Task	Select the human task that you want to include within the Application Integration process. The tab displays all the settings that were defined in the human task asset such as the task description, task priority, input fields, output fields, and assignments. The tab also displays whether the task is skippable or not.

Property	Description
Assignments	Select from the asset, field, or formula options corresponding to the **Task Role** field and assign a value to the field. By default, the **Asset** option is selected and the users and roles that were configured during task creation are retained. You can use the **Field** or **Formula** option to override the users and roles that were defined in the human task asset. For more information, see "Overriding assignments in human tasks" on page 128.
Input Fields	Enter the values in the input fields for the human task.
Outcomes	Displays a list containing the outcomes that you defined in the human task asset.
Fault Handling	Select the **Catch Faults** option to enable fault handling for the Human Task step. You can provide a name for the fault handler in the **Fault Field Name** field.

For information about creating a Human Task asset, see "Human Task creation " on page 333.

Note: The Human Task step is available for preview. Preview functionality is supported for evaluation purposes but is unwarranted and is not supported in production environments or any environment that you plan to push to production. Informatica intends to include the preview functionality in an upcoming release for production use, but might choose not to in accordance with changing market or technical circumstances. Note that if you are working on a preview POD, all data is excluded from SOC 2 compliance coverage. For more information, contact Informatica Global Customer Support.

Overriding assignments in human tasks

When you use a Human Task step in a process, you can override the users and roles that were defined in the human task asset using a formula or field. You can also use the Expression Editor to create simple expressions that only contain a field or complex expressions that include nested functions.

To override the assignments, perform the following steps:

1. Click the **Assignments** tab.

    The tab displays the users and roles that were defined in the human task asset.

2. Click the **Value From** field to select the asset, field, or formula. Perform one of the following steps:

    *   **Asset**: Select the **Asset** option to retain the users and roles that were defined in the human task asset.

    *   **Field**: Select the **Field** option to select a field and override the value.

    *   **Formula**: Select the **Formula** option to use a formula to override the value. Use the Expression Editor to create a formula with expressions.

Create Step

Use the Create step to add new object instances. For example, if you work with an Account object, you can use the Create step to add a new account.

The Create step name can contain only alphanumeric characters, underscores (_), spaces, and Unicode characters.

The following image shows a Create step for the Account object:

To create new objects:

1. Choose the **Input Fields** tab and select all the fields that you will be adding to the object. Some are required; others are optional.

2. For each field name, set its source to one of the following values: **Content**, **Field**, or **Formula**.

Using Reference Fields

Note that:

- If you do not include a reference field on the Input Fields tab for the object you are creating, Process Designer uses the value of the current object at runtime.

- Reference fields for the *Applies To* object are set if they are not set explicitly in the Input tab.

- When the content of a reference field is set to empty, it is not included in the create statement. (This prevents problems that might occur when explicitly setting a field to empty or null could violate data constraints.)

Receive Step

A Receive step can be defined in processes that make a Service and wait for some event data before the process can continue. For example, if the order process includes a step to request more credit information when the order is above a certain threshold, then a Receive step allows you to handle the blocking receive call.

The Receive step name can contain only alphanumeric characters, underscores (_), spaces, and Unicode characters.

The output of a Receive step is available to be used later in the process, similar to output fields in Service steps.

For the **Correlation Assignments**, select the fields you are correlating with the incoming message event. You can use these assignments to cancel further execution of a request (interrupting) or to provide status (non-interrupting).

For more information, refer to: "Correlated Message Events" on page 118

Wait Step

When you add a Wait step, you set some properties.

The following table describes the Wait step properties:

Property	Description
Name	The name of the Wait step. The name can contain only alphanumeric characters, underscores (_), spaces, and Unicode characters.
Wait	Properties that determine when and for how long the process pauses. You can choose from the following options: - **At a specific time** - **After a wait period**

Pause at a specific time

Select this option to pause the process at a particular time. Enter the **Time** you want the process to pause at, and optionally, a **Delay**. The **Delay** value can be an integer or a field that you define.

The following image shows the **At a Specific Time** option:
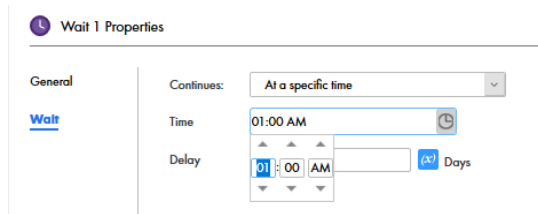
For example, set the process to pause at 1:00 am after three days. 1:00 am is the **Time** and three days is the **Delay**.

After a wait period

Select this option to pause the process after a period. The period begins when the process reaches the Wait step. Enter the **Wait Period** that you want the process to pause for. The **Wait Period** value can be an integer or a field that you define.

For example, set the process to pause for one hour from the time that the process reaches the Wait step.

The following image shows the **After a wait period** option:
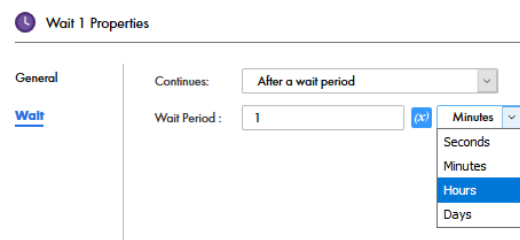
Milestone Step

Use the Milestone step to specify one of two actions when the process ends.

The following image shows a Milestone step:
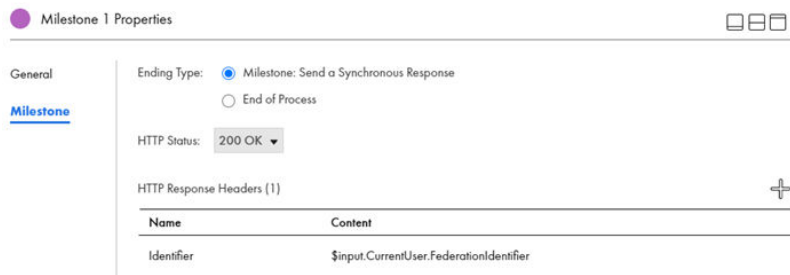
You can configure the following Milestone step properties:

Name

Displays the name of the milestone step. You can edit this value.

Ending Type

You can select one of the following values:

Milestone: Send Synchronous Response

Sends a synchronous reply to the original request. The reply has all the output fields defined in the **Process Properties** dialog. If a synchronous reply was already sent, no action occurs. This is not an error as it allows background and batch work to occur. You can have additional steps after a Milestone step to send a synchronous message, and then continue executing.

Note: If you insert a Milestone step into a Parallel Paths step, the **Ending Type** field must be set to **Milestone: Send a Synchronous Response**.

End of Process

No further action occurs when the process ends.

The default value is `Milestone: Send Synchronous Response`.

HTTP Status

Displays the HTTP response status code. The default value is `200 OK`. You can edit this value.

HTTP Response Headers

Specifies the HTTP response headers and values that the REST API associated with the process must return after the process runs.

For example, you can configure a response header to return the organization ID. You can consume the header value that the process returns in downstream applications.

You can configure a response header to return a lookup value. Based on the lookup value that the process returns, you can make a decision and perform further processing in downstream applications.

You can configure the `X-XSS-Protection` HTTP response header to enable additional security for a REST API in various browsers.

Enter `X-XSS-Protection` as the header name and `1; mode=block` as the value.

The `X-XSS-Protection` header enables cross site scripting (XSS) filtering. If the browser detects an XSS attack, it will not render the page.

You can add multiple response headers to the process by configuring the name and value for each header. The header name must be unique and cannot contain the following characters:

`( ) < > @ , ; \\ / [ ] ? = { } _`

The header name must also not contain non-English characters. If you want to provide a non-English character in the header value, you must encode the text in the Base64 format. The downstream

application must decode the text to get the original content. You can add the values manually or enter a formula.

A process uses the following default HTTP response headers:

- access-control-allow-credentials
- cache-control
- content-encoding
- content-type
- date
- expires
- keep-alive
- status
- strict-transport-security
- transfer-encoding
- vary
- x-frame-options
- x-http-status
- x-instanceid

You cannot override the names and values of the default headers.

After the process runs, the API returns the HTTP response headers that you configured. You can view the response headers on the **My Processes** page in Application Integration or on the **Processes** page in Application Integration Console.

End Step

An End step indicates the end of the process. When execution reaches this step, the process completes.

You can configure the following End step properties:
Name

Displays the name of the end step. You can edit this value.

Ending Type

You can select one of the following values:
Milestone: Send Synchronous Response

Sends a synchronous reply to the original request. The reply has all the output fields defined in the **Process Properties** dialog. If a synchronous reply was already sent, no action occurs. This is not an error as it allows background and batch work to occur. You can have additional steps after a Milestone step to send a synchronous message, and then continue executing.

If you set the value as `Milestone: Send a Synchronous Response`, you see an option to select the next step.

End of Process

No further action occurs when the process ends.

The default value is `End of Process`.

HTTP Status

Displays the HTTP response status code. The default value is `200 OK`. You can edit this value.

HTTP Response Headers

Specifies the HTTP response headers and values that the REST API associated with the process must return after the process runs.

For example, you can configure a response header to return the organization ID. You can consume the header value that the process returns in downstream applications.

You can configure a response header to return a lookup value. Based on the lookup value that the process returns, you can make a decision and perform further processing in downstream applications.

You can configure the `X-XSS-Protection` HTTP response header to enable additional security for a REST API in various browsers.

Enter `X-XSS-Protection` as the header name and `1; mode=block` as the value.

The `X-XSS-Protection` header enables cross site scripting (XSS) filtering. If the browser detects an XSS attack, it will not render the page.

You can add multiple response headers to the process by configuring the name and value for each header. The header name must be unique and cannot contain the following characters:

`( ) < > @ , ; \\ / [ ] ? = { } _`

The header name must also not contain non-English characters. If you want to provide a non-English character in the header value, you must encode the text in the Base64 format. The downstream application must decode the text to get the original content. You can add the values manually or enter a formula.

A process uses the following default HTTP response headers:

- access-control-allow-credentials
- cache-control
- content-encoding
- content-type
- date
- expires
- keep-alive
- status
- strict-transport-security
- transfer-encoding
- vary
- x-frame-options
- x-http-status
- x-instanceid

You cannot override the names and values of the default headers.

After the process runs, the API returns the HTTP response headers that you configured. You can view the response headers on the **My Processes** page in Application Integration or on the **Processes** page in Application Integration Console.

Throw Step

You can use a Throw step to construct an error payload.

The following table shows the properties in a Throw step:

Property	Description
Name	Required. A descriptive name for the Throw step. The name can contain only alphanumeric characters, underscores (_), spaces, and Unicode characters.
Code	Required. A string value that contains the name of the faultInfo payload.
Detail	Optional. A string value that contains the faultInfo detail.
Reason	Optional. The faultInfo reason detail, which can be type $any.

Loop Step

A Loop step runs the steps configured within the loop until a certain condition is satisfied.

The following table describes the properties in a Loop step:

Property	Description
Name	The name of the Loop step. The name can contain only alphanumeric characters, hyphens (-), underscores (_), spaces, and Unicode characters.
Loop	Properties that determine the steps to run within the loop until a certain condition is satisfied. You can select one of the following options: - **While** - **Repeat Until**

Type

You can select one of the following options:

While

In the While loop, the process enters the loop only if the loop condition is satisfied. The While loop executes an activity repeatedly until its condition evaluates to false.

In contrast to the Repeat Until activity, the While loop might not execute the contained activity at all.

Repeat Until

In the Repeat Until loop, the process enters the loop but exits only when the loop condition is satisfied. The Repeat Until loop executes an activity repeatedly until its condition evaluates to true.

In contrast to the While activity, the Repeat Until loop executes the contained activity at least once.

Looping Criteria

You can select one of the following options:

Formula

You can create a condition using an XQuery function. You can also evaluate simple and complex expressions directly in the Loop step with no dependency on the prior steps.

Field

Select an input field, output field, or temporary field from the list of fields.

Enter the condition and value that you want the Loop step to base the step execution on.

The conditions available depend on the field you select.

For example, if you select a field of type **Simple** > **Number**, the following conditions are available:

- Equals
- Not equal to
- Less than
- Less than or equal to
- Greater than
- Greater than or equal to
- Is set
- Is not set

You can enter a value against the condition you select.

Note: You must set an initial value for the fields used in the loop condition. Otherwise, the loop becomes infinite.

Loop step example

You create a Loop step with the following properties:

Type: While

Looping Criteria: Field with value temp

Condition: Less than with value set to 5

You have a temp field with the initial value set to 1, and the temp field is configured to increment by 1 using the Assignment step within the Loop step.

The following image shows the **While** option with the looping criteria set to field in the Loop step:
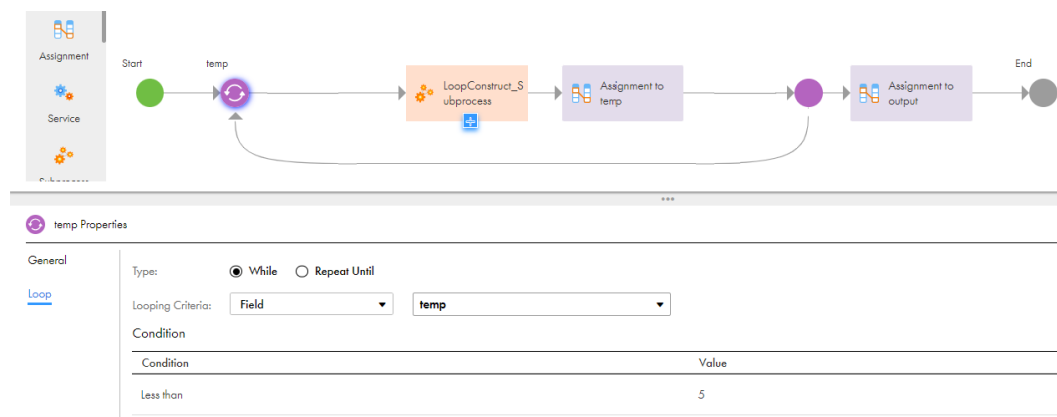
In this case, when you run the process, the subprocess runs four times until the temp value reaches 4, and then the process comes out of the loop and runs the subsequent steps.

For the same process, if you create a loop with the following properties:

Type: Repeat Until

Looping Criteria: Formula with `$temp.temp >5`

The following image shows the **Repeat Until** option with the looping criteria set to formula in the Loop step:
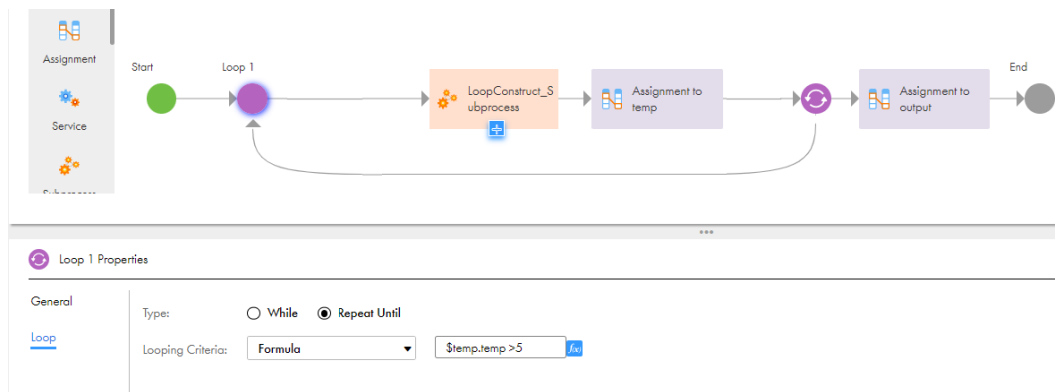
In this case, when you run the process, the subprocess runs five times until the temp value reaches 5, and then the process comes out of the loop and runs the subsequent steps.

Rules and guidelines for using Loop step

Consider the following rules and guidelines when you use the Loop step in a process:

- You cannot use the Jump step within the Loop step. You cannot use the Jump step from inside the loop to outside or from outside the loop to inside.
- If you use the Receive step inside the Loop step, you must use a Milestone step after the Receive step within the Loop step to send a synchronous reply to the request.
- You cannot use the End step and Throw step within the Loop step.
- It is recommended that you use the modern view to edit a process that contains a Loop step. When you use the Loop step in a process and switch between the modern view and classic view, you might encounter issues. For more information, see the following Knowledge Base article: https://knowledge.informatica.com/s/article/ERROR-Process-Designer-issues-when-you-use-a-Loop-step-in-a-process-and-switch-between-the-modern-view-and-the-classic-view?language=en_US

Note: The loop construct feature is available for preview. Preview functionality is supported for evaluation purposes but is unwarranted and is not supported in production environments or any environment that you plan to push to production. Informatica intends to include the preview functionality in an upcoming release for production use, but might choose not to in accordance with changing market or technical circumstances. Note that if you are working on a preview POD, all data is excluded from SOC 2 compliance coverage. For more information, contact Informatica Global Customer Support.

Decision Step

A Decision step allows a process to take different paths depending on the value of the field.

The following table describes the properties in a Decision step:

Property	Description
Name	The name of the Decision step. The name can contain only alphanumeric characters, underscores (_), spaces, and Unicode characters.
Decision	The process takes a decision based on the fields and paths you define here. Select a field name from the list of fields you define under **Start** > **Fields**. Enter conditions and values that you want the Decision step to base a decision on. The conditions available depend on the field that you select. For example, if you select a field of type **Simple** > **Text**, the following conditions are available: - Equals - Starts With - Ends With - Starts with any of - Contains You can enter text values against the conditions you select. You can add multiple conditions to a Decision step. Each condition is a potential data path. For each path that you add, a corresponding branch appears on the UI. Drag branches to rearrange the order in which the branches appear on the UI. Most Decision steps have an Otherwise path. This path handles execution if no data meets the conditions in your tests.

Evaluating Paths

A process evaluates conditions based on the criteria you specify. Ensure that you construct paths with non-intersecting conditions.

For example, you create a Data Decision step with the following paths:

- Path 1: Field less than or equal to 100.

- Path 2: Field less than or equal to 75.

- Path 3: Field less than or equal to 25.

- Path 4: Otherwise

If the integer field for which the Data Decision step was created has a value of 25, the Data Decision step takes path 1. This is because 25 is less than 100 and path 1 is the first option.

To ensure that the Data Decision step follows the "Field less than or equal to 25" path, re-create the paths with the following criteria:

- Path 1: Integer between 0 and 25

- Path 2: Integer between 26 and 75.

- Path 3: Integer between 76 and 100.

- Path 4: Otherwise

Important: The process evaluates conditions in a top-down manner. Ensure that the Otherwise branch is the last path.

A Decision step can lead to another Decision step. For example, a branch could run if an annual income exceeds $100,000. The next decision test along the same path could test if the city is Boston, or otherwise.

Using this technique, you use Boolean AND logic because you base the test for the second condition on the true branch of the first condition. In this example, you use the Decision step to set the condition "Annual Revenue exceeds $100,000 AND city is Boston".

Similarly, to support Boolean OR logic, you can add a test for the second condition on any branch.

Parallel Paths Step

When you add a Parallel Paths step, you set some properties.

The following table describes the properties in a Parallel Paths step:

Property	Description
Name	The name of the Parallel Paths step. The name can contain only alphanumeric characters, underscores (_), spaces, and Unicode characters.
Parallel Paths	The paths that you want the process to run in parallel. **Note:** When you use a Parallel Paths step, all the steps in all the branches will not start at the same time. There will be a time lag in the start time of the steps in all the paths. The execution of branches varies based on the steps used in each branch. When more than one branch is running in sequence, if one branch pauses for a specific time, instead of waiting for the steps to complete in that branch, the process starts running steps from another branch in parallel. For example, if you have a Wait step in a branch and the process is paused for a specific time, the steps in a different branch starts running in parallel. Click **Add** to add a new branch. You can add multiple steps to each branch. To add steps to a branch, drag and drop a step from the pallette on the left. When you use the Jump step in conjunction with the Parallel Path step, you can only jump to another step on the same Parallel Path branch. Keep in mind the following restrictions when you use the Jump step and the Parallel Path step together: - If you are in a Parallel Path step, you cannot jump to a step on another branch of the same Parallel Path step. - If you are in a Parallel Path step, you cannot jump to any step outside the Parallel Path step. - If you are outside a Parallel Path step, you cannot jump to any step inside the Parallel Path step.

Jump Step

When you add a Jump step, you set some properties.

The following table describes the properties in a Jump step:

Property	Description
To	The target of the jump. Select from a list of available steps.
	More than one step can jump to the same target step. To see how many Jump steps have a particular step as their target, place the cursor over the arrow next to the target step.
	When you use the Jump step in conjunction with the Parallel Path step, you can only jump to another step on the same Parallel Path branch.
	Keep in mind the following restrictions when you use the Jump step and the Parallel Path step together: - If you are in a Parallel Path step, you can't jump to a step on another branch of the same Parallel Path step. - If you are in a Parallel Path step, you can't jump to any step outside the Parallel Path step. - If you are outside a Parallel Path step, you can't jump to any step inside the Parallel Path step. - You can't jump to any step from the fault path. You must merge the fault path back to the main process path to add the Jump step. - You can't jump to a step inside the fault path if you are outside the fault path.
	Use the following best practices when you use the Jump step in a process: - Instead of using a Jump step from an outside flow to a step inside the fault path, design the process by replacing the Jump step with the same flow as in the fault path. - If you use a Jump step for iteration purposes and it points to a Service or a Subprocess step with fault handling, enclose the Service or Subprocess step with fault handling inside another Subprocess step.

System Service Actions

When you configure a Service step in a process and select system service as the service type, you can choose from a list of supported actions.

Delete Object

This Delete Object action lets you delete a record. Depending upon your application, what is being deleted can be called an object or an object instance.

As the figure shows, you will need to enter information for three fields.

Execute Data Marketplace API

You can use Data Marketplace APIs to access data assets and data collections through GET, POST, and PUT methods, and perform business orchestration. You can execute Data Marketplace APIs for processes that run on the Cloud Server.

You can configure a Data Marketplace API in the Service step and invoke it using API Gateway.

To use Data Marketplace APIs in a process, you must configure the **Allowed Groups** or the **Allowed Users** fields. Users must have the Data Marketplace administrator role to invoke a process that integrates with a Data Marketplace API.

The following image shows the Execute Data Marketplace API properties:

Input Fields

Configure the following input fields in the Service step for a Data Marketplace API:

HTTP Method

Required. The HTTP method to invoke the Data Marketplace API such as GET, PUT, or POST.

Relative Path

Required. The relative path of the Data Marketplace API.

For example, to perform operations on the data assets in Data Marketplace, enter the following as the relative path:

```
dataAssets
```

To perform operations on the data elements in Data Marketplace, enter the following as the relative path:

```
dataAssets/{{id}}/dataElements
```

Payload

The JSON payload to be sent to the Data Marketplace API using the PUT or POST method.

For example, to send employee data to the Data Marketplace API, enter the data in the JSON format as shown in the following sample:

```
'{
  "Employee": [
```

```
          {
            "firstName ": "Cindy",
            "lastName": "Louis",
            "department": "IT"
            "status": "Employed"
          }
      ]
    }'
```

Query Parameter

The process object that returns the data associated with the Data Marketplace API query parameters.

You can use the API query parameters such as sort or status to sort or filter the data using the GET method. Based on the query parameter, the API returns the associated data in the XML format.

You can configure the Query Parameter field by using a formula or field.

For example, to use a formula to get the maximum number of assets with the enabled status, use the status query parameter as shown in the following sample:

```
<DataMarketplaceQueryParameters>
    <queryParams>
      <name>status</name>
      <value>ENABLED</value>
    </queryParams>
    <queryParams>
      <name>limit</name>
      <value>10</value>
    </queryParams>
</DataMarketplaceQueryParameters>
```

For information about using another field to configure the Query Parameter field, see "Creating fields to execute Data Marketplace API " on page 143.

Path Variable

The process object that is associated with the Data Marketplace API path variables.

You can configure the Path Variable field by using a formula or field.

For example, to insert data elements into the data asset, provide the ID in the path variable as shown in the following sample:

```
<DataMarketplacePathVariables>
    <pathVariables>
      <name>id</name>
      <value>eu33-39839j2</value>
    </pathVariables>
    <pathVariables>
      <name>categoryId</name>
      <value>ad2c915-60c4</value>
    </pathVariables>
</DataMarketplacePathVariables>
```

For information about using another field to configure the Path Variable field, see "Creating fields to execute Data Marketplace API " on page 143.

Output Field

Configure the following output field in the Assignment step or Decision step for a Data Marketplace API:

Name	Type	Description
Data Marketplace Response	Text	The string variable that stores the output.

For more information about Data Marketplace APIs, see the Data Marketplace help.

Creating fields to execute Data Marketplace API

You can configure the query parameter and path variable by passing the value as a field.

To pass the value of the query parameter and path variable as a field, you must create temporary fields. You must select the custom type as **Data Marketplace Query Parameters** process object for the query parameter field and **Data Marketplace Path Variables** process object for the path variable field as shown in the following image:

The temporary fields you created appear in the Assignment step. In the Assignment step, you can configure the values for the query parameter and path variable temporary fields as shown in the following image:

Here, `queryParams` and `pathVariables` are the input fields configured in the Data Marketplace Query Parameters and Data Marketplace Path Variables process objects with the name-value pair.

In the Service step, you must select the temporary fields that were added for the query parameter or path variable field values as shown in the following image:

JMS Enqueue Service

Use the JMS Enqueue Service to send a message to a JMS service, where it will be queued. When the receiving process is ready, the service sends the message to it. The message is sent using the Informatica Secure Agent to communicate with an on-premise service.

The data in the message contains the following components:

Messaging Manager Name

The name of the messaging manager that contains the destination.

Destination Name

The name of the message's destination.

Format

The format to be serialized to the destination.

Message

The message to be sent to the destination.

Unwrap Root Element

Describes how the process object is serialized when it is sent to the destination.

The following image shows the JMS Enqueue Service properties:

Logging for the JMS Enqueue Service

By default, the JMS Enqueue Service logs only errors. To enable additional logging for the entire flow of message processing, perform the following steps:

1. Access the `log4j2.xml` file from the following directory:

   ```
   <Secure Agent installation directory>\apps\process-engine\<latest_version>\conf
   ```

2. Add the following code:

   ```
   <Logger name="org.apache.camel.component.file.remote" level="DEBUG"
   additivity="false">
        <AppenderRef ref="catalina"/>
   </Logger>
   ```

3. Save the `log4j2.xml` file.

MDM human tasks

Informatica Intelligent Cloud Services Business 360 Console uses the Application Integration processes as workflows for the user-triggered and system-generated business events. If you do not want to use any

predefined approval workflows in Business 360 Console, define a process in Application Integration and use it to configure business events. These processes must contain at least one human task.

A human task indicates that the process requires user interaction as part of the workflow. The MDM human task supports the following user tasks:

- Generic User Task. Defines actions for users included in the workflow.
- Originator User Task. Defines actions for users who trigger a workflow with their actions.

You can use the predefined MDM Multi-Step Approval Template to design a process. For more information about the predefined template, see MDM Multi-Step Approval Template.

To configure an MDM human task in a Service step, select **MDM > HumanTask > Generic User Task** or **MDM > HumanTask > Originator User Task** as a system service action.

Note: The options other than **Generic User Task** and **Originator User Task** under **HumanTask** are reserved for future use.

The following image shows the MDM human task properties:

Input Fields

Configure the following input fields in the Service step for an MDM human task:

Template

Required. The template field that you added to the **Input Fields** tab of the Start step.

TaskTypes

Required. The taskTypes field that you added to the **Input Fields** tab of the Start step.

Change List

Required. The input field that you added to the **Input Fields** tab of the Start step.

TaskName

Required. The unique name of the Service step that you specified on the **General** tab.

Task Actions

Required. The list of actions that the users can perform with this step. Use a semicolon as a delimiter. For example, `Send for Translation;Send Back to Originator;Release`. You can also use the action text to provide more information on the next process step. The output that returns from each task action must be handled within the process.

Output Fields

You can use the **LastAction** and **LastApprover** fields to define additional process steps, such as Assignment step or Decision step, to access data and services and to perform related orchestration activities. Each process step returns information about the last approver and the last action taken by the approver.

The following fields store the step values that the additional process steps return:

LastAction

The last task action, such as Send for Translation, Send Back to Originator, or Release.

LastApprover

The identifier of the user who took the last action.

For step-by-step instructions about creating a process with a human task, see [Defining Cloud Application Integration processes for user-triggered business events](#).

Run a Shell Command

You can use the Run a Shell Command action in a Service step to automate the call to a shell command as part of your application integration. For example, you might add this action in a subprocess to move files on the agent or to read the file contents of a particular directory by running cp or cat shell commands.

Note: When you use this action in a process, be sure to select a Secure Agent where you want to run the shell command in the **Server Configuration** > **System Services** list in Application Integration Console. However, you can access a shell command from a process running on the Cloud Server by calling the shell command in a subprocess that contains this Service step. Running system commands with this step is not recommended.

Enabling the Shell Command Service

Before you can execute a process that includes this action, enable the shell service in the Application Integration Console with the following steps:
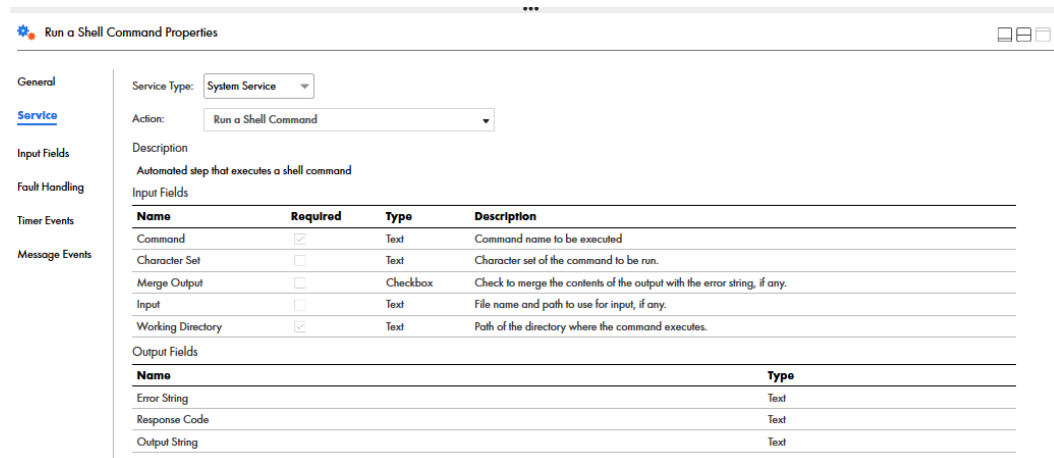
1. In Application Integration Console, choose **Server Configuration**.
2. Choose the agent where you want to run the shell service from the **Server Configuration** list.
3. On the **System Services** tab, choose **Shell Service**.
4. Click **Save**.

Input and Output Fields

When you include a Service step for Run a Shell Command, you can use the following options:

	Name	Type	Description
Input Fields	Command	Text	Required. The command name to be executed, which is specific to the operating system (Windows or Linux).
	Character Set	Text	The character set of the command to be run. Default, if not provided: UTF-8.
	Merge Output	Checkbox	When checked, the contents of the output are merged with the error string, if any. By default, this is enabled.
	Input	Text	File name with path to use for the input, if any.
	Working Directory	Text	Required. Path of the directory where the command executes
Output Fields	Error String	Text	String of error returned, if any.
	Response Code	Text	0 (success) or 1 (failure).
	Output String	Text	String of output returned, if any

The following image shows the properties required to run a Shell command:

Run Cloud Task

You can use a Service step to run Data Integration tasks. You can run mapping tasks and synchronization tasks through a Service step.

Perform the following steps to run a Data Integration task through a Service step:

1. Add a Service step to the process.

2. On the **Service** tab, select the service type as **System Service**, and then select the action as **Run Cloud Task**.

3. On the **Input Fields** tab, configure the required properties:

   - **Task Name**: The name of the Data Integration task that you want Process Designer to run. You can select from a list of mapping tasks and synchronization tasks. You can also enter keywords to search for a mapping task or synchronization task.

   - **Wait for Task to Complete**: Defines whether Process Designer waits for the task to complete before Process Designer resumes execution.

   - **Max Wait**: Defines the maximum number of seconds that Process Designer waits for a task to complete. The default value is 300 seconds.

     Important: You must enter a Max Wait value that is lesser than 604800 seconds, that is, 7 days or fewer. Process Designer waits for a maximum of 7 days for a task to complete.

   - **Fail on Cloud Task Errors**: Defines whether the process faults when the mapping task or synchronization task fails.

   The following image shows the input fields that you can configure for a Run Cloud Task Service step:

Run Cloud Task Properties			

General
Service
Input Fields
Fault Handling
Timer Events
Message Events

Service Type: System Service

Action: Run Cloud Task

Description
Start the specified Informatica Cloud Task.

Input Fields

Name	Required	Type	Description
Task Name	☑	Cloud Task	The name of the task to start.
Wait for Task to Complete	☐	Checkbox	Wait for the task to complete before returning.
Max Wait	☐	Integer	The maximum time in seconds to wait for a task to complete.
Fail on Cloud Task Errors	☐	Checkbox	Throw fault if ICS task completed with errors.

Output Fields

Name	Type
Run Id	Integer
Task Status	Text
Success Source Rows	Integer
Failed Source Rows	Integer
Success Target Rows	Integer
Failed Target Rows	Integer
Start Time	Date Time
End Time	Date Time
Error Message	Text

After you publish and run the process that contains the Run Cloud Task Service step, you can view the task execution history in Application Integration Console.

Fault Handling

When you integrate applications with Process Designer, you can handle error conditions by:

- Catching a fault in a process using boundary events.
- Returning faults from a process either with output fields or by adding a Throw step to the process to throw a fault.

Fault Handling and Boundary Events

A boundary event is an event that catches an error that occurs on the boundary of a particular step, that is, within the scope of the step where it is defined. When you enable fault handling for a step in your process, you are defining a boundary event. The fault handler listens for a fault on that step when the process executes. When the the fault is caught, the step is interrupted and the process then follows the path you define for handling the fault. In the fault path, for example, you might send an email notification and terminate the process, retry after waiting for a specified interval, suspend the process, or complete the process using an alternative path.

For example, you can catch a fault that occurs when a REST service returns an error code to the process because the host system is not accessible or because the process provided invalid data.

Note: A fault handler on a step is an interrupting boundary event. This means that when a fault occurs, the process follows the fault path and the main path from the step does not execute.

Design Guidelines for Fault Handling

When you define fault handlers, you must note the following points:

- Determine whether the boundary event occurs at a single step or in a set of steps. If the boundary event occurs on a set of steps you have two options:
  1. Add fault handling for each step individually.
  2. Define the steps inside a wrapping process.
     The latter option allows you to then add the fault handler on a Subprocess step and invoke the wrapped process.

- If you have a Parallel Path step for fault handling, be sure to provide unique field names for each field that contains fault information. Because you define the fault information ouptut field at the process level, there is a chance that fault information from one path can overwrite the fault information of another path. To avoid this, be sure to set a unique field name for fault information in each path of a Parallel Path step.

- To handle faults, you have several options:
  1. If there is an uncaught fault, suspend the process and debug it using the Application Integration Console. In this case, you must select Suspend on Uncaught Fault in the Advanced process properties. If you do not catch the fault, it will be suspended and visible in the Application Integration Console. You can also rethrow the fault after you catch it.
  2. Log the fault (for example, by sending an email notification) and continue gracefully.
  3. Add a Wait step to wait for some interval and then retry the step by adding a Jump step. However, you cannot use a Jump step from the fault path. This means you must merge back to the main process path and then add the Jump step.

- Because you get the fault detail (in the faultInfo process object) as an any type value, you can use XQuery to access the detail, provided you know the structure of the fault information returned from the service.

Faults can be triggered by:

- Service step (using a service connector or an automated step created in the process).
- Create step (a special kind of Service step available in Process Designer).
- Host system interaction.
- Throw step defined in a process.
- Other faults.

Faults Triggered by Service Connectors

If a fault is triggered by a service connector, the faultInfo process object includes a description of the error and returns this to the process.

When you enable fault handling in Process Designer, you also catch any explicit runtimeError faults that might be generated by an SOA connector. For example:

```
<wsdl:operation name="read">
  <wsdl:input message="tns:readRequest"></wsdl:input>
  <wsdl:output message="tns:readResponse"></wsdl:output>
  <wsdl:fault name="runtimeError" message="tns:faultResponse"/>
</wsdl:operation>
```

Step Types that Support Fault Handling

You can enable fault handling on the following step types:

- Create
- Service
- Subprocess

If enabled, the fault handler catches all faults on the individual step. Because it is interrupting, it creates a separate execution path in the process. You specify a fault name field so you can get all the fault details in the process object.

On the **Fault Handling** tab for each of these Step types, you can set these properties:

Property	Description
Catch Faults	Required. Check to enable fault handling on this step. By default, this option is disabled.
Fault Field Name	Required if Catch Faults is enabled. The fault name, which defaults to faultInfo.

The following image shows a service step with fault handling enabled:
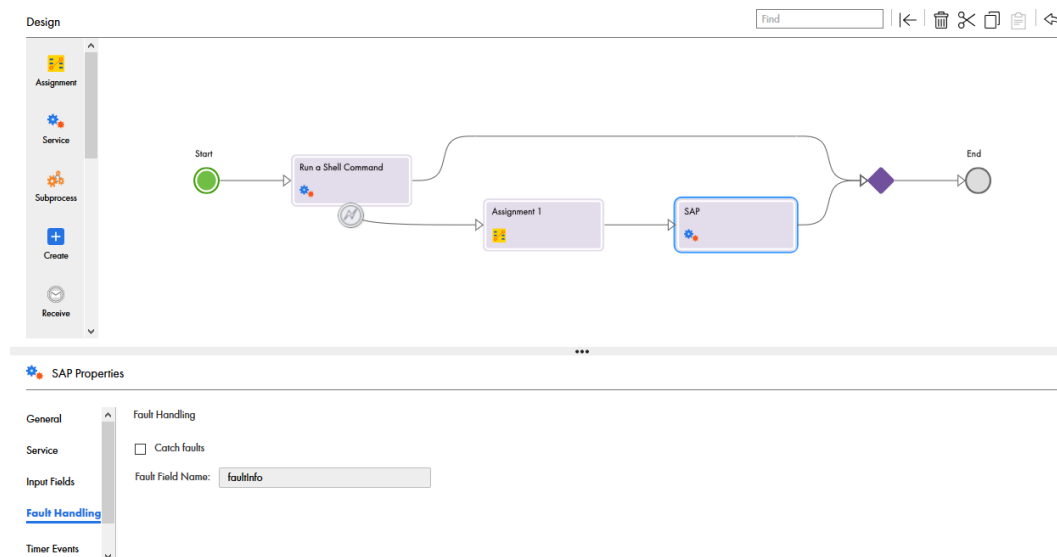
You can also configure a specific fault type, reason, and description in a Throw step. You can then call the Throw step from other processes when it is caught by a pattern or process, rather than using an output field to pass fault information.

Methods to Return a Fault from a Process

1. **Use an output**. This is suitable when you foresee that the error might occur and you want to use a data decision step to branch the process based on the normal process flow.
2. **Throw a fault**. This is suitable when the only appropriate response to the error is to pass the error up to the caller or log the error and suspend the process. For example, if you attempt to access a web service that responds with the HTTP status code, 403 Forbidden, it indicates that the service understood the request but refuses to take any action, because access is denied. Throwing a fault is normally the best option to handle this error because you are unlikely to resolve it in the normal process flow.

Throw Step Usage

When you add a Throw step to a process, the error information is propagated by the Throw step. This allows you to catch the fault when you call it, for example, through a Subprocess step.

Note that:

1. The name must be a string value.

2. Throw is a terminating step so it does not support any outbound links.

3. A Throw step can be used to rethrow information caught using a fault handler. In that case, the payload for the step is a faultInfo element that contains the parameters returned by the service, service connector, or SOA connector.

4. You can also use a Throw step to construct specific fault information that you want to use as part of your process.

Switching the Process Designer user interface

The Process Designer user interface is modified to have a modern, clean look and feel, and better fonts. You can use the modern view to experience the improved user interface. However, you can switch the Process Designer user interface between the modern view and the classic view, if required.

When you switch the view, all processes you open in the Process Designer open in the same view for the current organization until you switch the view again.

To switch the Process Designer user interface, perform the following steps:

1. On the **Explore** page, open a process to switch the view.

2. On the **Actions** menu, click **Switch to Classic View** or **Switch to Modern View** as needed.
   The following image shows the **Switch to Classic View** option:
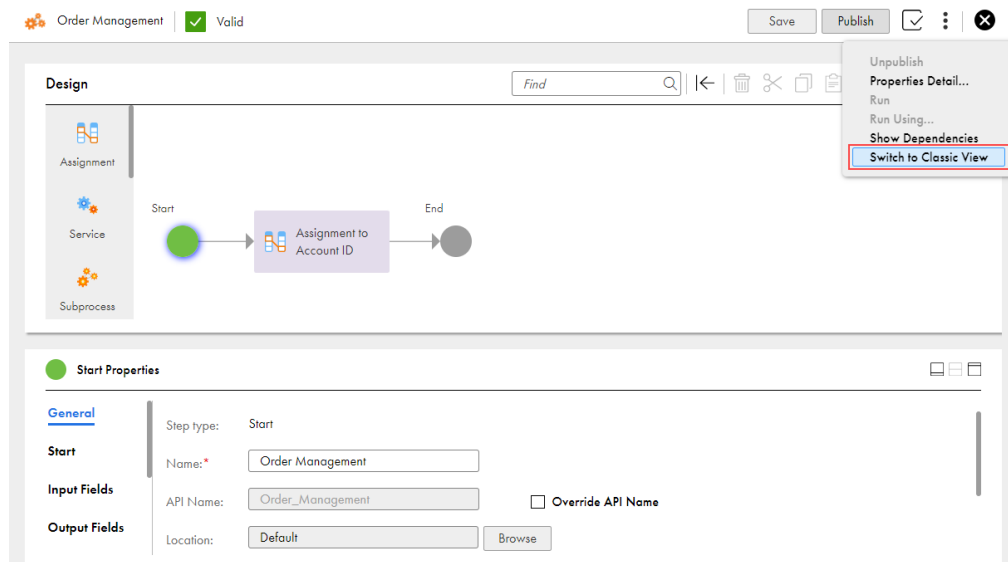
3. To switch from the modern view to the classic view, enter a reason.

4. Click **OK**.

Note: The new Process Designer user interface is available for preview. Preview functionality is supported for evaluation purposes but is unwarranted and is not supported in production environments or any environment that you plan to push to production. Informatica intends to include the preview functionality in an upcoming release for production use, but might choose not to in accordance with changing market or technical circumstances. Note that if you are working on a preview POD, all data is excluded from SOC 2 compliance coverage. For more information, contact Informatica Global Customer Support.

CHAPTER 3

Using and Displaying Data

The following diagram explains what you learn in this chapter:
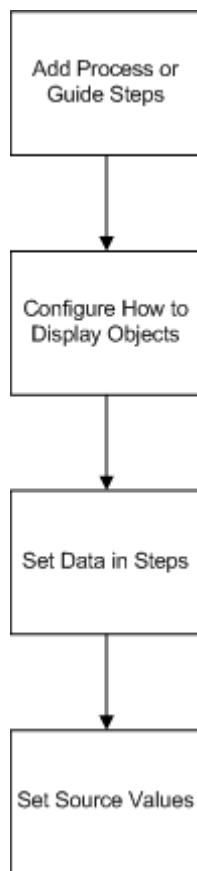
Selecting and Displaying Objects

Within a step, you may want your users to select one or more object. You can display this information in a variety of ways:

- A pop-up search dialog for searching for objects by name
- A picklist from which they can select one object
- A multi-select picklist for selecting more than one

- A table of objects, and you can select which of the object's fields it should display in separate column
  If only one object can be selected, Guide Designer adds a column containing radio buttons; otherwise, it adds checkboxes.

Note: If you are just displaying one object, you can display the object's fields as a one-row, multi-column table.

By default, Guide Designer displays a search dialog box when users are searching for information for fields that can be updateable. You can override the default and tell Guide Designer to display object information using either a picklist or table. Picklists and tables are populated using a query that filters the possible objects down to just those that should be presented. For example, Guide Designer could show only the contacts that are associated with an account.

You can customize the fields being shown for each object. If only one field from these objects is displayed, Guide Designer shows this information as a list. However, if you are displaying more than one (for example, a lead's first and last name), Guide Designer shows each within its own column in a table.

Note: Object queries are not available when the field you are updating can reference more than one type of object (that is, the **Reference To** control within the **Field Properties** dialog includes multiple object types). If this is your situation (that is, you cannot use a regular query), you must select and use an advanced query.

Inserting Fields Using Picklists

There are many places where you add or use field names.
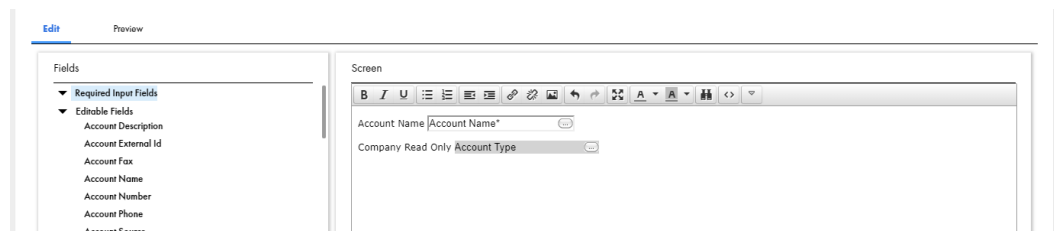
When a user runs a guide, Process Designer replaces the field with the value within the object. For example, if an Account Type field was inserted and the Account Type is "Grade 1", users will see "Grade 1" instead of the field name when they are running the guide.

Three kinds of lists exist in steps:

- **Read-Only Field**: A read-only field is a field whose value cannot be changed. You can tell that it is read-only field because it displays in gray.
  The text in front of the field was added as descriptive text in front of the field. If you didn't add a description, the user might not know what information was being displayed.

  Note: When an object ID is placed in a canvas as a read-only field or put into a column in a table, it displays as a link to the object. If you want to show the object ID's value, insert it into a text field as "`{!Id}`".

- **Editable Field**: An editable field lets the user see the current value of the field as well as modify its value.

-

Click the "**...**" button to see a field properties dialog. For more information, see [" Introduction to Data Types and Field Properties" on page 10](#).

Autogenerated Fields

A field can be virtual; that is, the object has a field that is actually a composite key or a composite field. When Guide Designer displays the name of this virtual and autogenerated field, it displays it as "`Object_name`" + "`_IID`".

Inserting Fields in Tables

When you insert a field into a step, clicking on the field's name in a picklist inserts a label and a text field into which the field information will display. If you are inserting more than one field, the step can be hard to read and does not look good. Placing the elements within a table can solve both of these problems.

Here's an example:
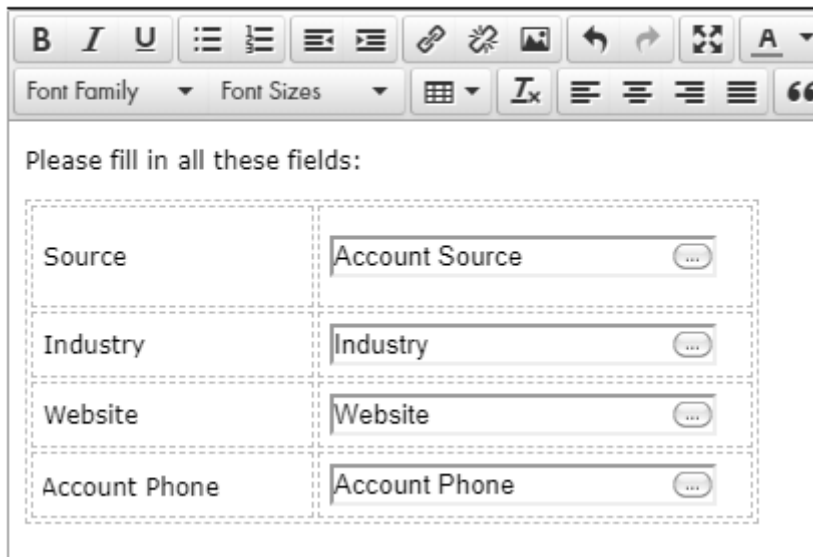
When handling fields in a table:

- If the cursor is not in the right-most cell, Guide Designer inserts the label at the current cursor position. The field is inserted into the cell to the right of the cursor. If the cell to the right has something in it, Guide Designer appends the field to this information.

- If the cursor is in the right-most cell and the cell isn't empty, Guide Designer places the label and the field into the first two columns of the row that follows. If that row has text in it, Guide Designer inserts a row immediately beneath the current row, and then inserts the label and field. If that row is empty, just write place the information into the cell.

- If the cursor is in an empty right-most cell or if the table has only one column, Guide Designer places the field (not the label) into this cell.

Entering Fields as Text

You can enter text and the value of a field as the information you type in text boxes such as the one in the following.

To specify a field value, choose Content as the Source and then click the small icon to the right of the text box. From the list of fields that displays, select the field you want to enter. If you know the name of the field, you can also type it in the text box as *{$name_of_field}*.

This example shows the values of four fields from the current object as the Content for the Description field.

You can add other text to display with the field values. For example, you could specify:

```
The name is: {$name}
```

Displaying Columns

Use the **Display Fields** field to tell Guide Designer which of the object's fields it should display in a table. If the field can be updated, each row also has either a radio button if users can only select one object or a checkbox if they can select more than one.

Use the **Add Field Column** button to add columns. You can use the arrow buttons to the right of the name to change the order. If you do not want to use the default column heading, enter the text you prefer.

Here's a four column table:

The three check boxes are as follows:

- *Paginate Result*: When selected, the output is paginated.
- *Allow Filtering*: When selected, the user can enter text in a text box to select some of the displayed results.

- *Allow User Column Sorting*: When selected, the user can click on a column heading to sort the results by that column's contents.

Note: When an ID field is included in a result from a Lookup dialog, the ID's value displays.

Setting Data in Steps

Some steps that you will create ask users to enter information before the step's actions occur. Examples include Service and Create. In contrast, other steps already have the information they need when they begin executing. If the step has the information it needs, the action occurs automatically and the user doesn't see it. Instead, the user sees the step that follows. Although this topic uses a Create step as an example, there is no difference setting data in other kinds of steps

In the step properties section, you can see information about the fields associated with the action or object. If an entity or object is not selected, no data appears in the Information tab.

This section lists all of the fields that can supply data when this step creates an opportunity object. The fields that you will actually be using are named and set within the Input tab.

Using Fields in Steps

The data types that you name in the property sheet may be numbers, text, time, and so on. These data types determine how the data is handled in a process and displayed in a guide.

The following image shows the Account Type field:

The Default Value and Show List fields appear only when you define the field in a step:

- **Default Value**: Select Content, Field, or Formula.

  - Content displays a control where you can enter a specific value.

  - Field displays a list from which you can select the default value.

  - Formula allows you to set the value using a formula you enter here.

  Note: If you select a default value for a field that is on a screen "for update", the user does not see the default value unless the field's current value was not set before the step displays. If it was set, the user sees this current value. Service Call steps differ as the default value is always shown since there is no existing value.

- **Show List**: For information, see "Selecting and Displaying Objects" on page 153.

Setting Source Values

You can tell Guide Designer about the source of the values that will be assigned to a field. The following figure shows part of a Create step where fields have different sources:

- **Add**: If the field being entered is of type `ObjectList`, selecting this item will let you append an item to that list.

- **Content**: The values used by the step is what you define here and does not change. For example, the Rating field, which was defined as having one of three values, will be set to one of these three as you are designing this guide. The value you set here is what will be used when this object is created. However, if this were text, the text may also have embedded fields. This example has three different kinds of content fields. The differ because Process Designer knows what kind of data is being set. For example, Converted was defined as a checkbox, Rating as a picklist with predefined values, and Zip/Postal Code is defined as text.

- **Field**: The value comes from a field in an object.

- **Formula**: You will use numeric operators such as:

  - +

  - -

  - *

  - div

  - mod

  You can use built-in and XQuery functions to set a value.

- **Query**: (This is not shown.) You tell Guide Designer which information it should locate. For example, Guide Designer might look for a company whose name is "Acme".

- **Screen:** (Guides only) You must insert an input field within the Input Screen tab's editing area.

Setting Source Values: Content

A Content source is one in which the person designing the guide is deciding the value that is passed as input to a step. For example, you could use Content to set the Description field of a Create step creating an event to "Demonstrate the product" rather than requiring the guide's user to provide the description.

Sometimes, you want to create content that contains text and field values. For example, you could have a "mailing" field that contains a person's name and address. You would add this information by including field strings.

Click the (x) icon to see a list if fields.

After you select a field, Process Designer places a field code into the text box at the cursor's position. You can individually select as many fields as you need; however, you can only select one item at a time.

Here's an example:

You can also add text to this inserted field indicator. For example, In front of `{$Account.name}`, you could type "`Name: `".

When Process Designer executes this step, it replaces each field indicator with the text extracted from the field.

Setting Source Values: Field

If you select Field for the Source, Process Designer displays a collapsed list to the right. You can now select the field whose value Process Designer will write into the object when this step executes. As with other places where you select fields, the bottom of the picklist may show related objects whose fields can be used.

Setting Source Values: Formula

Fields that take numeric input fields can use a formula to set a value. For example, you can use this with the Assignment step to add a number to an existing field. Use numeric operators such as +, -, *, div, and mod to perform an operation upon fields.

You can also use any XQuery function within a formula. (These are defined at http://www.w3.org/TR/xpath-functions-30/.)

Evaluation Notes

Process Designer evaluates formulas as you might expect. Note that:

- `$Field1.Field2.Field3` is pre-processed to de-reference down to the contents of `Field3` and would bind the contents to its value.

- Special characters in field names are removed. Fields that start with numbers are given an "`x`" prefix.

- If `$Account` is a by-value object, you must use an XPath path expression to get at its contents; for example, `$Account/Owner/Name`.

- Object lists should be sequences containing either ID values (for by-reference object lists) or the XML representation of the records. For example, two object lists can be appended using the following expression: `($a,$b)`.

- Object references within a list are separated by semi-colons; for example, `(a; b; c)`.

For information on available functions, see "Using Functions" on page 29.

Setting Source Values: Query

Use a query to retrieve object information from all of the objects that are selected by the Where Clause.

Query controls appear in different steps and in some dialogs so what you may see may differ slightly.

What you will be doing is creating an SQL WHERE clause. While you can enter your condition directly into the text area, it is usually far easier to click the **Add Condition** button and the Order By picklist. After Process Designer inserts this information, you can edit it if you need to. Letting Process Designer do the work is recommended as it is sometimes not obvious what the field's internal name is. The text entered here is a standard SQL WHERE clause. Also, this text does not include the WHERE keyword.

Each condition has four parts:

- The name of a field in the object. In this example, the field is one of those contained within the object.

- An operator that Process Designer uses when it compares the field value on the left with values on the right.

- The kind of data that will be compared. Your choices are Content, Field, or Formula. If you choose Field, you are comparing the contents within the first field in the current object (the one on the extreme left of the **Condition** dialog) with the contents of the field in the type of object being queried.

- The data to which the field on the left is being compared. If the source is Content and you click within this area, a small icon appears to the right. After selecting it, Process Designer displays a picklist from which you can select the name of a field in another object that will be used when making the comparison.

Pressing the *Order By* picklist button lets you select the fields that the query sorts upon when it displays information. For example, you might want to sort the displayed information by a person's last name. You can add additional keys by reselecting a field from this picklist. The *Order By* clause is placed within the text box. This lets you edit it after it is inserted. You can also directly type the Order By clause here as well.

Note: You can add an SQL LIMIT clause to your query by entering it in the Where clause text box. For example, you can limit the number of retrieved rows to 200 by typing "Limit 200". When adding this clause, be sure to add a space after the where information or place it on its own line after the where. ("Limit" can be in upper, lower, or mixed case.) By default, 100 rows are returned. If your guide requires more than 100, you must add this clause.

Specifying Conditions in a WHERE Clause

Your ability to define a condition within a WHERE clause depends upon what Produce Designer is interacting with.

Direct Interaction with a Relational Database

Most databases fully support the SQL WHERE operator. If you encounter problems, you should consult the documentation for that database.

Using an Informatica Connector

If you need more than one condition, only simple cognations (field operator values) are supported. You may join multiple conditions using either AND or OR. You can also use the NOT operator to invert the meaning of the condition.

Salesforce

The WHERE condition must conform to SOQL requirements. Consult the Salesforce SOQL documentation for more information.

Using a WHERE Clause When No Answer Is Expected

One common situation, and a situation that isn't at all unique to Process Designer, is that you are creating a WHERE clause that is based on a multiple criteria. However, not all of the criteria will have values. For example, an account and a territory, and the territory is optional. Setting up a WHERE clause such as `Account={$account) AND Territory={$territory}` wouldn't work if no territory value was used.

The solution for creating a WHERE clause with fields that are optional is to use the LIKE operator; for example, `Account LIKE '%{$account}%' AND Territory LIKE '%{$territory}%'`. If territory doesn't exist, this would be the same and entering `Account LIKE '%{$account}%' AND Territory LIKE '%%'`. The LIKE following the AND will always evaluate to true.

Setting Source Values: Screen

If you select Screen for the Source, the end user provides data for this field. This means that you must add a field to the step's Input Screen tab so that the guide's user can type information. In the following example, the source on the Input tab for three fields was set to Screen:
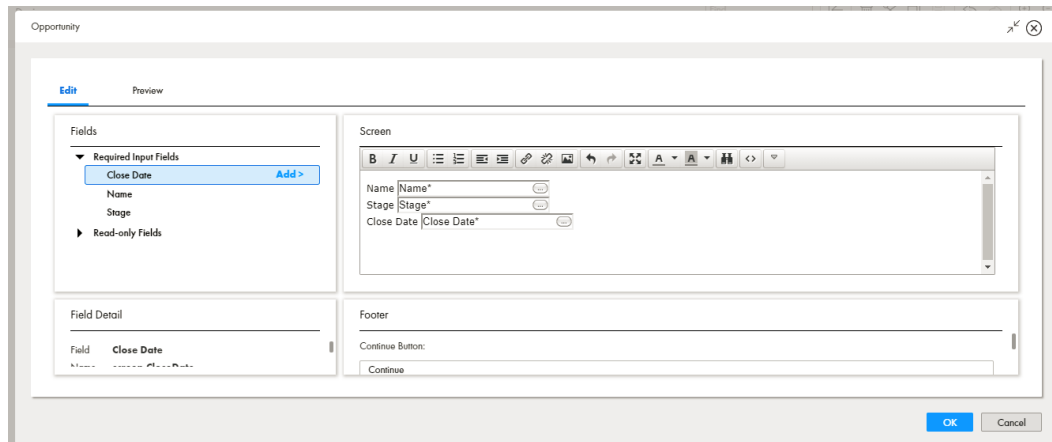
Note: The text you enter will look better if placed in tables.

Like all fields, you can click within it to display a properties dialog.

CHAPTER 4

Designing Guides

A guide is a set of screens that prompts users to review, enter, or confirm data. For example, a step might display account details or prompt the user to confirm the status of a sales call. Behind the scenes, steps interact with your application by extracting and storing data. Guides run within mobile apps or on traditional platforms such as a PC or a Mac.

The following image shows the guide design process:
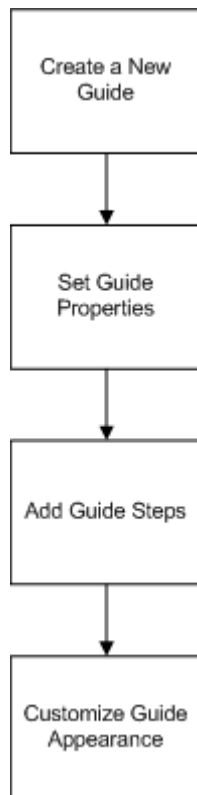
You can create guides without technical expertise or formal training.

You want to automate the process a sales manager follows to create a new account or update an existing account in Salesforce.

To create the guide, you perform the following general steps:

1. Create the screens that contain questions and possible answers. As the options are defined, the sequence of steps from the script appear as a flowchart in the canvas.
The following image depicts the guide:
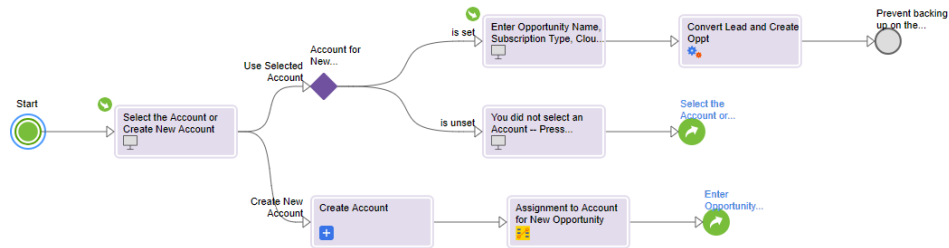
2. Insert optional automated actions. For example, call a process that converts a lead and creates a new opportunity. When you run the guide, if the guide takes the appropriate path, the process is called.

3. Publish the guide so that users can access it for sales activities.

Creating a Guide

Perform the following steps to create a guide:

1. In Application Integration, select **New**.
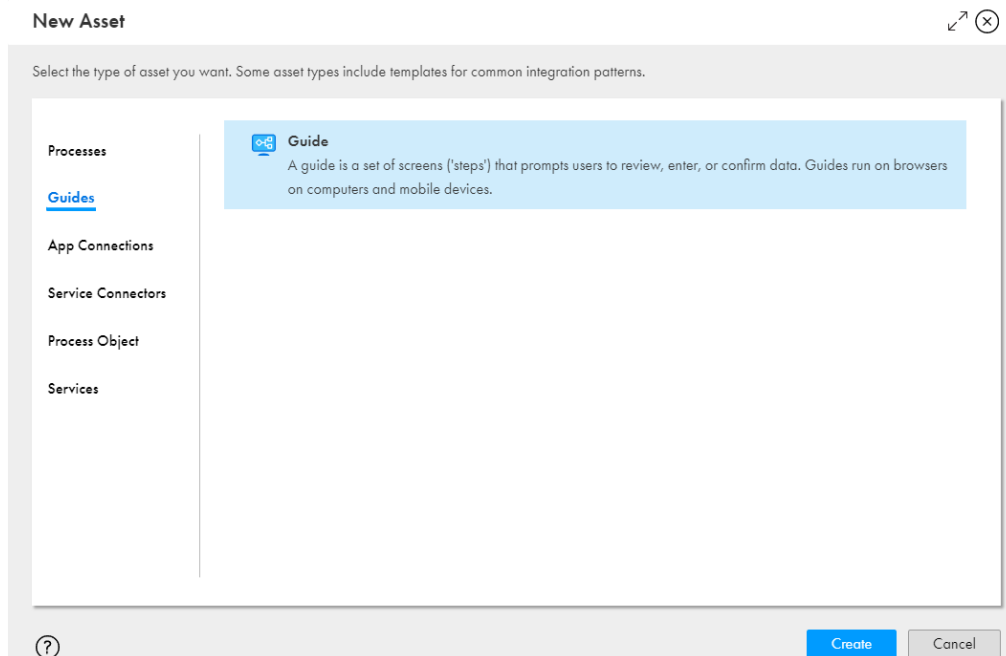
2. In the **New Asset** dialog box, select **Guides** > **Guide**, and then click **Create**.
   A guide template opens.
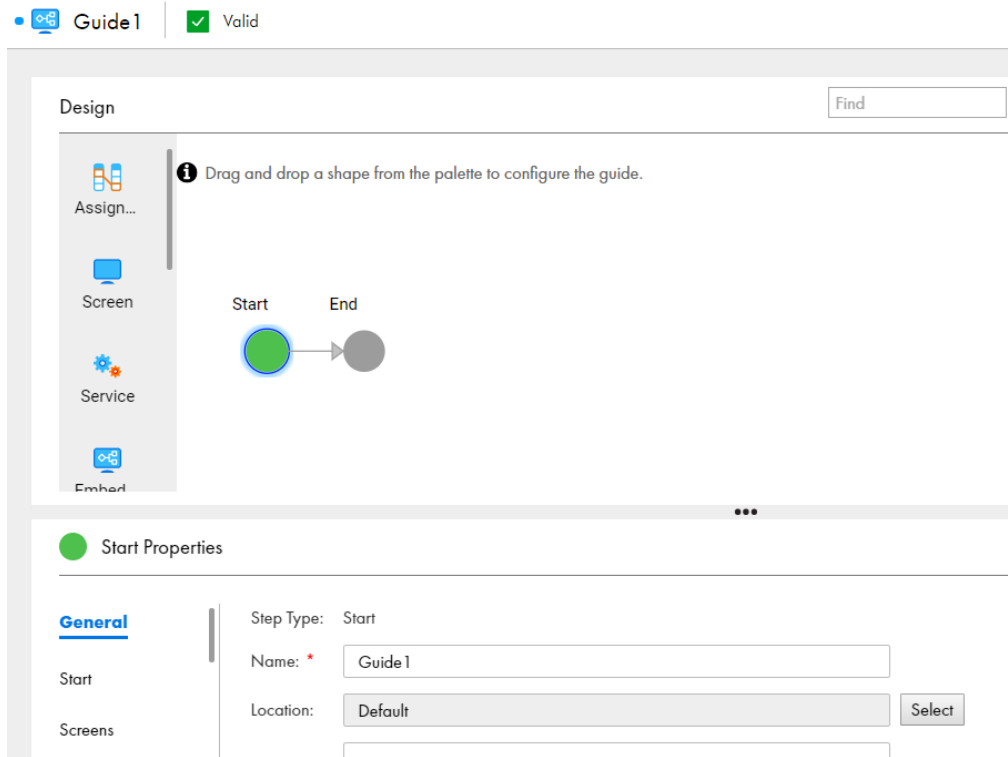
3. Enter guide properties and add steps.

Setting Guide Properties

1. Select the **Start** step.
2. Enter properties on the following tabs:
   - General
   - Start
   - Screens
   - Input Fields
   - Output Fields
   - Temp Fields
   - Outcomes
   - Advanced
   - Notes

General Properties

You can specify the following General properties for guides:

Property	Description
Name	A descriptive name to identify the guide in Guide Designer and the name that appears when the guide is available for use in other objects such as process call steps. The name can contain multibyte characters and must not exceed 80 characters. To change the name of a published guide, you must first unpublish the guide. Then, change the name and republish the guide.
Location	The location of the project or folder you want the process to reside in. To select a location for the guide, browse to the appropriate project or folder, or use the default location.
Description	A description of the guide.

Start Properties

You can define the following Start properties for a guide:

Property	Description
Applies To	The type of object associated with the guide. The objects that display in this list are the process objects you defined separately, or generated in a defined connection. Drill-down in the list to see process objects and connections, and then select one of these items. Select a process object only if the guide will be embedded in another guide. **Note:** To change the **Applies To** property of a published guide, you must first unpublish the guide. Then, change the **Applies To** property and republish the guide. Choose **Any** if you do not want to associate the guide with a specific object. When you select this option, the guide does not automatically have access to an object's fields. Use this object type when a guide creates a new object and does not access information within existing objects. **Note:** If you run a guide, the **Applies To** setting must be either **Any** or a specific object in a service, but not a process object.
Run As (Salesforce only)	For Salesforce only, you can choose whether to run the guide as the Current User or System. Choose **Current User** if the guide should have the same privileges as the user who is running it. Choose **System** if the guide should have system-level permissions to enable actions that the guide user may not otherwise be able to access. For example, if a user does not have access to account objects and the guide is running as System, the guide can still access the account objects to the extent needed to run the guide.

Screens Properties

Use the **Screens** tab to configure where you want the guide to display and the languages you want the guide to support.

You can define the following properties for the Screen step:

Property	Description
Intended Display	The environment in which the guide will run. Select **Browser** if you want the guide to run on a browser of the following devices: - Desktop - Laptop - Tablet Select **Mobile App** if you want the guide to run on the following mobile applications: - Informatica - Salesforce S1
Support multiple languages	The list of languages supported by the guide for localization. Select a language from the **Add Language** list.

Guide Translations

Use the **Start** > **Screens** > **Support Multiple Languages** option to select the languages that you want the guide to appear in.

This section discusses the following topics:

Select **Add a language** to see the list of languages.

The following image shows a sample **Languages** list:

By default, all languages use the name and description of the Default language. If you want translated names and descriptions to appear, enter the **Guide Name** and **Guide Description** against each language.

To remove a language, click **Delete**.

You cannot delete the **Default** item.

The following image shows the **Properties Detail** dialog box for a guide that has added languages:

Properties Detail for Close Lead

Basic

Unique Name:	Close_Lead-1
Publication Status:	✓ Published
Published On:	2018-07-03 15:38
Published By:	sabraham@informatica.com
Applies To:	* Any *
Run As (Salesforce Only):	$current
Languages:	Default: Catalan(ca), Default: French(fr), Default: Spanish(es)

Creating Content for a Translated Screen

Select a language from the **Actions** option, as shown in the following image:
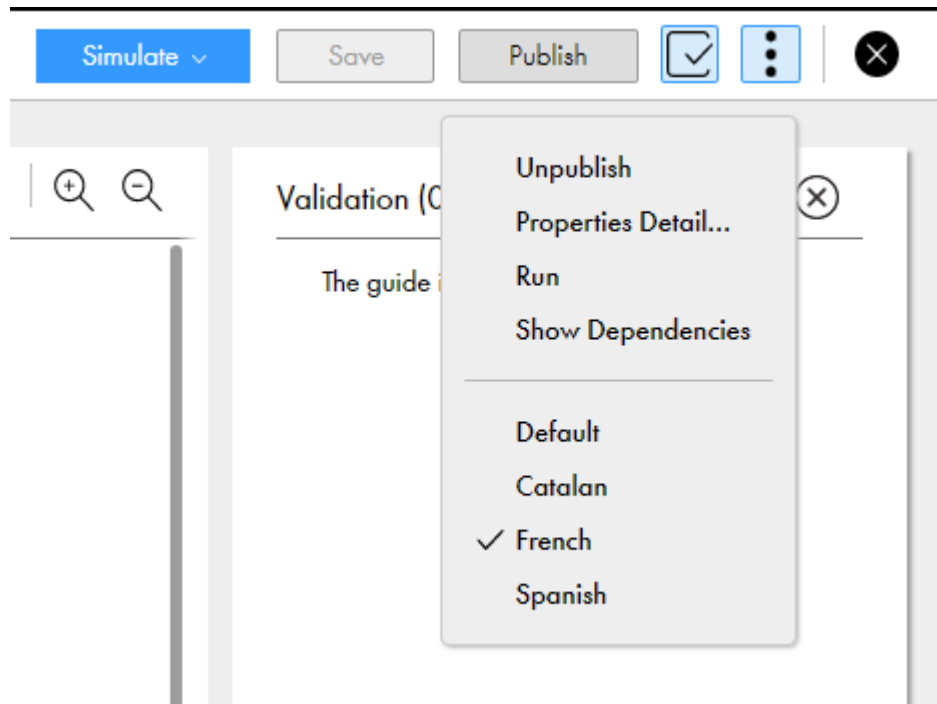
To edit the content of a translated screen, perform the following steps:

1. Go to **Actions** and select the language.

2. Select the step that you want to edit.

3. In the step properties section, click **Preview Screen**.
   You see a window with sections for the Default language and for the translated language.

   Note: The **Edit Screen** and **Preview Screen** options are available for all steps except for the Decision and Jump steps.

   The following image shows the default screen on the left and the translated screen on the right:
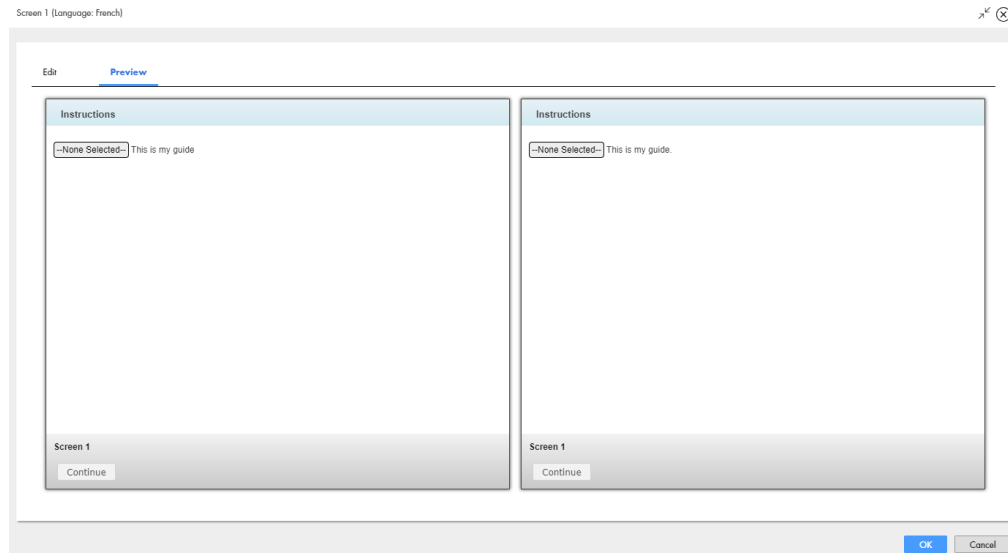
4. Select the **Edit** tab and enter the content that you want the translated guide to contain.
   The following image shows the **Edit** tab with content entered:
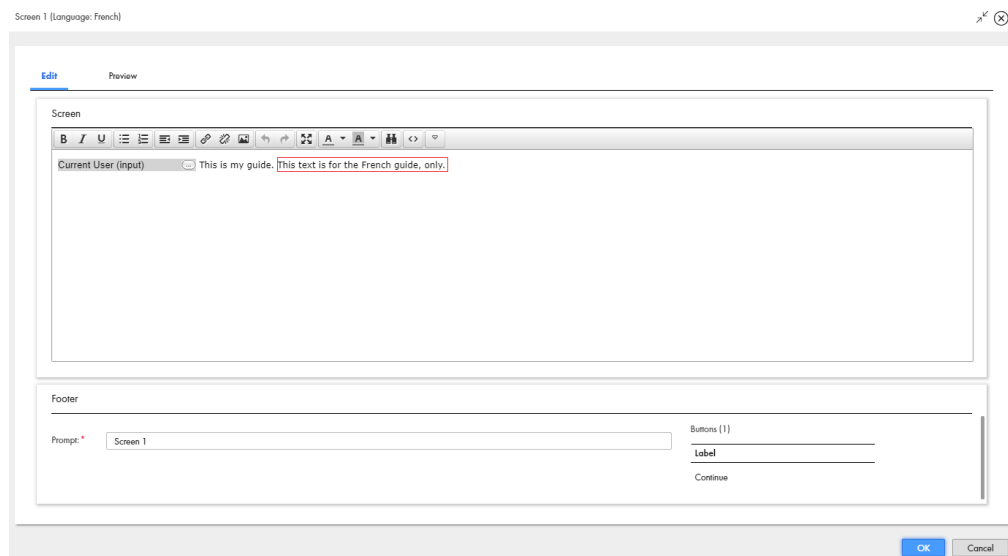
5. Select the **Preview** tab.

    The following image shows that the content that you entered appears only in the French screen:
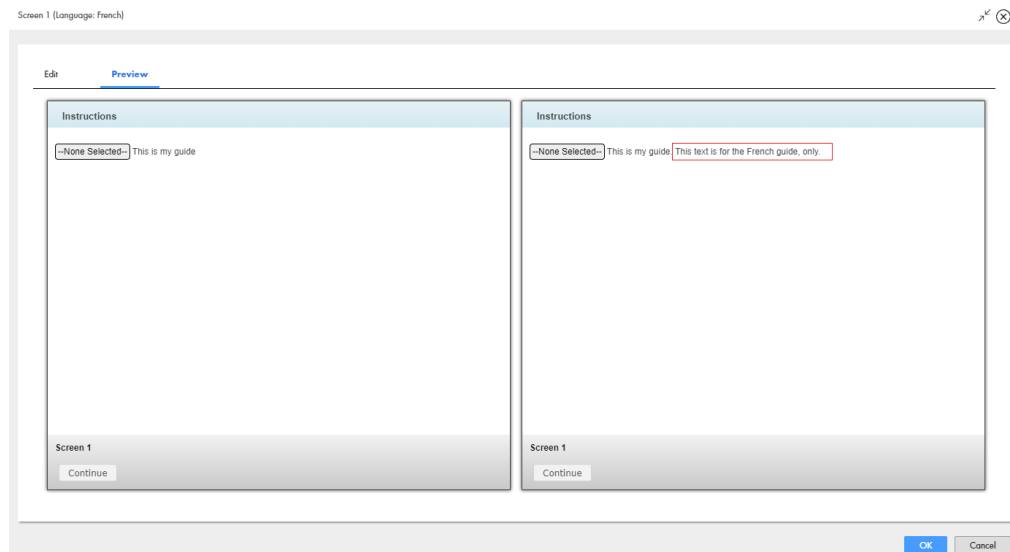
To make changes that apply to all guide languages, select **Default** from the Actions option, select a step, and then click **Edit Screen**.

The following table lists what happens to a translated screen when you make a change to the default screen:

Change Made in the Default Screen	Action in Translated Screen
Add a field	The field is added to the bottom.
Delete a field	Text is added to the translated screen telling you that a field was deleted. You will need to delete this inserted text from the translated screen.
Add an answer	The answer is added. All translations use the default answer until you translate the answer for each language.
Delete an answer	The answer in the same position is deleted.
Reorder answers	Answers are reordered identically.
Add text	Text added to the Default is added to languages that are not translated.
Delete text	Text is deleted from languages that are not translated.

What You Can Change in a Translated Screen

You can only change text within the Instruction, Prompt, Title, and Answers areas. You can also change the layout of information within the translation screen. For example, you can place two bulleted items in default screen within a table in the translation screen.

You cannot change text that is part of the guide. Examples are "Instructions", "Restart", "History", and so on.

Behind the scenes, Application Integration keeps the default and all translated screens synchronized, ensuring that the basic structure between the default and all translated screens is maintained. It prevents you from changing the step type, adding a field, or reordering answers by not allowing you to use them. You are

prevented from some actions by controls not being selectable. However, you can still accidentally delete a field or an answer. If you do delete a field or answer, Guide Designer displays an error box and reinserts what was deleted and tells you it did this. If there is text associated with something you accidentally deleted, the information from the Default is copied. For example, if you had removed an answer, Guide Designer inserts the text of the answer that was in the default into the translated screen.

Simulating a Translated Screen

To view how a user will see a translated screen, perform the following steps:

1. Go to **Actions** > **Select Language**.

2. On the canvas, select the step you want to view.

3. From the **Simulate** list at the top of the page, select **From selected step**.
    The following image shows the Simulate list:
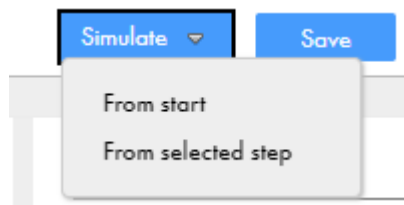
The following image shows the simulated step:
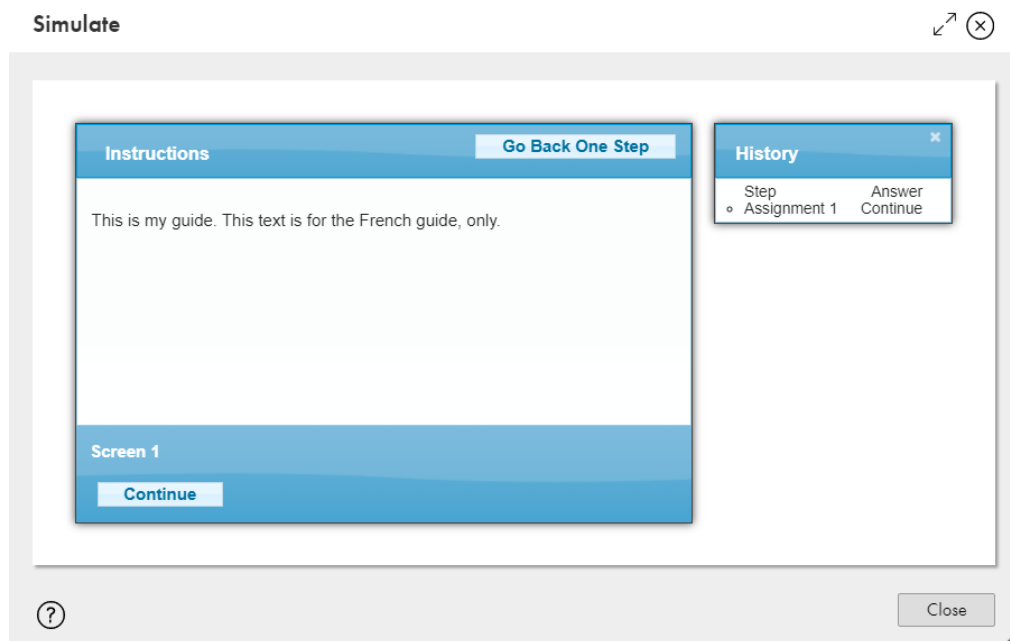
What Do End Users Do to Select a Translated Guide?

End users do not need to select a translated guide. Application Integration uses the browser locale to select a language to display.

Salesforce only: Salesforce users are always associated with a language. Depending upon which version of Salesforce you are using, you can see this setting within the Personal Information section of "My Personal Information" or within the Languages & Time Zone section of "My Settings | Personal". Guide Designer checks this value and if a translated guide exists for that language, it displays that guide.

Input Field Properties

The input fields available for a guide are those contained in the object(s) to which the guide applies. This is specified in the **Applies To** property on the **Start** tab.

The Input field values are available in all steps of the guide. An embedded guide may need to access information from the embedding guide. Creating input fields is also a way to define this information so that it can be passed to the embedding guide.

The following table describes the name and type for input field category for use in a guide:

Property	Description
Name	The name of the input field.
Type	The type of the input field. For example, select **Date**, **Date Time**, **Time**, **Integer**, or **Text**. You can also add an input field of a simple type, custom type, or connection defined type. To do this, select **More types**, and then in the **Edit Type** dialog box, select the **Category** as **Simple Types**, **Custom Types**, or **Connection defined Types**.
Description	The description for the input field.
Required	If the field is required for a guide to execute, select **Required**.

For information about using input fields in embedded guides, see .

For more information about fields, see .

Output Field Properties

The output fields available for a guide are those contained in the object(s) to which the guide applies. This is specified in the **Applies To** property on the **Start** tab.

The output field values are set when the guide executes and are then available in the steps that follow. Define output fields only for use in an embedded guide. These output fields do not appear in lists until the values are returned from an embedded guide.

The following table describes the name and type for the output field category for use in a guide:

Property	Description
Name	The name of the output field.
Type	The type of the output field. For example, select **Date**, **Date Time**, **Time**, **Integer**, or **Text**. You can also add an output field of a simple type, custom type, or connection defined type. To do this, select **More types**, and then in the **Edit Type** dialog box, select the **Category** as **Simple Types**, **Custom Types**, or **Connection defined Types**.
Description	The description for the output field.
Initial Value	The initial value for the output field.

For information about using output fields in embedded guides, see .

For more information about fields, see .

Temporary Field Properties

The temporary fields available for a guide are those contained in the object(s) to which the guide applies.

The temporary field values can be used in all of the current guide's steps. However, a temporary field's value is not available to an embedded guide.

The following table describes the name and type for the temporary field category for use in a guide:

Property	Description
Name	The name of the temp field.
Type	The type of the temp field. For example, select **Date**, **Date Time**, **Time**, **Integer**, or **Text**. You can also add an temp field of a simple type, custom type, or connection defined type. To do this, select **More types**, and then in the **Edit Type** dialog box, select the **Category** as **Simple Types**, **Custom Types**, or **Connection defined Types**.
Description	The description for the temp field.
Initial Value	The initial value for the temp field.

For more information about fields, see " Introduction to Data Types and Field Properties" on page 10.

Outcomes

When a user finishes the last step in the guide, you may want to note how it ended. These endings are called 'outcomes'.

Use the **Start** > **Outcomes** to define possible outcomes.

In the following image, six outcomes are defined for a guide:
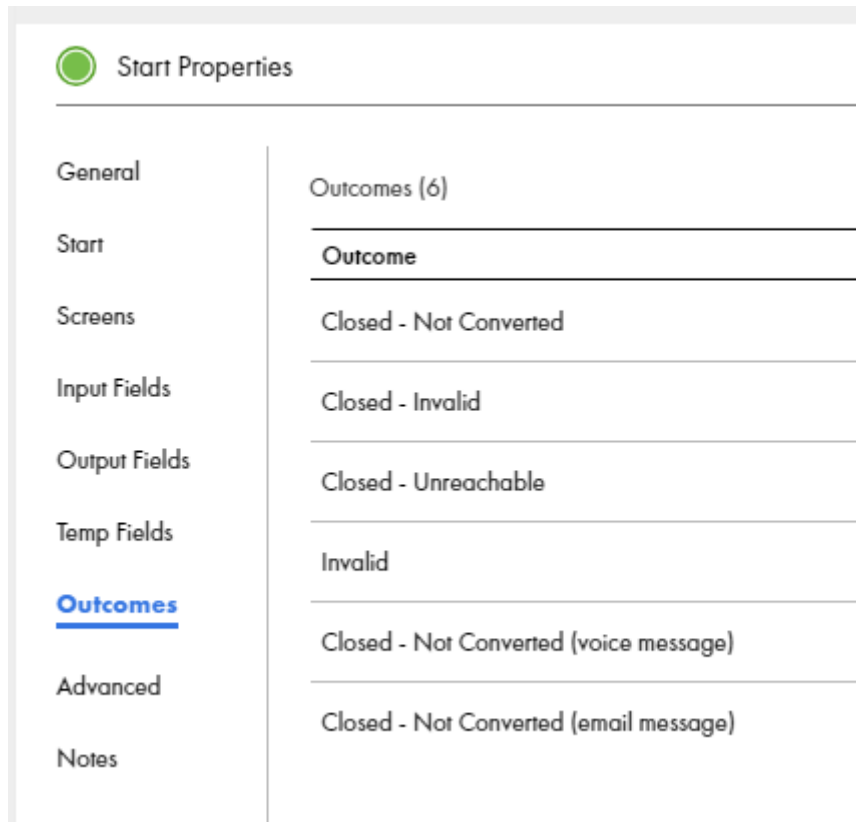
The following image shows the canvas view of a guide with part of a guide with four outcomes:
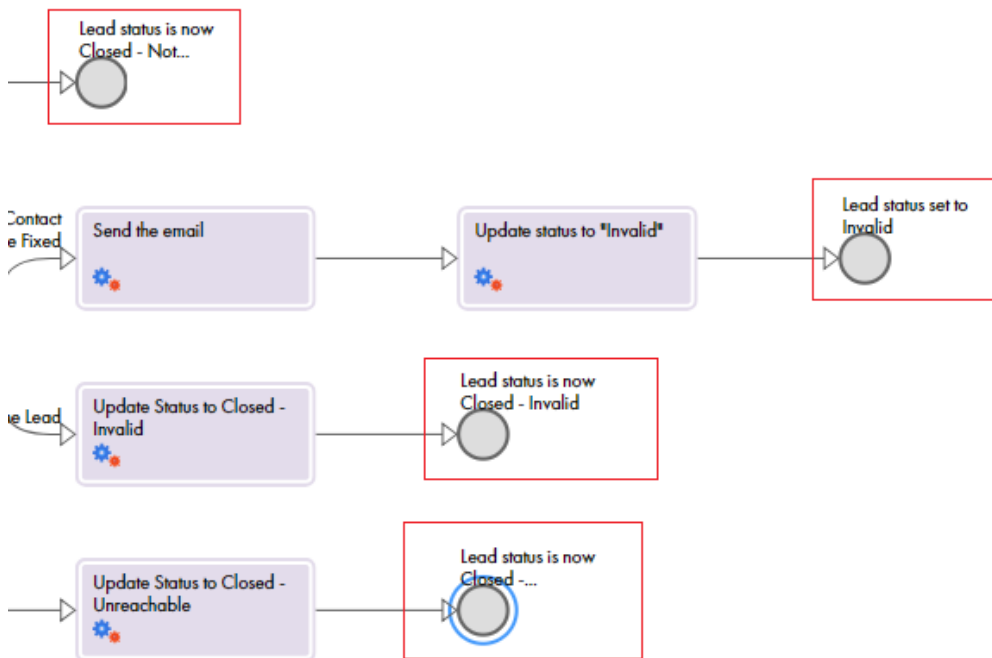
You can also add an outcome while editing an End/Milestone Stepon page 183 by selecting **Add new outcomes** in the list of available outcomes.

Advanced Properties

Use the Advanced tab to set the following properties:

Allow Restart

Determines whether users can restart this guide from the first step. If you enable this option, when the guide restarts, it undoes previously executed steps.

Execution Logging

If checked, Application Integration logs details about the guide steps and execution, so it is available for reports.

Rollback Screens

If selected, the previous input values are reset when a user clicks **Go Back One Step** or clicks a step in the History panel.

If not selected, the previous input values are not reset.

If you select this option for an embedded guide, the reset is applicable only to the scope of the embedded guide, unless the parent guide has also enabled this option.

Always Start New

If checked, opening a guide always creates a new instance of the guide.

Feedback email list

If you want users to send feedback, add one or more email addresses (separated by commas) to receive feedback. Users can then click the feedback option on the guide to display the following dialog:

The dialog shows the guide's name and the step being executed.

Show Guide Only When (Salesforce only)

You can specify criteria that determine when this guide displays in the "Guides" area on a user's Salesforce page.

The following image shows the **Advanced** tab:

Show Guide Only When

Note: This feature only applies to guides that execute within Salesforce.

You can determine which guides display within the "Guides" area on a user's Salesforce page by entering criteria in the guide properties. Enter the criteria in the **Show Guide Only When** field on the Guide Properties **Advanced** tab.

The criteria must be entered as a Boolean expression. You can use the functions and operators described in the Salesforce documentation. See the examples below.

Use the **Insert Field** list to select insert objects and fields into the expression.

This expression can be as complex as you need. For example, here is an expression that tells Guide Designer that it should only display a guide when an account's owner is one of three people:

```
CONTAINS($Account.Owner.Name, "John Smith")
    || CONTAINS($Account.Owner.Name, "Bob Smith")
    || CONTAINS($Account.Owner.Name, "Jim Smith")
```

Note: Do not use Salesforce field syntax (enclosing field names in curly braces with the "!" prefix) when you enter field names. The expression is passed directly to Salesforce as an Apex expression. If you pass an invalid Apex expression, Salesforce returns an error message.

Example 1

If the field is a list, you can use an equals sign to select one of the items from the list. For example:

```
Account.Type = 'Analyst'
```

Example 2

To define multiple criteria, use "&&" (logical AND) and "||" (logical OR) in your expression. For example:

```
(Account.Type = 'Analyst' && Account.testfield__c= true) || (Account.Phone ='1234')
```

In this case, the expression means that:

1. The account type must be Analyst AND the testfield must be true, OR

2. The account phone number must be 1234.

In this case, the parentheses group the first two tests so they are evaluated as a single unit.

In the following example, the parentheses cause this expression to be evaluated differently:

```
(Account.Type = 'Analyst') && (Account.testfield__c= true || Account.Phone ='1234')
```

Here, the same criteria means that:

1. The account type must be Analyst AND

2. Either the testfield must be true OR the account phone must be 1234.

Notes

Use the **Notes** tab to add information that you or other developers might need. The Notes you enter display only at design time. The guide or process users or consumers do not see the notes. For example, you might add a reminder about something that needs to be done before you deploy a process or guide.

Adding Guide Steps

Add steps to access services and data, and to perform multiple related activities.

When you add a step, you also define properties for that step.

Perform one of the following tasks to add a step to a guide:

- Drag a step from the pallette on the left onto the canvas.
- Double click on the canvas and select a step from the **Step Type** list.

Assignment Step

Use the Assignment step to set a value to a field. To create an Assignment step, click a process, and then select **Assignment** from the Design canvas. Then, from the Assignments properties panel, click the **Add** icon to assign Name and Assignment values.

The following image shows an Assignment step:

To add a field, click the **Add** icon, and then add the following information for each field:

Name

This is the fully-qualified field name. The name can contain only alphanumeric characters, underscores (_), spaces, and Unicode characters.

You can select a field name from the list of fields you defined under **Start** > **Fields**.

Assignments

This is the source from which the field takes values. The fields you see depend on the data type you defined under **Start** > **Properties** > **Input Fields or Temporary Fields**.

For example, if you define the data type in the Input Fields as Date or Time, the following options appear for that field in the Assignment step:

- Specific date
- Days from today
- Days before/after
- Screen
- Field
- Formula

If you define the data type as Integer or Text, you see the following options to specify the value of the field:

- Content
- Screen
- Field

- Formula

For more information about data types, see "Using the Field Properties Dialog " on page 20.

Create Step

Use the Create step to add new object instances. For example, if you work with an Account object, you can use the Create step to add a new account.

The Create step name can contain only alphanumeric characters, underscores (_), spaces, and Unicode characters.

The following image shows a Create step:

Perform the following tasks to create new objects:

1. Choose the **Input Fields** tab and select the fields that you want to add to the object. Some are required, some are optional.
2. Set the source to one of the following values: **Content**, **Field**, **Formula**, or **Screen**.

Use the **Edit Screen** option to place the fields and labels or prompts for the user, if any.

Using Reference Fields

Note the following information about reference fields:

- If you do not include a reference field on the **Input Fields** tab for the object you are creating, Guide Designer uses the value of the current object at runtime.
- Reference fields for the *Applies To* object are set if they are not set explicitly in the Input tab.
- When the content of a reference field is set to empty, it is not included in the create statement. This prevents problems that might occur when explicitly setting a field to empty or null could violate data constraints.

Screen Step

Use the a Screen step to perform the following tasks:

- Show instructions about what is being displayed, what to enter, and so on.
- Update an object's data based on user input.

The Screen step name can contain only alphanumeric characters, underscores (_), spaces, and Unicode characters.

The following image shows a Screen step that contains what a sales rep would say to a prospect:
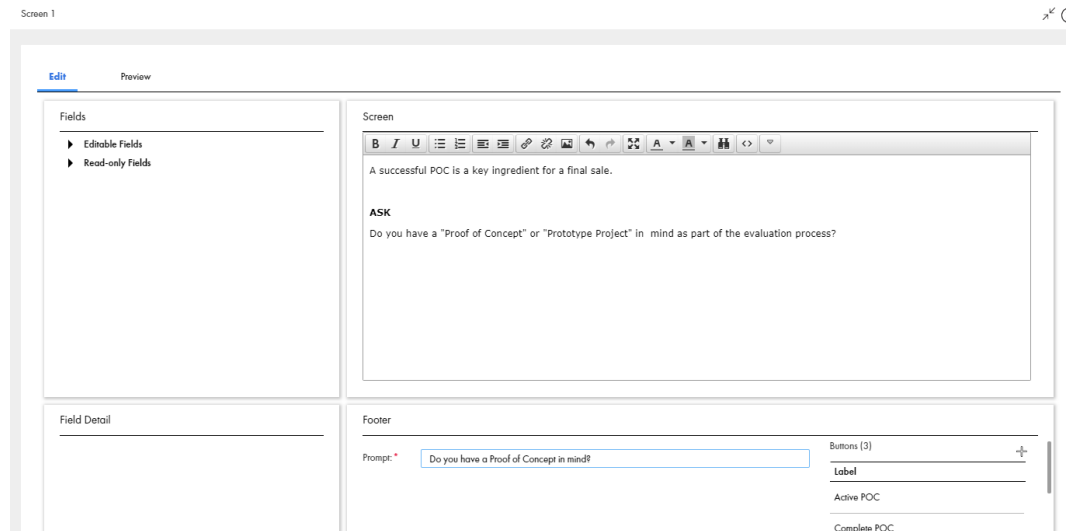
When the sales representative runs the guide, they see the question that they need to ask as well as buttons for each possible answer.

The following image shows a part of a simulated Screen step:
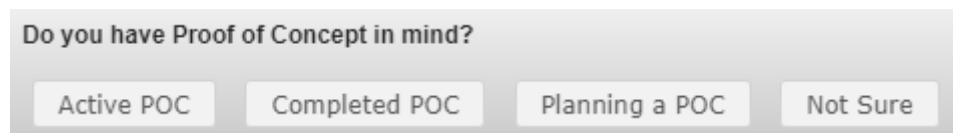
After you add a Screen step with different choices, Guide Designer creates a branch for each answer.

The following image shows a Screen step with four possible answers:
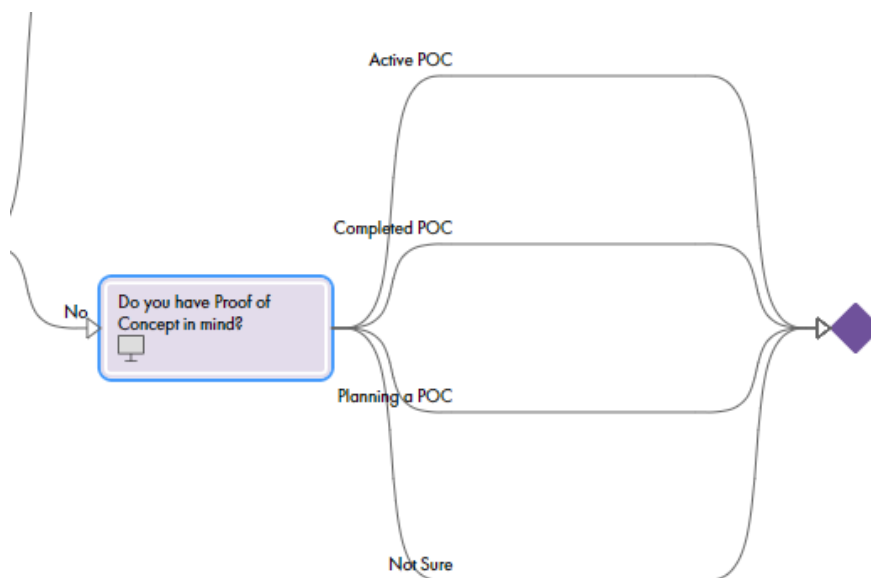
You can add steps to each branch of the Screen step.

The following image shows a Screen step with two answers:
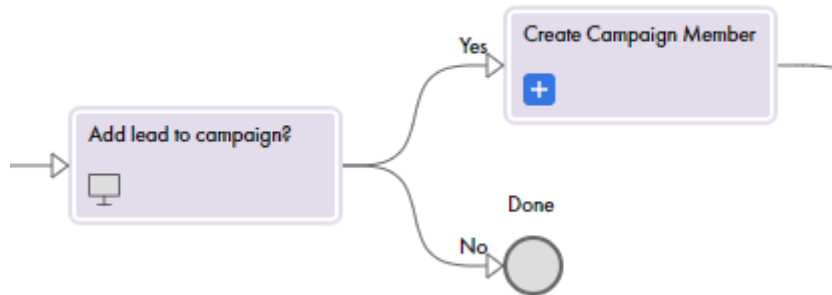
You can insert the following field types when you edit a Screen step:

- Editable Fields: Fields that the user can edit. You can proceed to the next step even if you do not enter values in these fields.
- Read-only Fields: Fields that the end user cannot edit.

To create a translated version of a Screen step, see Guide Translationson page 166.

Decision Step

The following table describes the properties in a Decision step:

Property	Description
Name	The name of the Decision step. The name can contain only alphanumeric characters, underscores (_), spaces, and Unicode characters.
Decision	The guide takes a decision based on the fields and paths you define here. Select a field name from the list of fields you define under **Start** > **Fields**. Enter conditions and values that you want the Decision step to base a decision on. The conditions available depend on the field that you select. For example, if you select a field of type **Simple** > **Text**, the following conditions are available: - Equals - Starts With - Ends With - Starts with any of - Contains You can enter text values against the conditions you select. You can add multiple conditions to a Decision step. Each condition is a potential data path. For each path that you add, a corresponding branch appears on the UI. Drag branches to rearrange the order in which the branches appear on the UI. Most Decision steps have an Otherwise path. This path handles execution if no data meets the conditions in your tests.

Evaluating Paths

A guide evaluates conditions based on the criteria you specify. Ensure that you construct paths with non-intersecting conditions.

For example, you create a Decision step with the following paths:

- Path 1: Field less than or equal to 100.
- Path 2: Field less than or equal to 75.
- Path 3: Field less than or equal to 25.
- Path 4: Otherwise

If the integer field for which the Decision step was created has a value of 25, the Decision step takes path 1. This is because 25 is less than 100 and path 1 is the first option.

To ensure that the Data Decision step follows the "Field less than or equal to 25" path, re-create the paths with the following criteria:

- Path 1: Integer between 0 and 25
- Path 2: Integer between 26 and 75.
- Path 3: Integer between 76 and 100.
- Path 4: Otherwise

Important: The guide evaluates conditions in a top-down manner. Ensure that the Otherwise branch is the last path.

A Decision step can lead to another Decision step. For example, a branch could run if an annual income exceeds $100,000. The next decision test along the same path could test if the city is Boston, or otherwise. Using this technique, you use Boolean AND logic because you base the test for the second condition on the true branch of the first condition. In this example, you use the Decision step to set the condition "Annual Revenue exceeds $100,000 AND city is Boston".

Similarly, to support Boolean OR logic, you can add a test for the second condition on any branch.

Jump Step

When you create a guide, one set of choices might require some activities and a branch would occur. After these activities are completed, the flow of steps should rejoin the actions of another branch.

For example, a guide that leads a sales rep through a conversation with a prospect could take many different branches as the rep talks about features and overcomes objectives. However, at some point in some of these branches, the rep should schedule a meeting. The scheduling step and the steps that follow may be same. Rather than add a scheduling step to each branch that needs one, you could instead jump to a scheduling step that has the scheduling steps.

Set the following property when you create a Jump step:

Property	Description
To	The target of the jump. Select from a list of available steps.
	More than one step can jump to the same target step. To see how many Jump steps have a particular step as their target, place the cursor over the arrow next to the target step.
	The Jump step has the following restrictions when used in conjunction with the Parallel Paths step: - You cannot jump to any branch of a Parallel Path step from outside of the Parallel Paths step. - You cannot jump to outside a Parallel Path step from within the Parallel Path step.
	You can jump from one branch of a Parallel Path step to another branch.

Process Call Step

Use a Process Call step to invoke a process from within a guide.

The Process Call step name can contain only alphanumeric characters, underscores (_), spaces, and Unicode characters.

Guides need not be self-contained. Most guides embed externally created sets of actions, as follows:

- Process Calls. The actions defined in a process are placed within the current guide.

- Service Calls. The actions defined in a service are placed within the current guide.

- Embedded Guides. Another separate guide is placed within the current guide.

The way in which you use a Process Call step is the same as for using an embedded guide step. See Embedded Guide stepon page 181 for more information.

Embedded Guide step

Use an Embedded Guide step to better orchestrate and manage your business processes. Design guides that embed other guides, similar to how you would create a subprocess that is called from another process.

Using an embedded guide has the following advantages:

- **Avoids Repetition**: If you place commonly used patterns in a separate guide, you can save time and create other efficiencies if you place the discrete set of steps into a guide that can be embedded and reused in other guides. When you change the embedded guide, you can easily make those changes available to all guides that use it.

- **Clarifies Design**: Embedding can make your design look simpler. Removing steps and placing them into an embedded guide means that the embedding object only shows one step rather than all of the steps. This can simplify the diagram on the canvas. Also, if a problem occurs in the ways steps occur, embedding can make it easier to locate a problem.

- **Clarifies Logic**: If there are many steps that need to operate on other objects, this allows you to create logical chunks of work that each apply to different objects (such as opportunities and leads). This decreases the complexities of working with multiple object types in a single guide.

You handle the data received from a guide and the data that can get sent back to it using the Input and Output fields defined in the guide properties.

For example, suppose there are a set of steps associated with entering client information. If there are different ways that your users interact with clients and many need to enter client information, you could group these client information steps into their own guide. Now, whenever a guide needs to enter client information, it just "embeds" the guide. This also means that all embedding guides use the same steps

When you create an Embedded Guide step, you enter the following properties:

Property	Description
General	The name of the Embedded Guide step. The name can contain only alphanumeric characters, underscores (_), spaces, and Unicode characters.
Guide	Select the guide that you want to embed within the current guide. The current guide is called the embedding guide. The guide that the embedding guide embeds is called the embedded guide.
Iteration Rule	Configure whether you want the embedded guide to run on a single object or on all objects. If you choose **Run on a single object**, the embedded guide runs on the specific object it receives as input. The embedded guide passes control to the step that follows in the embedding guide when execution completes. If you choose **Run on each object in list until** , you also choose an event that controls guide execution: You can also choose from the following objects to process at runtime: - **Field**: A single object or a list of objects is passed to the embedded guide (based on the selected Run choice). - **Query**: Available only with Run on a single object. Allows you to define a WHERE condition that selects the object passed to the embedded guide. - **List**: Available only with Run on each object in list until outcome. Allows you to define a WHERE condition that selects the objects passed to the embedded guide.
Input Fields	Displays the input fields available for the selected object. If the embedded guide receives one of these fields as input, the value of that field may be updated while the embedded guide executes. If so, the value is updated in the parent guide after this embedded guide completes execution.

The following image shows an embedding guide that contains an embedded guide:
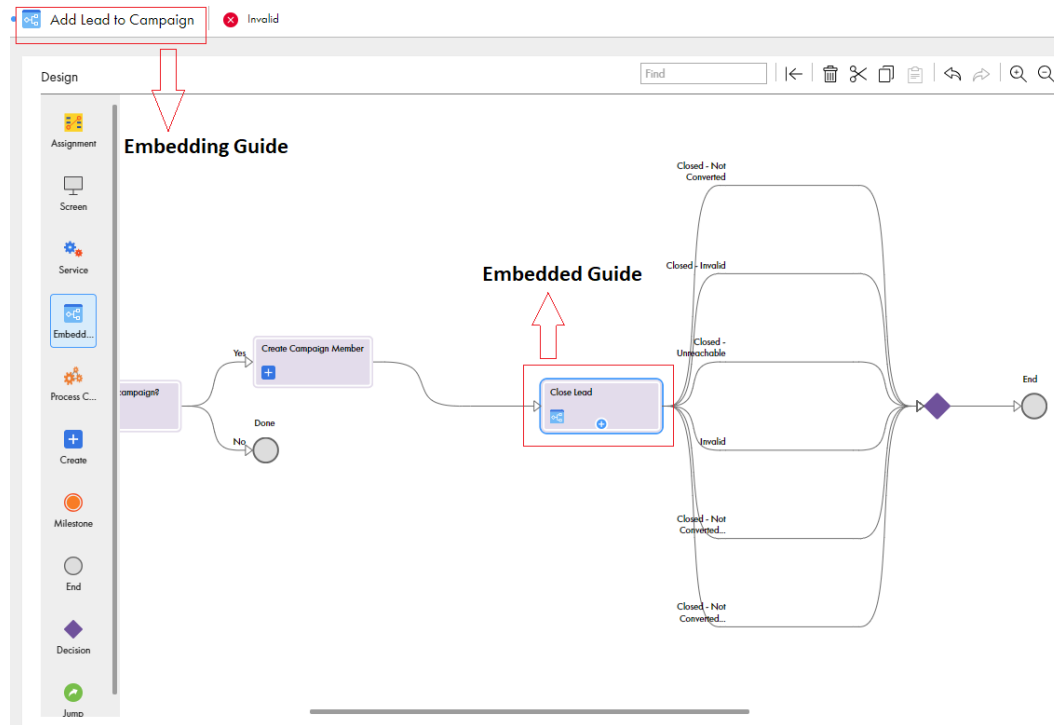
Service Step

Use a Service step to connect to an external service, a system service, or a process.

When you add a Service step, you need to configure the Service Type, Connection, and Action.

The following table describes the Service step properties:

Property	Description
General	The name of the Service step. The name can contain only alphanumeric characters, underscores (_), spaces, and Unicode characters.
Service Type	The connection, process, or system service you add to the process. Select from a list of existing tasks. **Note:** You must have an existing item to add to the guide. You cannot create an item when you use the Service step.
Connection	The connection to the service connector.
Action	The action contained within the connection.

The following image shows a Service step with the Salesforce connection selected:

Input Fields

The **Input Fields** section shows the names of input fields that are most often used when using this kind of step. For Service steps that create objects, the input fields shown are the fields that are most frequently needed when the object is created. (If you do not need a field and it is optional, you can delete it.) You can choose additional (optional) input fields from the list.

Use the delete icon to remove a field. This removes input fields that you do not want to pass to the Service step. Some fields are required and cannot be deleted.

End/Milestone Step

The End/Milestone step marks the end of a guide's execution or a milestone.

When you add an End step to a guide, you can configure the step to be a **Milestone** type step or an **End of Guide** type step. A Milestone ending indicates the end of one phase in a guide's activities.

Note: You can make the text that you enter more readable if you insert text and fields into a table. For more information, see *Inserting Fields in Tables*.

End of Guide Step

An End of Guide step marks the place where a guide finishes executing. Every branch in a guide concludes with an End step, unless it is a Jump Stepon page 180. The An End of Guide step specifies an outcome.

The following image shows a sample **End of Guide** step:

Set the following properties when you select the **End of Guide** option:

- **Ending Type**: Milestone

- **Show Screen**: Select this option for the user sees the milestone step. If you do not select it, the user does not see the milestone step. Instead, the action indicated in the On Done Button area immediately occurs after this step executes.

- **Outcome**: A list containing the outcomes you defined in the Start step. You can also add a new outcome.

- **Preprocessing**: Select an option from this list to configure the action occurs after the Milestone step completes.

   - **Refresh Current Object**: The page from which you launched the guide is refreshed so that the action performed by the guide is visible on the object's page.

   - **Go to Other Object**: When selected, you see a **Click To Select Field** list to the right. Select one of the items in it. Many are IDs associated with the current object. This is especially valuable for a guide whose primary purpose is to create a new object (for example, a lead). You might then have the End step specify that the user should be taken to the object after the guide finishes executing.

   - **Go to URL**: When selected, you see a **Click To Select Field** list to the right. Select a field that is of type URL.

Milestone Step

Milestones are especially useful when a user hands off work to someone else. (If the guide is handed off, the user who will be handing the guide off should let the next user know that something needs to be done. This may be built into your application's workflow rules.) When that user starts the guide, it starts at the step following the milestone. Milestones should also be used if the user normally stops executing the guide at a step to do something else.

In some cases, you may simply be using a milestone stop to record an event that occurred and the guide will automatically continue to the next step.

The following image shows a sample **Milestone** step:

After a guide hits a milestone, further activities do not need to occur. However, the guide will be terminated if no further actions occur within 14 days. Guides without a Milestone guides are terminated in seven days. Once action begins after a milestone is reached, the guide will be terminated if it does not either end or reach another milestone in seven days.

Set the following properties when you select the **Milestone** option:

- **Ending Type**: Milestone

- **Show Screen**: Select this option for the user sees the milestone step. If you do not select it, the user does not see the milestone step. Instead, the action indicated in the On Done Button area immediately occurs after this step executes.

- **Allow user to go back**: Select this option for the user to be able to go back to steps executed before the milestone. In either case, all steps that occurred within the guide display in the guide's history. However, if the user cannot go back, the user cannot select one of these previously executed steps.

- **Preprocessing**: Select an option from this list to configure the action occurs after the Milestone step completes.

  - **Refresh Current Object**: The page from which you launched the guide is refreshed so that the action performed by the guide is visible on the object's page.

  - **Go to Other Object**: When selected, you see a **Click To Select Field** list to the right. Select one of the items in it. Many are IDs associated with the current object. This is especially valuable for a guide whose primary purpose is to create a new object (for example, a lead). You might then have the End step specify that the user should be taken to the object after the guide finishes executing.

  - **Go to URL**: When selected, you see a **Click To Select Field** list to the right. Select a field that is of type URL.

  - **Refresh Current Object and Continue**: After the object upon which this guide is executing is refreshed, execution moves to the step that follows.

  - **Continue**: Notes that the milestone was reached and execution continues with the next step.

Setting the Appearance of a Guide

You can customize the appearance of a guide. You can select from built-in themes to determine what a guide will look like when a user runs the guide on a desktop browser. You can also customize the instructions that display when the guide runs.

Guide Appearance

Use the **Configure Guides** page to specify guide themes and manage guide images at an organizational level.

Use the tabs on the **Configure Guides** page to configure guide settings.

Themes

Select from a list of inbuilt themes to configure your guide. You can select one of the following inbuilt themes:

- Blueray
- Default
- Lightning
- Redmond
- Vader

The following image shows the **Configure Guides** page with the **Lightning** theme selected in the **Theme** tab:
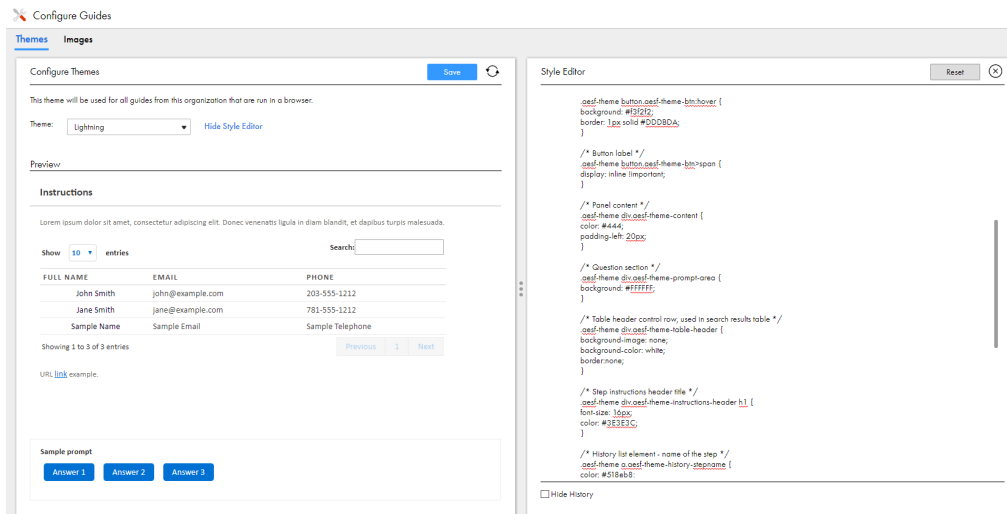
Click **Show Style Editor** to view and edit the CSS for a guide step, paragraph, or character in the theme. You can use any CSS property normally available for the element type.

For example, to change the color of the steps header text to blue, set the following CSS:

```
aesf-theme div.aesf-theme-instructions-header h1 {
   color:blue;
}
```

When you select, configure, and save a theme, it applies to all guides in your organization.

Images

Use the **Images** tab to view and edit icons that you want to use in guides.

The following image shows the **Configure Guides** page with the **Images** tab selected:
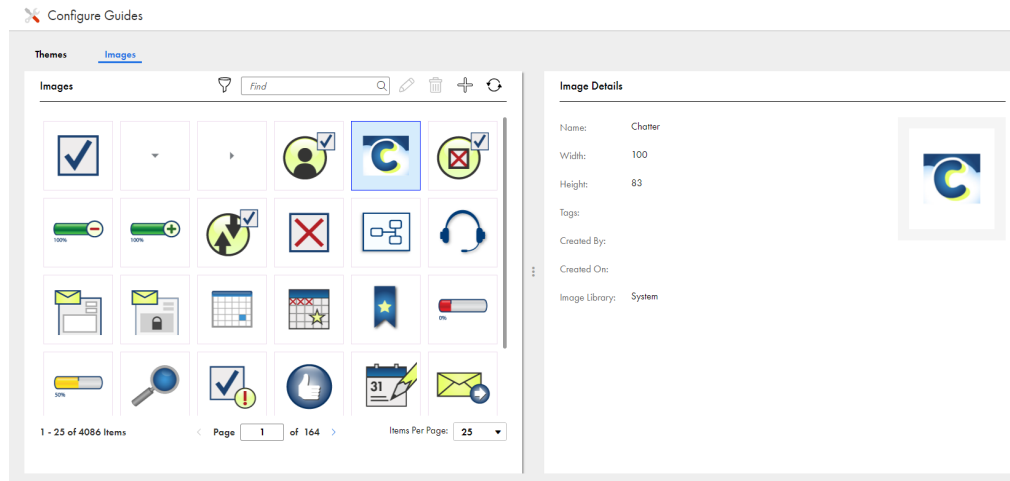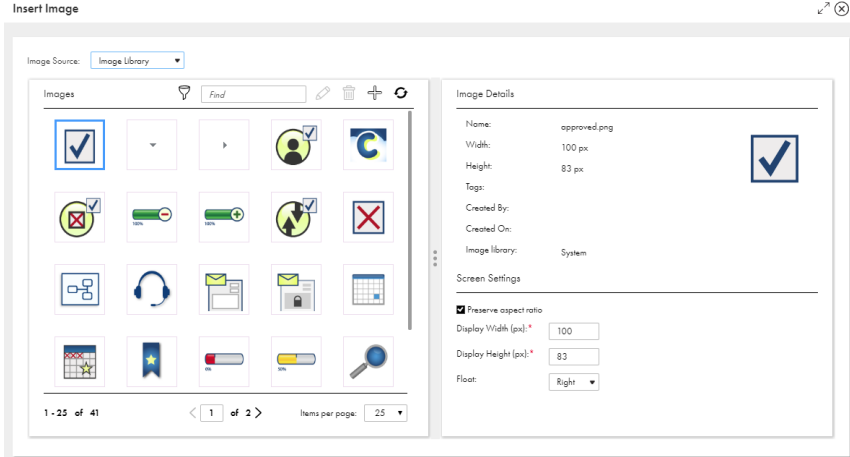
When you click **Add Image** at the top of the **Images** tab, the **Add Image** dialog box opens. You can upload `.gif`, `.jpg`, `.jpeg`, or `.png` images from your system to the Application Integration image library.

Guide screen Editor
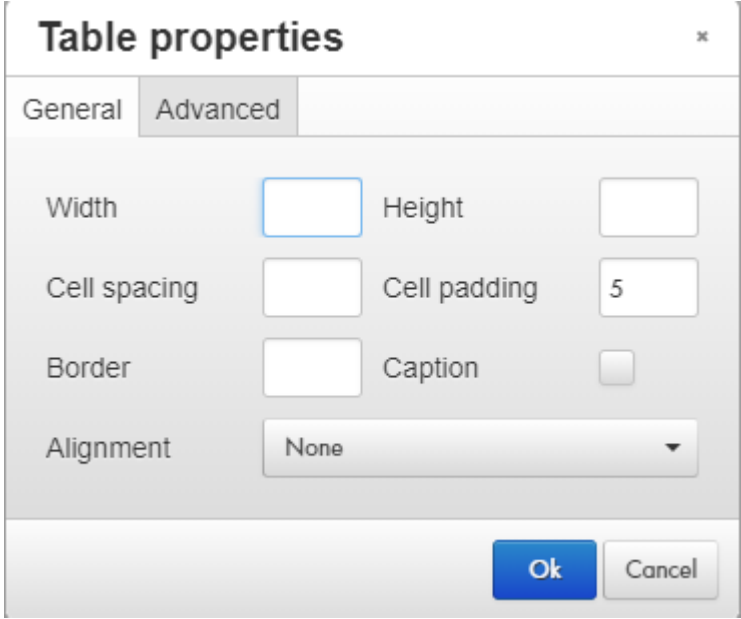
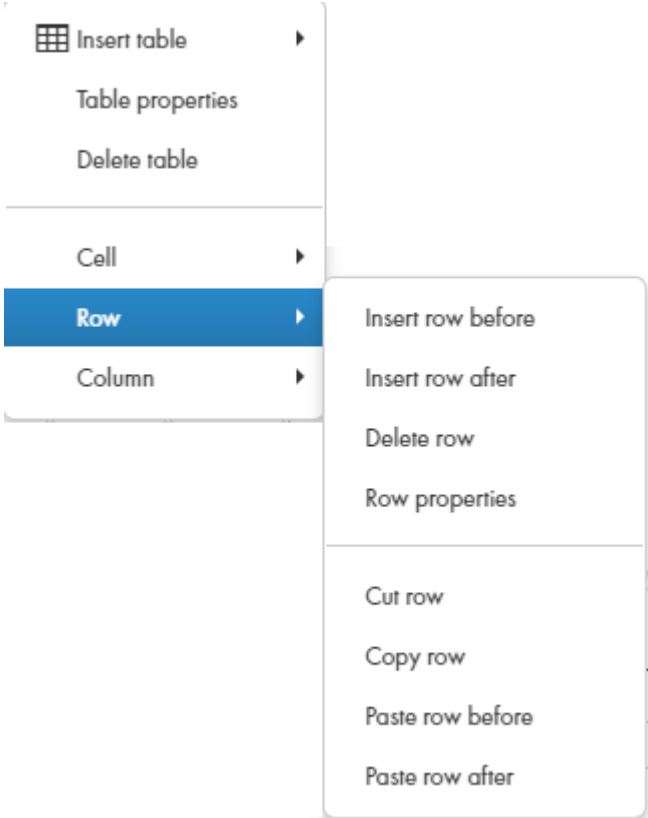Use the screen editor to insert and edit instructions and fields into a screen that displays in a guide. You can use the following tools in the screen editor:

Button	Description
B (Bold)	Sets selected text to **bold** or indicates that the text you are about to type will be bold. If you press this button when bold text is selected, the bolding is removed. (Keyboard: Ctrl+B)
I (Italic)	Sets selected text to *italic* or indicates that the text you are about to type will be italic. If you press this button when italic text is selected, the italic is removed. (Keyboard: Ctrl+I)
U (Underline)	Underlines selected text or indicates that the text you are about to type will be underlined. If you press this button when underlined text is selected, the underlining is removed. (Keyboard: Ctrl+U)
(Bullet list)	If the current paragraph does not have a bullet, adds a bullet and changes the paragraph's indent. If the paragraph has a bullet, this command removes the bullet and removes a level of indenting. When you press Enter within or at the end of a paragraph, the next paragraph also has a bullet. To end the bullet list, either type two returns or reselect this icon.
(Numbered list)	If the current paragraph does not have a number, adds a number and changes the paragraph's indent. If the paragraph has a number, this command removes the number and removes a level of indenting. If the preceding paragraph is part of a numbered list, selecting this command adds it to that list. When you press Enter within or at the end of a paragraph, the next paragraph also has a number. To end the numbered list, either type two returns or reselect this icon.
(Decrease indent)	If the current paragraph or selected paragraphs are indented, it removes a level of indenting.

Button	Description
(Increase indent)	Adds a level of indenting to the current paragraph or to selected paragraphs.
(Insert/ edit link)	Changes the selected text into a link. After you select this button, Guide Designer displays the following dialog: Insert Link ⊗ Link URL: [_____] Title: [_____] **Insert Link** Cancel The files in this dialog are: - **Link URL**: Where you type a Web address. - **Title**: Text that appears when a user moves the cursor over the link. (This text may not appear in all browsers.) If you press this button while your cursor is anywhere in a link, you can edit the link's information. If you forget to select text, the link will not display. However, it will be created. Correct this by pressing the **<>** button to locate the link. If you know HTML, you can edit the link directly. If you do not, just delete it. (An empty link's text is similar to "`<a href=""><a>`".)
(Remove link)	Removes a previously created link. Your cursor can be anywhere within the link. The text in the "Text to display" field remains after you select this button.

Button	Description
(Insert/ edit image)	If an image is not selected, press this button to insert a new image. To change an existing image or its properties, select the image before pressing this button. After pressing this button, Guide Designer displays the following dialog: The images you see in the top area were previously imported into Guide Designer using the **Settings** button located on the Design Home page. You can also upload an image for use in the current screen step. An image uploaded here is not available for use in other steps within this guide or in any other guide. You can also insert any image available on the web into the step. Do this by dragging it from the web site directly into the screen editor at the position at which you want to inset it. The fields within this dialog are: - *Search area*: Use the left text box to search for an image using text that you typed either in the *Title* or *Description* fields. Use the right text box to type a tag that was used to label an image. As you type a tag's text, you will see tags that match what you are typing. - **URL**: If the is accessible using a URL, enter it here. - **Title**: A title for the image. - **Description**: Text that describes the image. Notice the text that was added to this dialog. - **Width x Height**: Use these fields to change the width and height of an image. You can either enter the size in pixels or enter a percent of the original size. The values you type can be less than or more than what the image's original dimensions were. If you only type a width and you entered pixels, the size is changed proportionally. For example, if you change the display width to 235 in the previous example, the display height will be 93 or 94 depending upon your browser. If you entered a percent value, that percent is also applied to the height. - **Original W x H**: The images width and height as it is stored. - **Float**: Use the options within this picklist to indicate the image's position in relation to where you inserted it. Your choices are None (it is displayed in line), Left (the image floats to the left), and Right (the image floats to the right). - **Browse**: Press this button to display a dialog that lets you select an image using your computer's file explorer. After you select and confirm your choice, the image will appear here.
(Undo)	Eliminates the previously executed action. (Keyboard: Ctrl+Z)
(Redo)	If you have previously selected **Undo**, pressing this restores the change. (Keyboard: Ctrl+Y)
(Fullscreen)	Enlarges the size of the step so that it completely fills the area beneath the Guide Designer tabs. If you have already enlarged it, pressing this button restores the step to its original size. **Note**: You cannot insert fields into the step when the editor is in full screen mode.

| Button	Description
(Text color)	Pressing this button displays a pallette of colors that can be used when displaying text. If no text is selected, pressing this button and selecting a color changes the color of the text you type. If you move the cursor away from this text, the color is what exists at the cursor. Pressing this button and selecting a color while text is selected changes the color of the selected text.
(Background color)	Pressing this button displays a pallette of colors that can be used for coloring the background behind text. If no text is selected, pressing this button changes the background color of the text you type. If you move the cursor away from this text, the background color is what exists at the cursor. Pressing this button and selecting a color while text is selected changes the background color of the selected text.
(Find and replace)	Pressing this button tells Guide Designer to display a dialog into which you can type the text you wish to locate in the current screen. If you enter text in the Replace with field and press the Replace button, the selected text is replaced. If you select Replace all, all the text that can be found is replaced.
(Source code)	If text is being displayed, pressing this button tells Guide Designer to display the HTML that is being used to format the text in a different browser window, which is the HTML source editor. This button is not a toggle. After you finish making changes in the source editor, press its **OK** button.
(Show/hide more tools)	Adds a second row of screen editor buttons. If this second row is showing, pressing this button removes it.
(Font family)	Pressing this button displays a picklist of the fonts that can be used to display text. If no text is selected, pressing this button and selecting a font family changes the font of the text you type. If you move the cursor away from this text, the font will be what exists at the cursor. Pressing this button and selecting a font while text is selected changes the font of the selected text.
(Font size)	Pressing this button displays a picklist of the font sizes that can be used to display text. If no text is selected, pressing this button and selecting a size changes the size of the text you type. If you move the cursor away from this text, the font size is what exists at the cursor. Pressing this button and selecting a size while text is selected changes the size of the selected text.
(Table)	Pressing this button tells Guide Designer to display a picklist of table commands. Select Insert Table to display a grid of boxes. Use your mouse to select the number of rows and columns that will appear within the table. If the cursor is within a table, you can select other commands in this picklist.

Button	Description
table properties	Selecting Table Properties within the picklist displays a dialog box with two tabs. **Table properties** General | Advanced Width [ ] Height [ ] Cell spacing [ ] Cell padding [5] Border [ ] Caption [ ] Alignment [None ▾] [Ok] [Cancel] Use this dialog to display the height and width of the table. (You should probably set the width as a percent -- use the percent symbol.) The height is seldom set. Cell spacing sets the space between table cell. Cell padding is the amount of space between a cell's contents and the cells edge. If you want a border, enter a number to indicate its width. If you want a caption to a table (this is not usual given the small size of the step's canvas), select this button. A small area above the table will appear. Finally, use the Alignment picklist to select None, Left, Center, or Right. None and Center are the most commonly used. The Advanced tab lets you set the table's style, and border and background colors. Select the **Delete Table** command to remove the table from the canvas.
cell properties	If you select Cell from the Table picklist, a second picklist displays. Use this picklist to define cell properties, merge two cells together, or split cells that were merged. When defining cell properties, you can specify the cells width and height (normally, you wouldn't select a height), its type (Cell or Header), the Scope (None, Row, Column, Row group, or Column group) of what is changed, and the horizontal and vertical alignment of a cell's contents.

Button	Description
row properties	If you select Row from the Table picklist, a second picklist displays. ![Table menu with Row submenu] ⊞ Insert table ▶ Table properties Delete table Cell ▶ Row ▶ Insert row before Column ▶ Insert row after Delete row Row properties Cut row Copy row Paste row before Paste row after Row properties are similar to cell properties. Other commands in this picklist are self-explanatory
$\mathcal{T}_x$ (Clear formatting)	If you've changed the way in which text displays, pressing this button tells the screen editor to remove your changes.
(Align left)	Aligns currently selected paragraphs or the paragraph containing the cursor to the left margin.
(Align center)	Centers currently selected paragraphs or the paragraph containing the cursor within the left and right margins.
(Align right)	Aligns current selected paragraphs or the paragraph containing the cursor to the right margin.
(Justify)	Adds spaces between words to lines in the currently selected paragraphs or the paragraph containing the cursor so that the text is justified; that is, all text is aligned to the left and right margins.
66 (Blockquote)	Changes the indent of selected paragraphs or the paragraph containing the cursor. If the paragraphs are already within a blockquote, this restores the original margin.

Validating and Saving a Guide

When you create a Guide, you can use the Validation panel to verify if there are validation errors and view the error details.

1. Click the **Validation** icon in the toolbar as shown in the following image:
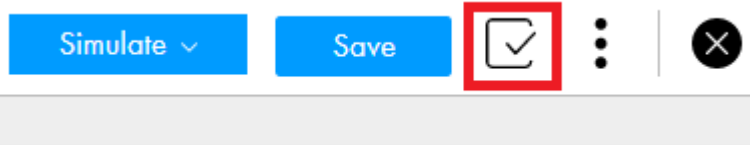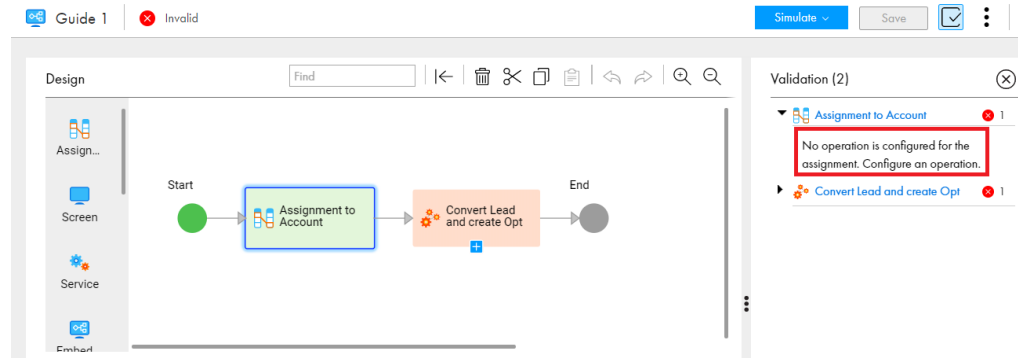

   The **Validation** panel appears on the right pane. In the **Validation** panel, errors are grouped based on the tabs where they occur.

2. Click an arrow corresponding to a tab name as shown in the following image:
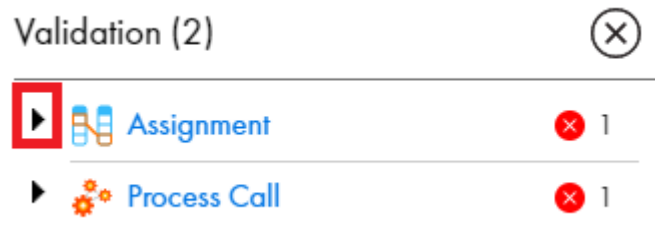
   A list of errors that occur in the tab is displayed.

3. Click an error row as shown in the following image:

   As and when you correct errors, the error list in the **Validation** panel is refreshed.

4. After you validate a guide, you can save the guide.

   Click **Save** in the toolbar as shown in the following image:

Guide Simulation

Use the **Simulate** option in Guide Designer to see how a guide appears when you run a guide.

You can simulate a guide in Guide Designer from the beginning or from any step.

The following image shows the **Simulate** option:
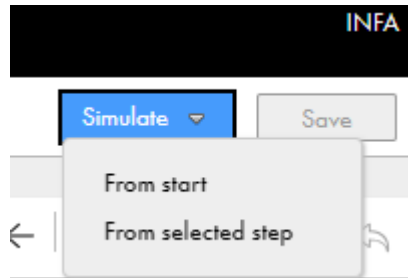
The type of simulation you see depends on whether you configured Browser or Mobile App display type in **Start** > **Screen** > **Intended Display**.

The following image shows guide simulation with the display configured as Browser:
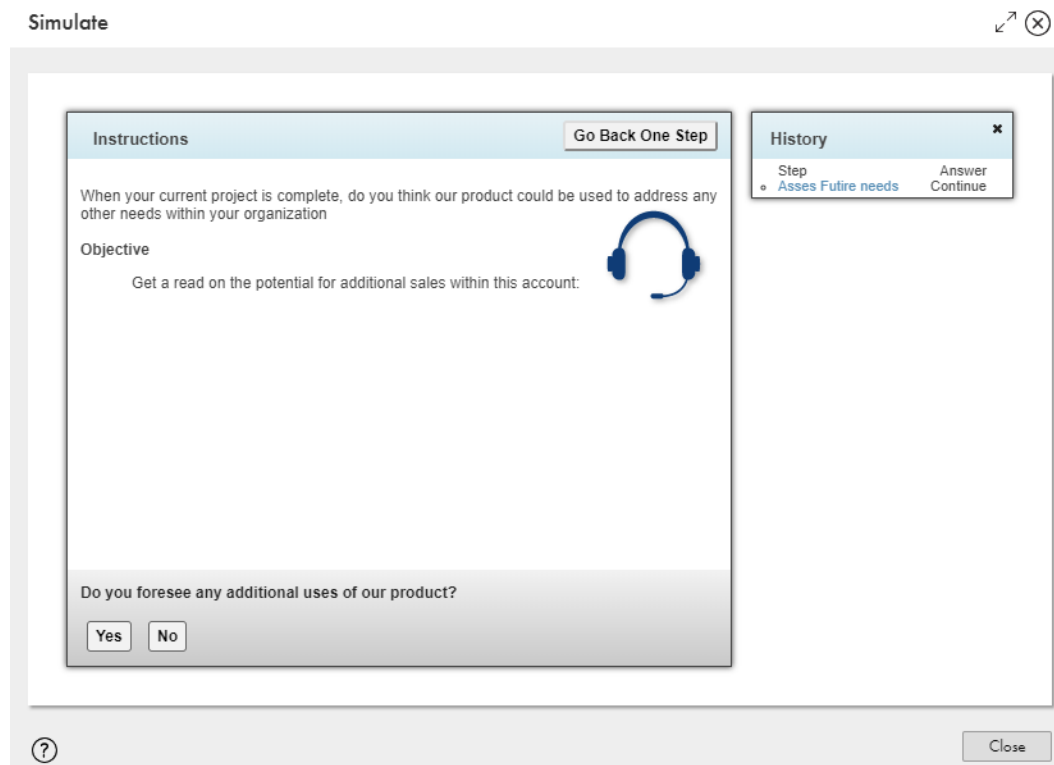
The following image shows guide simulation with the display configured as Mobile App:

Simulate

Back   ⚙ History

When your current project is complete, do you think our product could be used to address any other needs within your organization

Objective

Get a read on the potential for additional sales within this account:

Do you foresee any additional uses of our product?

Yes   No

?   Close
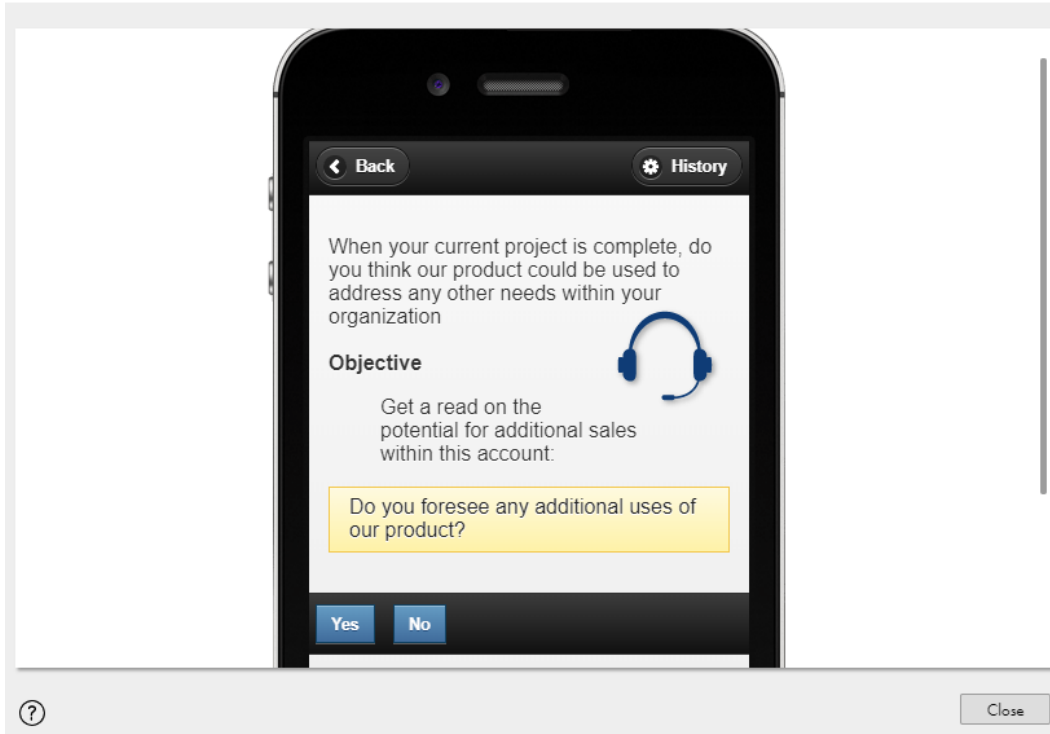
CHAPTER 5

Designing Process Objects

To design a business process, you often need to retrieve sets of structured data. For example, instead of creating separate items for each kind of demographic data, you could have a group that contains the name, address, and phone number fields. These groups are called as process objects. Process objects group and structure data for use in processes. The named process objects also help you to access returned information more efficiently. With process objects, you can create common objects that can be reused in multiple processes.

You can define process objects in two ways:

1. As a standalone process object for use in your applications.
2. Within a service connector to handle data returned from a service.

Process objects generated in a service connector are available only within the service connector. Standalone process objects are available in any service connector. Both are used similarly in processes.

Using the Process Object editor, you can define standalone process objects. They are then useable in the same way as built-in data types such as integer, text, or phone. In the Service Connector editor, you can generate process objects based on the data available from the service and the actions, bindings, and other properties you specify there.

Process Objects Creation Options

You can create process objects in the following ways:

Create process objects manually

You can create simple process objects manually by adding fields and configuring the field properties.

Create process objects from WSDL and XSD files

You can easily and simultaneously create multiple large and hierarchical process objects by importing one or more WSDL and XSD files.

Create process objects from Swagger files

You can easily and simultaneously create multiple large and hierarchical process objects by importing one or more Swagger JSON or YAML files.

Create process objects from OpenAPI 3.0 files

You can easily and simultaneously create multiple large and hierarchical process objects by importing one or more OpenAPI 3.0 JSON or YAML files.

Create process objects from JSON files

You can easily create multiple large and hierarchical process objects that cover data types used in JSON payloads by importing one or more JSON files.

Creating Process Objects Manually

You can create simple process objects by manually adding fields and configuring the field properties.

1. In Application Integration, click **New** > **Process Object** > **Process Object** > **Create**.

   The process object editor appears.

   The following image shows the process object editor:

2. Specify a name for the process object.

   The name must start with a letter, and can contain only alphanumeric characters, multibyte characters, underscore (_), and hyphens (-). The name must not exceed 120 characters.

3. Click **Browse** to specify a location to save the process object.

4. Optionally, enter a description for the process object.

5. To add a field to the process object, perform the following steps:

   a. Enter a name for the field.

      The name must start with a letter, and can contain only alphanumeric characters and hyphens (-).

   b. Optionally, enter a label for the field.

   c. Optionally, from the **Type** list, select an appropriate data type for the field. Default is **Text**.

      To reference another process object, select the type as **Reference**.

d. Click **Add Field** to add the field to the process object.

- If you selected the type as **Object List** or **Reference**, browse and select the process object to be referenced. You can also enter the process object name in the **Find** box to search for a process object.

  Note: If you change the name of a referenced process object, you must also save the parent process object. For example, you can make a minor change to the description of the parent process object and click **Save**.

  You can click **View Referenced Process Object** to view the referenced process object from the process object editor itself. The referenced process object opens on a new tab on the left navigation bar in a read-only mode.

- If you selected the type as **Picklist** or **Multi-Select Picklist**, enter one or more options separated by commas.

e. To set the type as a list of simple types for the text, integer, number, check box, date, date time, or time data types, select the row that contains the type, and then select the **This is a list** check box.

   Note: The list of simple types feature in process object is in technical preview. Technical preview functionality is supported but is unwarranted and is not production-ready. Informatica recommends that you use in non-production environments only. Informatica intends to include the preview functionality in an upcoming GA release for production use, but might choose not to in accordance with changing market or technical circumstances. For more information, contact Informatica Global Customer Support.

f. To configure additional fields for the text, date, date time, time, number, or integer data types, expand the **Name** field and enter the additional fields as follows:

- Length and format of the text for the text data type

- Date range for the date data type

- Date time range for the date time data type

- Time range for the time data type

- Minimum and maximum values for the number data type

- Minimum and maximum values for the integer data type

g. To set the field as a required field, select **Required**.

   A required field must be included in the request payload. It can contain a value, be empty, or be null.

h. If you do not want the field to accept null values, clear the **Nullable** check box. By default, the check box is selected.

6. Configure the required fields for the process object.

   The following image shows an Employee process object with three fields namely, ID, Name, and Address:

To order a field, use the arrow icons against the row that contains the field to move up or down.

To delete a field, hover over the row that contains the field and click the **Delete** icon.

7. To edit the field values, click anywhere on the row that contains the field.

All the fields in the row become editable.

8. Click **Save** to save the process object.

Rules and Guidelines for using Process Objects in a Process

Consider the following rules and guidelines when you use process objects in a process:

- If you use a process object with required fields as the input for an Application Integration process, you must ensure that the required fields are present in the request payload. Otherwise, when you run the process, you receive an error for the missing fields.

  For example, in a process object, you have set the ID field as required, but the field is not present in the request payload as shown in the following sample:

  ```
  <root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <Employee>
          <Name xsi:nil="true"></Name>
          <Address xsi:nil="true"></Address>
      </Employee>
  </root>
  ```

  After you run the process, you will receive the following error:

  ```
  <root>
      <code>400</code>
      <details>
          <missing>[ID]</missing>
      </details>
      <message>The parameter validation failed</message>
  </root>
  ```

- If you set a process object field as Nullable, it can contain a valid value or be null. Non-nullable fields cannot be set as null.

  In the JSON and XML formats, null values in a process object field that is used as the input for an Application Integration process are handled differently.

In the JSON format, you can specify null values directly as a field value.

For example, `{"Name":null}`

XML does not support null values. Therefore, in the XML format, if a field is set as Nullable and you want to pass null values, you must set the `xsi:nil` attribute as `"true"` in the field element and include `<xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">` in the root element as shown in the following sample:

```
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Employee>
        <ID>1234</ID>
        <Name xsi:nil="true"></Name>
        <Address xsi:nil="true"></Address>
    </Employee>
</root>
```

Creating Process Objects from WSDL and XSD Files

You can easily and simultaneously create multiple large and hierarchical process objects by importing one or more WSDL and XSD files.

Use the **New Process Objects Creation from WSDL and/or XSD Files** wizard to create process objects from WSDL and XSD files. After you specify the source, Application Integration parses the source, loads the resources, and generates XSD schemas. Based on the XSD schemas, Application Integration generates the process objects. You can specify import preferences such as the location for saving the process objects and optionally specify the tags to be used for the process objects.

In case errors occur during the process object creation, the wizard displays the errors under the **Related Logs** tab of the corresponding step. You can review the log and take corrective actions. You can also download the log. When you download the log at a particular step, the log also shows cumulative logging information for all the previous steps.
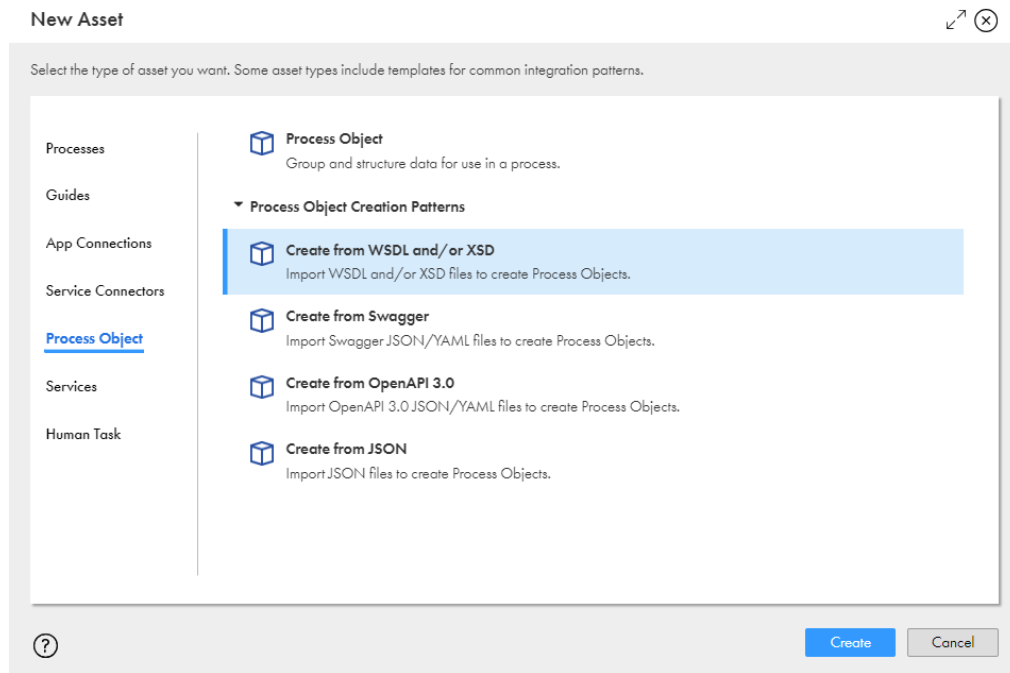
The following video shows you how to create process objects by importing a WSDL file:

1. In Application Integration, click **New**.

    The **New Asset** dialog box appears.

    The following image shows the **New Asset** dialog box:

2. Click **Process Object** on the left pane.

3. On the right pane, expand the **Process Object Creation Patterns** list, click **Create from WSDL and/or XSD**, and then click **Create**.

    The **New Process Objects Creation from WSDL and/or XSD Files** dialog box appears.

    The following image shows the **New Process Objects Creation from WSDL and/or XSD Files** dialog box:

4. In the **Source** field, select one of the following values:

Option	Description
File	Select **File** to specify a single WSDL or XSD source file with type definitions from your local system. You can also specify HTTP resources. Click **Choose File** to select the WSDL or XSD source file. The maximum allowed file size for a single file is 5 MB. Default is **File**.
Zip (multiple files)	Select **Zip** to specify multiple WSDL or XSD source files with type definitions in the form of a zip file from your local system. Click **Choose File** to select the zip file. The maximum allowed file size for a zip file is 3 MB.
URL	Select **URL** to specify a URL that contains a WSDL or XSD source file with type definitions. The **URL** and **Use Authentication** fields appear. In the **URL** field, enter the source URL. Select the **Use Authentication** option to specify the user name and password to access the source URL.

5. Click **Next**.

   Application Integration parses the specified source file or URL, and displays the resource details on the **Load Resources** tab.

   The following image shows the **Load Resources** tab with the resource details:

New Process Objects Creation from WSDL and/or XSD Files

① Select Source ② Load Resources ③ Generate Schemas
④ Generate Objects ⑤ Import Preferences ⑥ Import Objects

Files Found (3)   Related Logs

File Name	File Path	Type	Status	Comments
http://schemas.x...		XSD	✓ Valid	File type supported.
ndfdXML.wsdl	https://graphical....	WSDL	✓ Valid	File type supported.
http://schemas.x...		XSD	✓ Valid	File type supported.

< Back   Next >   Cancel

6. Review the resource details.

    The following table describes the columns that the **Load Resources** tab displays:

Column Name	Description
File Name	The file name of the resource.
File Path	The URL or relative location of each resource within the source.
Type	The file type of the resource. Displays one of the following values: - **XSD** - **WSDL**
Status	The status of the resource. Displays one of the following values: - **Valid**. Indicates that the file type is supported and that the file content is valid. - **Invalid**. Indicates that either the file type is not supported or the file content is not valid. For example, for a `.txt` file, the status shows up as **Invalid** because the file type is not supported.
Comments	Indicates whether the file type is supported or not.

If there are parsing errors, Application Integration displays the error details on the **Related Logs** tab. Fix the errors and try again.

7. Click **Next**.

    Application Integration converts the resources into XSD schemas based on which the process objects will be created. It displays the schema details on the **Generate Schemas** tab.

The following image shows the **Generate Schemas** tab with the schema details:

8. Review the schema details.

The following table describes the columns that the **Generate Schemas** tab displays:

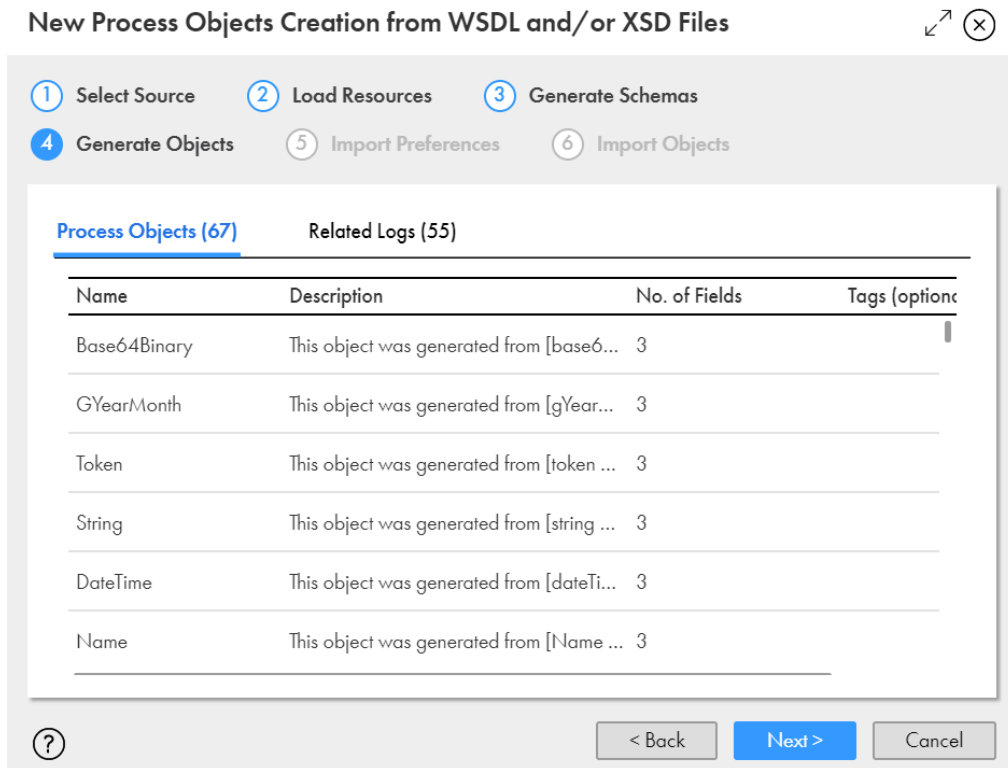| Column Name	Description
Schema	The name of the generated XSD schema file.
Source	The source from which the XSD schema file was generated.
Complex Types	The number of complex types that will be created for each XSD schema.

Column Name	Description
Prefixes	The prefix to be used for the XSD schema.
	You can use a prefix to easily identify process objects that are created from a schema. For example, you might want to add the prefix **Salesforce** for all process objects that are created from a Salesforce schema.
	To specify a prefix for a schema, double-click under the **Prefixes** column against the row that contains the schema, and enter a prefix. All the process objects that are created from the schema will use the same prefix that you specify in this field.
Tags	The tags to be used for the XSD schema.
	You can use a tag to group related schemas together. For example, you might want to group all schemas containing financial resources under the tag **Finance**.
	To specify a tag for a schema, double-click under the **Tags** column against the row that contains the schema, and enter a tag. All the process objects that are created from the schema will use the same prefix that you specify in this field.

If there are schema generation errors, Application Integration displays the error details on the **Related Logs** tab. Fix the errors and try again.

9. Click **Next**.

   Application Integration applies the prefixes and tags that you specified for the XSD schemas, and generates the process objects based on the XSD schemas. It displays the process object details on the **Generate Objects** tab.

   The following image shows the **Generate Objects** tab with the process object details:

10. Review the process object details.

The following table describes the columns that the **Generate Objects** tab displays:

Column Name	Description
Name	The name of the process object that was generated from the schema.
Description	The schema from which the process object was generated. You can edit the description.
No. of Fields	The number of complex types that the process object contains.
Tags	The tags to be used for the process object. By default, the process object tags are derived from the tags that you specified for the XSD schema. You can edit the tags if you want to specify different tags at the process object level. To edit the tags for a process object, double-click under the **Tags** column against the row that contains the process object, and update the tags.

11. Click **Next**.

The **Import Preferences** tab appears displaying the number of process objects that will be created.

The following image shows the **Import Preferences** tab:

12. Optionally, click **Select** to specify a location where you want to save the process objects.

Default is the location that was selected when you opened the **New Process Objects Creation from WSDL and/or XSD Files** wizard.

13. Optionally, specify additional tags for the process objects as comma-separated values.

14. Click **Start Import**.

Application Integration starts importing the process objects and displays the import status.

The following image shows the import status:

15. Perform one of the following actions:

- To dismiss the dialog box and get notified when the process objects import is complete, click **Dismiss**.

- To view and fix import errors, and retry the import, stay on the dialog box.

After importing the process objects, Application Integration displays a status message indicating if the import was successful or failed.

If the import failed, check the **Related Logs** tab, fix the errors, and try importing the process objects again. You can also click the **Download** icon to download the entire log.

The following image shows a sample log stating that one process object was not imported due to a 401 error for the POST operation:

New Process Objects Creation from WSDL and/or XSD Files

① Select Source   ② Load Resources   ③ Generate Schemas
④ Generate Objects   ⑤ Import Preferences   ⑥ Import Objects

Review   **Related Logs (2)**

2020-01-12T10:41:33.368Z [ERROR] : Provider operation failed for operation POST, reason - 401.

2020-01-12T10:41:33.368Z [INFO] : Process Objects importing has been finished. Successfully imported 66 Process objects. Failed to import 1 objects. Objects use 0 tags

< Back   Cancel

16. Click **Done**.

17. Navigate to the location that you specified for the process objects creation and review the process objects.

Creating Process Objects from Swagger Files

You can easily and simultaneously create multiple large and hierarchical process objects by importing one or more Swagger JSON or YAML files.

Use the **New Process Objects Creation from Swagger Files** wizard to create process objects from Swagger files. After you specify the source, Application Integration parses the source, loads the resources, and generates XSD schemas. Based on the XSD schemas, Application Integration generates the process objects. You can specify import preferences such as the location for saving the process objects and optionally specify the tags to be used for the process objects.

In case errors occur during the process object creation, the wizard displays the errors under the **Related Logs** tab of the corresponding step. You can review the log and take corrective actions. You can also download the

log. When you download the log at a particular step, the log also shows cumulative logging information for all the previous steps.

1. In Application Integration, click **New**.

   The **New Asset** dialog box appears.

   The following image shows the **New Asset** dialog box:

2. Click **Process Object** on the left pane.

3. On the right pane, expand the **Process Object Creation Patterns** list, click **Create from Swagger**, and then click **Create**.

   The **New Process Objects Creation from Swagger Files** dialog box appears.

   The following image shows the **New Process Objects Creation from Swagger Files** dialog box:

4. In the **Source** field, select one of the following values:

Option	Description
File	Select **File** to specify a single Swagger source file from your local system. Click **Choose File** to select the Swagger source file. The maximum allowed file size for a single file is 5 MB. Default is **File**.
Zip (multiple files)	Select **Zip** to specify multiple Swagger source files in the form of a zip file from your local system. Click **Choose File** to select the zip file. The maximum allowed file size for a zip file is 3 MB.
URL	Select **URL** to specify a URL that contains a Swagger source file. The **URL** and **Use Authentication** fields appear. In the **URL** field, enter the source URL. Select the **Use Authentication** option to specify the user name and password to access the source URL.

5. Click **Next**.

Application Integration parses the specified source file or URL and displays the resource details on the **Load Resources** tab.

The following image shows the **Load Resources** tab with the resource details:

6. Review the resource details.

   The following table describes the columns that the **Load Resources** tab displays:

Column Name	Description
File Name	The file name of the resource.
File Path	The URL or relative location of each resource within the source.
Type	The file type of the resource. Displays the value as **SWAGGER**.
Status	The status of the resource. Displays one of the following values: - **Valid**. Indicates that the file type is supported and that the file content is valid. - **Invalid**. Indicates that either the file type is not supported or the file content is not valid. For example, for a .txt file, the status shows up as **Invalid** because the file type is not supported.
Comments	Indicates whether the file type is supported or not.

If there are parsing errors, Application Integration displays the error details on the **Related Logs** tab. Fix the errors and try again.

7. Click **Next**.

   Application Integration converts the resources into XSD schemas based on which the process objects will be created. It displays the schema details on the **Generate Schemas** tab.

   The following image shows the **Generate Schemas** tab with the schema details:

8. Review the schema details.

The following table describes the columns that the **Generate Schemas** tab displays:

Column Name	Description
Schema	The name of the generated XSD schema file.
Source	The source from which the XSD schema file was generated.
Complex Types	The number of complex types that will be created for each XSD schema.
Prefixes	The prefix to be used for the XSD schema.
	You can use a prefix to easily identify process objects that are created from a schema. For example, you might want to add the prefix **Salesforce** for all process objects that are created from a Salesforce schema.
	To specify a prefix for a schema, double-click under the **Prefixes** column against the row that contains the schema, and enter a prefix. All the process objects that are created from the schema will use the same prefix that you specify in this field.
Tags	The tags to be used for the XSD schema.
	You can use a tag to group related schemas together. For example, you might want to group all schemas containing financial resources under the tag **Finance**.
	To specify a tag for a schema, double-click under the **Tags** column against the row that contains the schema, and enter a tag. All the process objects that are created from the schema will use the same prefix that you specify in this field.

If there are schema generation errors, Application Integration displays the error details on the **Related Logs** tab. Fix the errors and try again.

9. Click **Next**.

   Application Integration applies the prefixes and tags that you specified for the XSD schemas, and generates the process objects based on the XSD schemas. It displays the process object details on the **Generate Objects** tab.

   The following image shows the **Generate Objects** tab with the process object details:

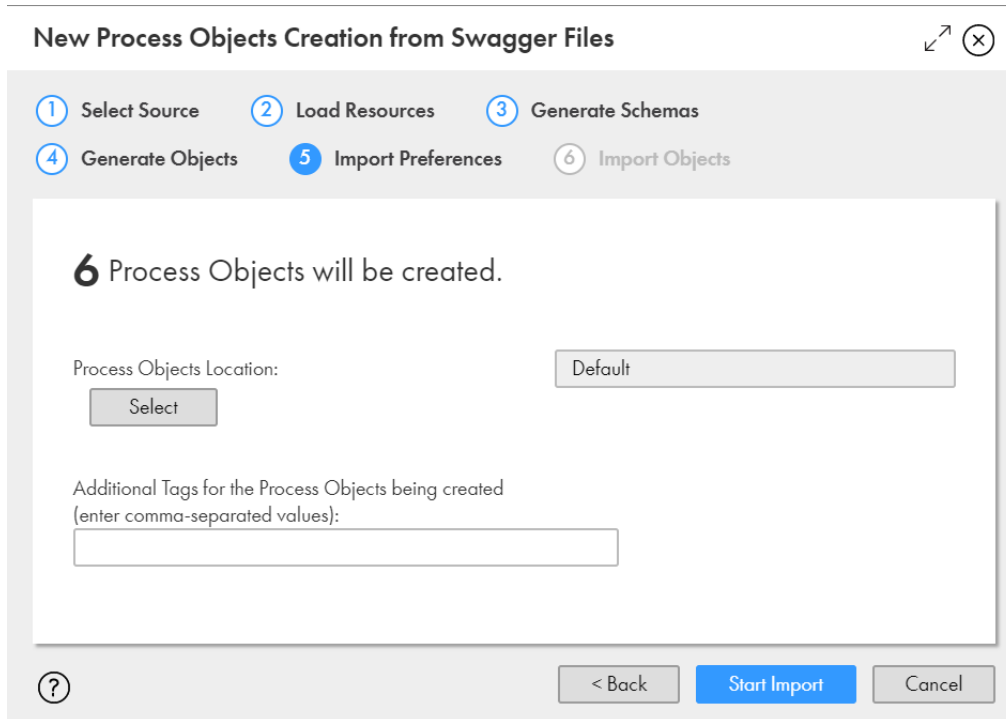10. Review the process object details.

    The following table describes the columns that the **Generate Objects** tab displays:

Column Name	Description
Name	The name of the process object that was generated from the schema.
Description	The schema from which the process object was generated. You can edit the description.
No. of Fields	The number of complex types that the process object contains.
Tags	The tags to be used for the process object. By default, the process object tags are derived from the tags that you specified for the XSD schema. You can edit the tags if you want to specify different tags at the process object level. To edit the tags for a process object, double-click under the **Tags** column against the row that contains the process object, and update the tags.

11. Click **Next**.

    The **Import Preferences** tab appears displaying the number of process objects that will be created.

    The following image shows the **Import Preferences** tab:

New Process Objects Creation from Swagger Files

1 Select Source 2 Load Resources 3 Generate Schemas
4 Generate Objects 5 Import Preferences 6 Import Objects

6 Process Objects will be created.

Process Objects Location: Default
    Select

Additional Tags for the Process Objects being created
(enter comma-separated values):

[   ]

< Back   Start Import   Cancel

12. Optionally, click **Select** to specify a location where you want to save the process objects.

    Default is the location that was selected when you opened the **New Process Objects Creation from Swagger Files** wizard.

13. Optionally, specify additional tags for the process objects as comma-separated values.

14. Click **Start Import**.

    Application Integration starts importing the process objects and displays the import status.

    The following image shows the import status:

15. Perform one of the following actions:

    - To dismiss the dialog box and get notified when the process objects import is complete, click **Dismiss**.

    - To view and fix import errors, and retry the import, stay on the dialog box.

    After importing the process objects, Application Integration displays a status message indicating if the import was successful or failed.

    If the import failed, check the **Related Logs** tab, fix the errors, and try importing the process objects again. You can also click the **Download** icon to download the entire log.

    The following image shows a sample log stating that six process objects were not imported because they already exist in the specified location:
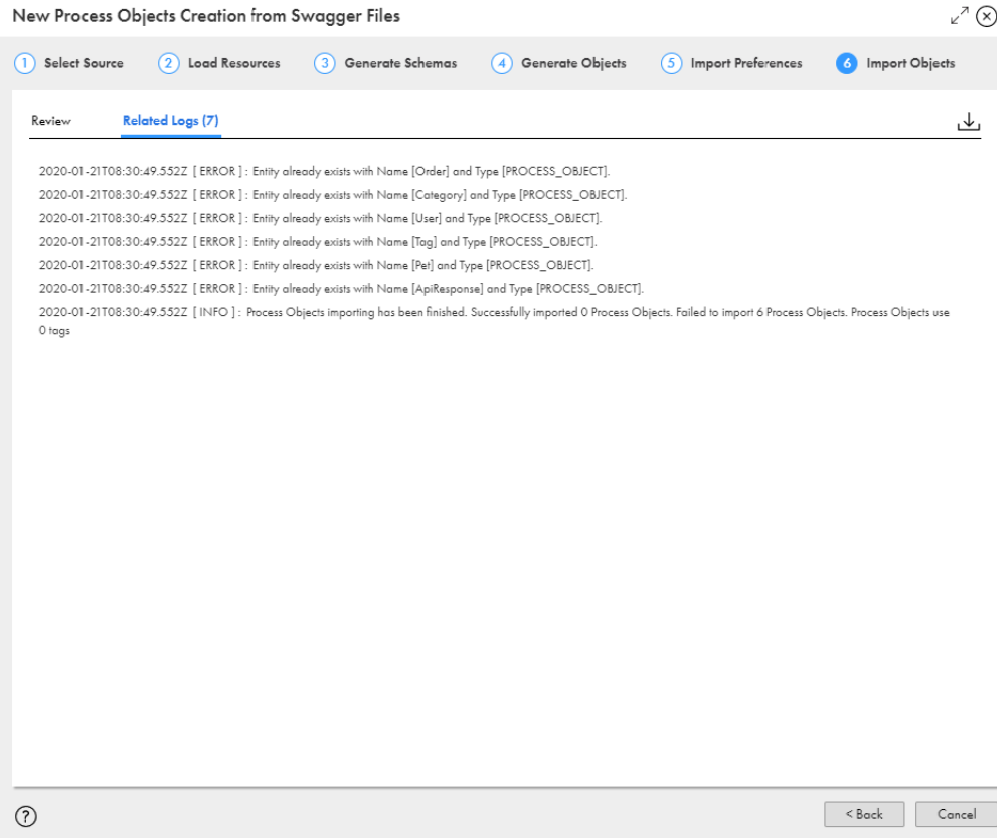
16. Click **Done**.

17. Navigate to the location that you specified for the process objects creation and review the process objects.

Creating Process Objects from OpenAPI 3.0 Files

You can easily and simultaneously create multiple large and hierarchical process objects by importing one or more OpenAPI 3.0 JSON or YAML files.

Use the **New Process Objects Creation from OpenAPI 3.0 Files** wizard to create process objects from OpenAPI 3.0 files. After you specify the source, Application Integration parses the source, loads the resources, and generates XSD schemas. Based on the XSD schemas, Application Integration generates the process objects. You can specify import preferences such as the location for saving the process objects and optionally specify the tags to be used for the process objects.

In case errors occur during the process object creation, the wizard displays the errors under the **Related Logs** tab of the corresponding step. You can review the log and take corrective actions. You can also download the

log. When you download the log at a particular step, the log also shows cumulative logging information for all the previous steps.

1. In Application Integration, click **New**.

   The **New Asset** dialog box appears.

   The following image shows the **New Asset** dialog box:

2. Click **Process Object** on the left pane.

3. On the right pane, expand the **Process Object Creation Patterns** list, click **Create from OpenAPI 3.0**, and then click **Create**.

   The **New Process Objects Creation from OpenAPI 3.0 Files** dialog box appears.

   The following image shows the **New Process Objects Creation from OpenAPI 3.0 Files** dialog box:

4. In the **Source** field, select one of the following values:

Option	Description
File	Select **File** to specify a single OpenAPI 3.0 source file with type definitions from your local system. You can also specify HTTP resources. Click **Choose File** to select the OpenAPI 3.0 source file. The maximum allowed file size for a single file is 5 MB. Default is **File**.
Zip (multiple files)	Select **Zip** to specify multiple OpenAPI 3.0 source files with type definitions in the form of a zip file from your local system. Click **Choose File** to select the zip file. The maximum allowed file size for a zip file is 3 MB.
URL	Select **URL** to specify a URL that contains an OpenAPI 3.0 source file with type definitions. The **URL** and **Use Authentication** fields appear. In the **URL** field, enter the source URL. Select the **Use Authentication** option to specify the user name and password to access the source URL.

5. Click **Next**.

Application Integration parses the specified source file or URL, and displays the resource details on the **Load Resources** tab.

The following image shows the **Load Resources** tab with the resource details:

6. Review the resource details.

The following table describes the columns that the **Load Resources** tab displays:

Column Name	Description
File Name	The file name of the resource.
File Path	The URL or relative location of each resource within the source.
Type	The file type of the resource. Displays the value as **OPENAPI**.
Status	The status of the resource. Displays one of the following values: - **Valid**. Indicates that the file type is supported and that the file content is valid. - **Invalid**. Indicates that either the file type is not supported or the file content is not valid. For example, for a .txt file, the status shows up as **Invalid** because the file type is not supported.
Comments	Indicates whether the file type is supported or not.

If there are parsing errors, Application Integration displays the error details on the **Related Logs** tab. Fix the errors and try again.

7. Click **Next**.

Application Integration converts the resources into XSD schemas based on which the process objects will be created. It displays the schema details on the **Generate Schemas** tab.

The following image shows the **Generate Schemas** tab with the schema details:

8. Review the schema details.

The following table describes the columns that the **Generate Schemas** tab displays:

Column Name	Description
Schema	The name of the generated XSD schema file.
Source	The source from which the XSD schema file was generated.
Complex Types	The number of complex types that will be created for each XSD schema.

Column Name	Description
Prefixes	The prefix to be used for the XSD schema.
	You can use a prefix to easily identify process objects that are created from a schema. For example, you might want to add the prefix **Salesforce** for all process objects that are created from a Salesforce schema.
	To specify a prefix for a schema, double-click under the **Prefixes** column against the row that contains the schema, and enter a prefix. All the process objects that are created from the schema will use the same prefix that you specify in this field.
Tags	The tags to be used for the XSD schema.
	You can use a tag to group related schemas together. For example, you might want to group all schemas containing financial resources under the tag **Finance**.
	To specify a tag for a schema, double-click under the **Tags** column against the row that contains the schema, and enter a tag. All the process objects that are created from the schema will use the same prefix that you specify in this field.

If there are schema generation errors, Application Integration displays the error details on the **Related Logs** tab. Fix the errors and try again.

9. Click **Next**.

Application Integration applies the prefixes and tags that you specified for the XSD schemas, and generates the process objects based on the XSD schemas. It displays the process object details on the **Generate Objects** tab.

The following image shows the **Generate Objects** tab with the process object details:

10. Review the process object details.

The following table describes the columns that the **Generate Objects** tab displays:

Column Name	Description
Name	The name of the process object that was generated from the schema.
Description	The schema from which the process object was generated. You can edit the description.
No. of Fields	The number of complex types that the process object contains.
Tags	The tags to be used for the process object. By default, the process object tags are derived from the tags that you specified for the XSD schema. You can edit the tags if you want to specify different tags at the process object level. To edit the tags for a process object, double-click under the **Tags** column against the row that contains the process object, and update the tags.

11. Click **Next**.

The **Import Preferences** tab appears displaying the number of process objects that will be created.

The following image shows the **Import Preferences** tab:

12. Optionally, click **Select** to specify a location where you want to save the process objects.

Default is the location that was selected when you opened the **New Process Objects Creation from OpenAPI 3.0 Files** wizard.

13. Optionally, specify additional tags for the process objects as comma-separated values.

14. Click **Start Import**.

    Application Integration starts importing the process objects and displays the import status.

    The following image shows the import status:

15. Perform one of the following actions:

    - To dismiss the dialog box and get notified when the process objects import is complete, click **Dismiss**.

    - To view and fix import errors, and retry the import, stay on the dialog box.

    After importing the process objects, Application Integration displays a status message indicating if the import was successful or failed.

    If the import failed, check the **Related Logs** tab, fix the errors, and try importing the process objects again. You can also click the **Download** icon to download the entire log.

    The following image shows a sample log stating that 8 process objects have been imported successfully:

New Process Objects Creation from OpenAPI Files

| 1 Select Source | 2 Load Resources | 3 Generate Schemas | 4 Generate Objects |
| 5 Import Preferences | 6 Import Objects |

Review   **Related Logs (5)**

2022-05-16T10:38:49.174Z [INFO] : Import destination path is: /Explore/Default

2022-05-16T10:38:49.829Z [INFO] : Created import job: apYgyQYy8qjhTOg5PwlaUH. Initial status: NOT_STARTED

2022-05-16T10:38:50.455Z [INFO] : Import job apYgyQYy8qjhTOg5PwlaUH has been started.

2022-05-16T10:38:54.972Z [INFO] : Porcess Object import job completed.

2022-05-16T10:38:54.98Z [INFO] : Process Objects importing has been finished. Successfully imported 8 Process Objects. Failed to import 0 Process Objects. Process Objects use 0 tags

Done

16. Click **Done**.

17. Navigate to the location that you specified for the process objects creation and review the process objects.

Creating Process Objects from JSON Files

You can easily and simultaneously create multiple large and hierarchical process objects by importing one or more JSON files.

Use the **New Process Objects Creation from JSON Files** wizard to create process objects from JSON files. After you specify the source, Application Integration parses the source, loads the resources, and generates XSD schemas. Based on the XSD schemas, Application Integration generates the process objects. You can specify import preferences such as the location for saving the process objects and optionally specify the tags to be used for the process objects.

In case errors occur during the process object creation, the wizard displays the errors under the **Related Logs** tab of the corresponding step. You can review the log and take corrective actions. You can also download the

log. When you download the log at a particular step, the log also shows cumulative logging information for all the previous steps.

1. In Application Integration, click **New**.

    The **New Asset** dialog box appears.

    The following image shows the **New Asset** dialog box:

2. Click **Process Object** on the left pane.

3. On the right pane, expand the **Process Object Creation Patterns** list, click **Create from JSON**, and then click **Create**.

    The **New Process Objects Creation from JSON Files** dialog box appears.

    The following image shows the **New Process Objects Creation from JSON Files** dialog box:

4. In the **Source** field, select one of the following values:

Option	Description
File	Select **File** to specify a single JSON source file with payload from your local system. You can also specify HTTP resources. Click **Choose File** to select the JSON source file. The maximum allowed file size for a single file is 5 MB. Default is **File**.
Zip (multiple files)	Select **Zip** to specify multiple JSON files that contain samples of payloads to be received or returned by the process in the form of a zip file from your local system. Click **Choose File** to select the zip file. The maximum allowed file size for a zip file is 3 MB.
URL	Select **URL** to specify a URL that contains an JSON source file with payload. The **URL** and **Use Authentication** fields appear. In the **URL** field, enter the source URL. Select the **Use Authentication** option to specify the user name and password to access the source URL.

5. Click **Next**.

   Application Integration parses the specified source file or URL, and displays the resource details on the **Load Resources** tab. If you import a zip file, the **Load Resources** tab contains a list of JSON resources.

   The following image shows the **Load Resources** tab with the resource details:

6. Review the resource details.

   The following table describes the columns that the **Load Resources** tab displays:

Column Name	Description
File Name	The file name of the resource.
File Path	The URL or relative location of each resource within the source.
Type	The file type of the resource. Displays the value as **JSON**.
Status	The status of the resource. Displays one of the following values: - **Valid**. Indicates that the file type is supported and that the file content is valid. - **Invalid**. Indicates that either the file type is not supported or the file content is not valid.
Comments	Indicates whether the file type is supported or not.

If there are parsing errors, Application Integration displays the error details on the **Related Logs** tab. Fix the errors and try again.

7. Click **Next**.

   Application Integration converts the resources into XSD schemas based on which the process objects will be created. It displays the schema details on the **Generate Schemas** tab.

   The following image shows the **Generate Schemas** tab with the schema details:

8. Review the schema details.

   The following table describes the columns that the **Generate Schemas** tab displays:

Column Name	Description
Schemas	The name of the generated XSD schema file.
Source	The source from which the XSD schema file was generated.
Complex Types	The number of complex types that will be created for each XSD schema.
Prefixes	The prefix to be used for the XSD schema. You can use a prefix to easily identify process objects that are created from a schema. For example, you might want to add the prefix **Salesforce** for all process objects that are created from a Salesforce schema. To specify a prefix for a schema, double-click under the **Prefixes** column against the row that contains the schema, and enter a prefix. All the process objects that are created from the schema will use the same prefix that you specify in this field.
Tags	The tags to be used for the XSD schema. You can use a tag to group related schemas together. For example, you might want to group all schemas containing financial resources under the tag **Finance**. To specify a tag for a schema, double-click under the **Tags** column against the row that contains the schema, and enter a tag. All the process objects that are created from the schema will use the same prefix that you specify in this field.

If there are schema generation errors, Application Integration displays the error details on the **Related Logs** tab. Fix the errors and try again.

9. Click **Next**.

    Application Integration applies the prefix and tags that you specified for the XSD schemas, and generates the process objects based on the XSD schemas. It displays the process object details on the **Generate Objects** tab. The process object names are suffixed with the term **Type**.

    The following image shows the **Generate Objects** tab with the process object details:

10. Review the process object details.

    The following table describes the columns that the **Generate Objects** tab displays:

Column Name	Description
Name	The name of the process object that was generated from the schema.
Description	The schema from which the process object was generated. You can edit the description.
No. of Fields	The number of complex types that the process object contains.
Tags	The tags to be used for the process object. By default, the process object tags are derived from the tags that you specified for the XSD schema. You can edit the tags if you want to specify different tags at the process object level. To edit the tags for a process object, double-click under the **Tags** column against the row that contains the process object, and update the tags.

11. Click **Next**.

    The **Import Preferences** tab appears displaying the number of process objects that will be created.

    The following image shows the **Import Preferences** tab:

12. Optionally, click **Browse** to specify a location where you want to save the process objects.

   Default is the location that was selected when you opened the **New Process Objects Creation from JSON Files** wizard.

13. Optionally, specify additional tags for the process objects as comma-separated values.

14. Click **Start Import**.

   Application Integration starts importing the process objects and displays the import status.

   The following image shows the import status:

15. Perform one of the following actions:

    - To dismiss the dialog box and get notified when the process objects import is complete, click
      Dismiss.

    - To view and fix import errors, and retry the import, stay on the dialog box.

    After importing the process objects, Application Integration displays a status message indicating if the
    import was successful or failed.

    If the import failed, check the **Related Logs** tab, fix the errors, and try importing the process objects
    again. You can also click the **Download** icon to download the entire log.

    The following image shows a sample log stating that 8 process objects have been imported
    successfully:

16. Click **Done**.

17. Navigate to the location that you specified for the process objects creation and review the process
    objects.

Rules and guidelines for JSON files

Consider the following rules and guidelines when you import a JSON file to create a process object:

- When you import a JSON file to create a process object, in addition to the objects defined within the JSON
  file, a root process object with the file name is also created.
  Consider that you have a JSON file named `sampleJson.json`, which contains the JSON payload as shown
  in the following sample:

```
{
"address": "India",
"age": 25,
"contact":
    {
    "telephone": 5434344985
    },
"ID": 111,
"name": "",
"test":
    {
    "contact":
```

```
                    {
                    "mobile": 1267673455
                    }
            }
        }
```

After you import the `sampleJson.json` file, three process objects named `ContactType`, `TestType`, and `SampleJsonType` are created.

The following image shows the process objects that are created after importing the JSON file:
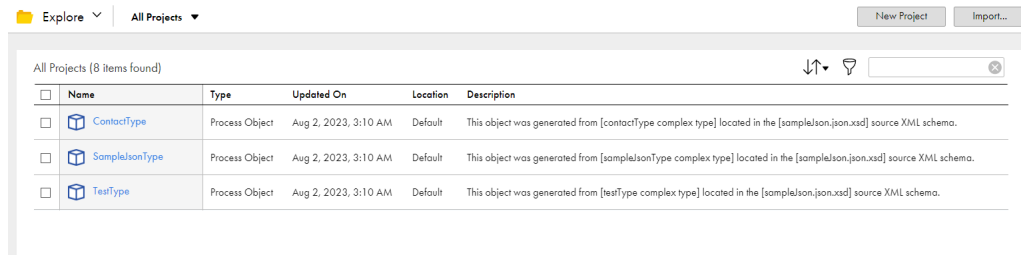
- If the JSON file name contains only numeric characters, `RootPO` is prefixed to the process object name. For example, if the file name is `1234.json`, the process object name would be `RootPO1234`.

- If there are multiple process objects with the same name in a single file, only one process object is created and reused. All the fields are merged into the same process object.
  For example, in the above `sampleJson.json` file, there are two objects with the name contact. When you import the JSON file, the `telephone` and `mobile` fields are merged into the `ContactType` process object.

- If the process objects with the same name already exist in the selected folder, the following message appears in the log:
  ```
  [WARNING]: Process object <Process_Object_Name> imported with warning. Resolution:
  CONFLICT, Status: SUCCESSFUL, Status Code: MigrationSvc_055, StatusMesssage: Reuse
  existing object
  ```

Required and nullable elements

You can define process object fields as required and nullable in the WSDL, Swagger JSON or YAML, and OpenAPI 3.0 JSON or YAML files before you import the files to generate a process object.

In a WSDL file, to define a field as nullable, set the `nillable` attribute to `"true"` in the field element. If you do not specify the `nillable` attribute, the field is set to not nullable by default. If you do not want to set a field as required, set the `minOccurs` attribute to `"0"` in the field element. If you do not specify the `minOccurs` attribute, the field is considered as required by default.

For example, consider a WSDL file that has a schema named ProcessObjectReq. The schema is of a complex type and contains two elements called as Name and ID. To set the ID field as nullable and not required, set the `nillable` attribute to `"true"` and the `minOccurs` attribute to `"0"` as shown in the following sample:

```
<xsd:complexType name="ProcessObjectReq">
    <xsd:sequence>
        <xsd:element name="Name" nillable="false" type="xsd:string"/>
        <xsd:element name="ID" minOccurs="0" nillable="true" type="xsd:integer"/>
    </xsd:sequence>
</xsd:complexType>
```

When you import the WSDL file, the nullable and required field elements defined in the WSDL file are propagated to the process object as shown in the following image:

Similarly, you can set the required and nullable field elements in the Swagger JSON file and create a process object as shown in the following sample:

```
"definitions" : {
  "Process1-3Request" : {
    "type" : "object",
    "required" : [ "Test" ],
    "properties" : {
      "Test" : {
        "$ref" : "#/definitions"/ProcessObjectReq"
      }
    }
  },
  "ProcessObjectReq" : {
    "type" : "object",
    "properties" : {
      "Name" : {
        "type" : "string"
      },
      "ID" : {
        "type" : "integer",
        "format" : "int32",
        "x-nullable": true
      }
    },
    "required": [
      "Name"
    ]
  }
}
```

You can also set the required and nullable field elements in the OpenAPI 3.0 file and create a process object as shown in the following sample:

```
"ApiResponse": {
  "type": "object",
  "properties": {
    "code": {
      "type": "integer",
      "format": "int32",
      "required": true,
      "nullable": false
                },
```

```
                    },
...
```

If you use a process object with nullable fields in a process, and generate the Swagger file for the process, the generated Swagger file might not contain the `x-nullable` field defined in the process object. The field is set to not nullable by default. Therefore, Informatica recommends that you use a WSDL file or OpenAPI 3.0 file to preserve the nullable field values.

When you use this process object in a process, the fields marked as required and nullable use the same restrictions in the generated WSDL file, Swagger file, and OpenAPI 3.0 file.

When you import the JSON file and create a process object, all the fields are marked as nullable and not required by default.

Rules and Guidelines for Process Object Import

Consider the following rules and guidelines when you import process objects from WSDL, XSD, Swagger JSON or YAML, OpenAPI 3.0 JSON or YAML, and JSON files:

- Application Integration does not generate process objects if the WSDL, XSD, Swagger, OpenAPI 3.0, or JSON document is incorrect.

- If you use WSDL or XML schema files, Application Integration generates process objects from complex types and does not use the element declarations.

- Application Integration converts enumerations to picklists.

- Application Integration maps the unsupported types to string or the closest related type. Application Integration also prints a warning message about the type conversion on the **Related Logs** tab of the associated step.

- If you use a Swagger file, Application Integration generates process objects only from the type definitions available in the **Definitions** section. Application Integration does not generate process objects for the types that are used inline in the path declarations.

- When multiple types share the same name, Application Integration adds indexes to the generated process object names to avoid duplication.

- When you import process objects, you cannot overwrite existing process objects in the same location. Application Integration does not import duplicate process objects and instead uses the existing process object.

- When you import process objects with names that include the period (.) character from a URL that contains a WSDL source file, Application Integration replaces the period (.) character with the following string:

  `_dot_`

  For example, if a process object name is `Cart.CartStatus`, Application Integration changes the name as follows:

  `Cart_dot_CartStatus`

- When you import process objects with names that include the period (.) character from a URL that contains an OpenAPI 3.0 source file, Application Integration replaces the period (.) character with an underscore (_) character.

  For example, if a process object name is `Dept.DeptID`, Application Integration changes the name as follows:

  `Dept_DeptID`

- When you import process objects with file names that include Japanese characters, Application Integration fails to create a process object.

CHAPTER 6

Designing Service Connectors

You can use Process Designer to interact with programs and services that are in Data Integration, inside your firewall, or anywhere in the cloud, as long as they have an API. For example, Google provides many APIs to access the services that Google provides.

You can create two types of service connectors:

Service Connector

You can create a service connector to connect to third-party services using REST or SOAP APIs.

Use one of the following ways to create a service connector:

- Manually create a service connector.

- Generate a service connector by importing a WSDL, Swagger JSON, or OpenAPI 3.0 JSON or YAML file.

- Download a service connector from GitHub.

- Download a service connector from Informatica Marketplace.

- Download a service connector from Administrator.

Data Access Service Connector

You can create a data access service connector on a Secure Agent to access data directly from a database.

You can use the data access service connector to connect to the following databases:

- IBM DB2

- Microsoft SQL Server

- MySQL

- Oracle

- PostgreSQL

You can create a data access service connector by defining the database properties and specifying variables to interact directly with the database. Using this service, you can create an invoke activity that executes SQL CRUD operations such as select, insert, update, or delete on a specified data source and receives responses based on the columns in the database table, expression, result set, and attachments from the data source.

To use the processes and services, perform the following steps:

1. Create a service connector or data access service connector, an object that defines the interaction between Process Designer and the service, so that the connector can send and receive data.

2. Define an app connection that a process can use to access the service.

3. Add a Service step to the process where you want to use the service.

For example, the step shown in the following image uses the Get Vehicle Detail action to obtain information from Edmunds:

The services you enable using a service connector or data access service connector are accessible from processes.

The following image shows the process you go through to create a service connector:
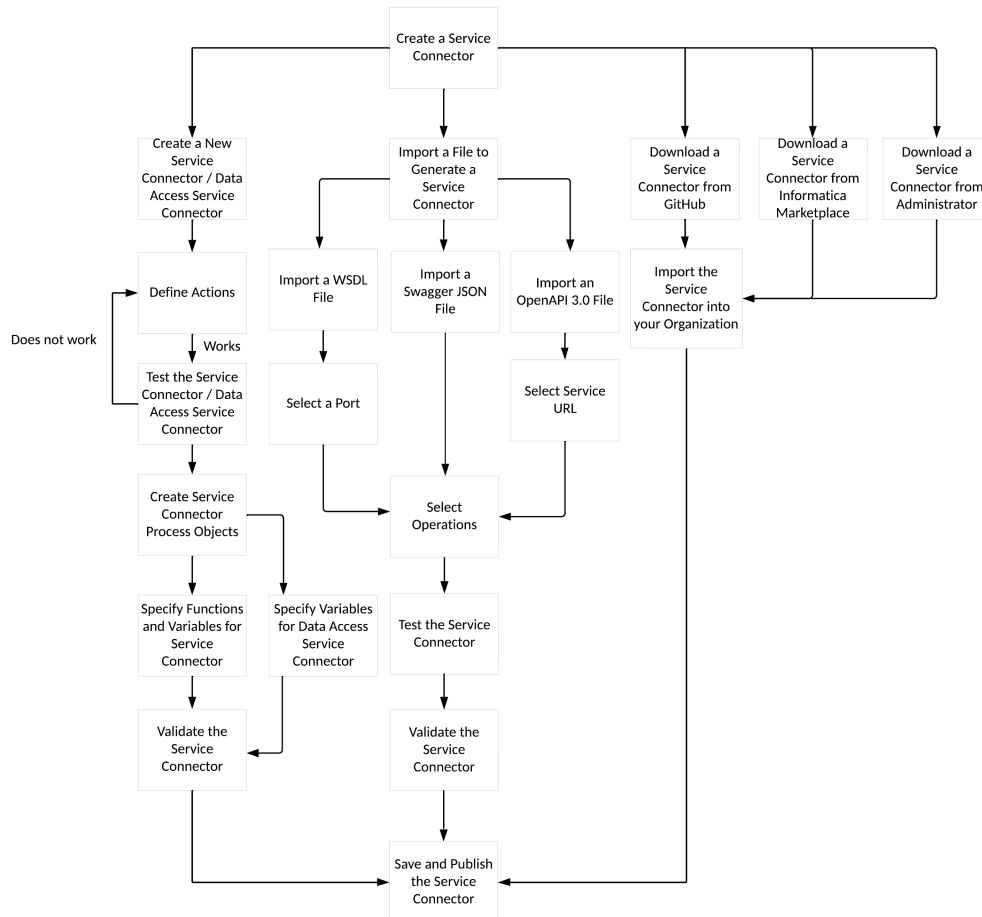
Option 1: Creating a New Service Connector

You can create a service connector by defining properties and specifying functions and variables. You can also define the actions associated with a service connector.

To create a service connector, perform the following steps:

1. In Application Integration, click **New** > **Service Connectors**.
2. In the **New Asset** dialog box, click **Service Connector using Form**, and then click **Create**.
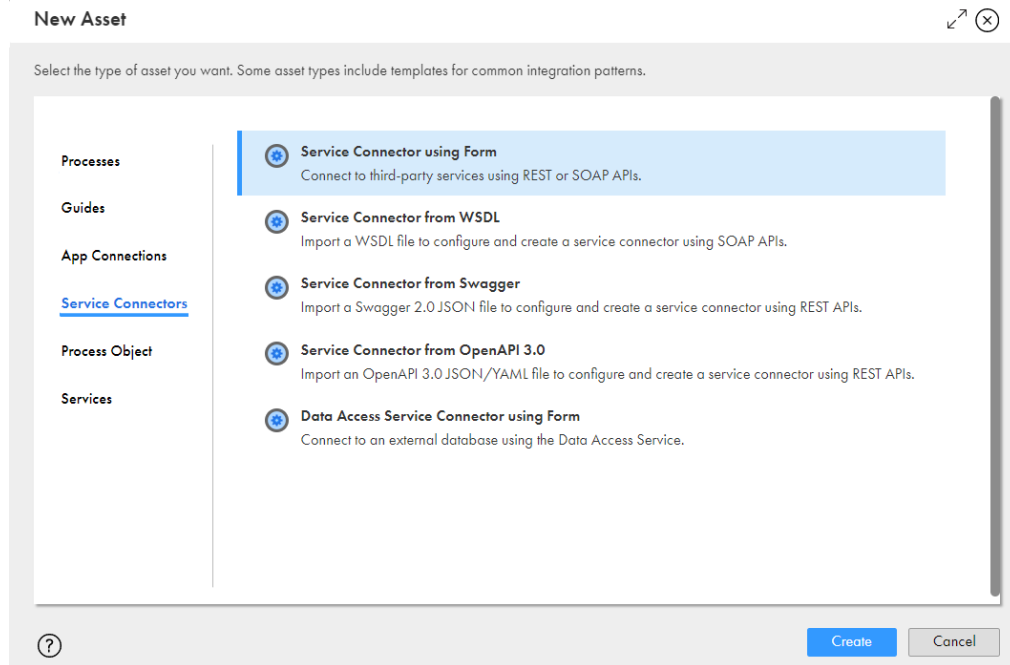
Defining Properties

To create a service connector and define the properties, perform the following steps:

1. In Application Integration, click **New** > **Service Connectors**.
2. In the **New Asset** dialog box, click **Service Connector using Form**, and then click **Create**.
3. Define the following basic properties for the service connector on the **Definition** tab:
   - **Name**: The name by which the service connector is made available for processes. The name must start with a letter or number, and can contain only alphanumeric characters, multibyte characters, spaces, underscores (_), and hyphens (-). The name must not exceed 128 characters. This is a required field.
   - **Location**: Specify the project or folder to save the service connector.
   - **Description**: Enter a description for the service connector.
   - **Agent Only**: Check if this service connector should only be available to run on the Secure Agent.
4. Click **+** to add the connection properties to connect to this service. Specify the following details for each connection property in the **Connection Properties** section:
   - **Name**: The connection property name. The name must start with a letter or number, and can contain only alphanumeric characters, spaces, underscores (_), and hyphens (-). This is a required field.

- **Description**: Enter a description for the connection property.

- **Test With**: Enter a value that Process Designer will use to test the connection.

- **Type**: Select an appropriate data type for the connection property.

- **Required**: Select this check box to set the connection property as a required property.

- **Encrypt**: Select this check box to encrypt the connection properties. In the Developer Console, the values entered in the fields are visible when you save the connector. The value appears encrypted after you save, close, and reopen the connector.

The following image shows a sample service connector:
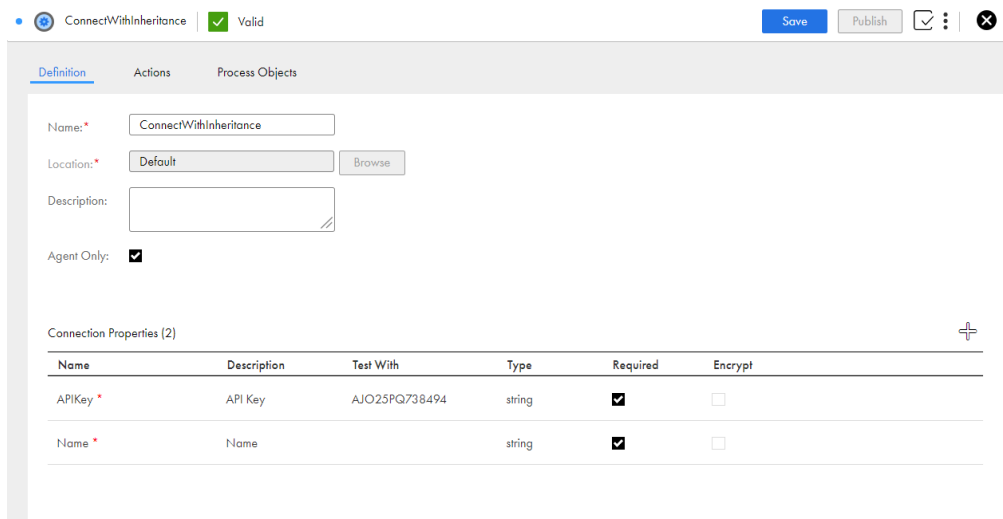
5. To edit the connection properties, click anywhere on the row that contains the connection property. All the properties in the row become editable.

6. Click **Save**.

For more information about the actions and process objects that define service connectors, see "Defining Actions" on page 240 and "Creating Service Connector Process Objects" on page 253.

Specifying Functions and Variables

To configure service connectors, you may need to specify variables and functions to define bindings, output fields, and other properties.

Built-in Variables

You can reference these variables in a service connector using an XQuery expression:

Variable	Variable Type	Description
$VariableName	All connection properties	Properties, input and Other Parameters can be specified using this format.
$ResponseStatusCode	Output field mapping	HTTP response code.

Variable	Variable Type	Description
$ResponseHeaders	Output field mapping	Contains the HTTP response header in an element list where each item is: <header name="Content-Type">text/plain</header> For example: $ResponseHeaders[@name = "Content-Type"]/text()
$RESTResponse	Output field mapping	Contains the RESTResponse XML data, including the headers and code. Visible in the Test Results for the Service Connector.

Output Field Mapping Functions

When you define bindings in the service connector, you can use many functions in the Expression Editor.

For information on all the available functions, see "Using Functions" on page 29 and "Using the Expression Editor" on page 23.

The following table describes functions available for service connector output field mappings:

Function	Syntax	Description
responseHeaderExists	svc:responseHeaderExists($ResponseHeaders, headerName) : boolean	Returns a Boolean value that indicates if a header parameter exists within the response.
getResponseHeader	svc:getResponseHeader($ResponseHeaders, headerName, defaultValue) : string svc:getResponseHeader($ResponseHeaders, headerName): string	Returns a response header value. If the header parameter is not defined, this function returns the optional default value.
getResponseHeaderNames	svc:getResponseHeaderNames($ResponseHeaders) : list of strings	Returns a list that contains the names of all the parameters within a response header.

Defining Actions

On the Actions tab, you can create and describe one or more actions associated with a service connector.

The following image shows the **Actions** tab:
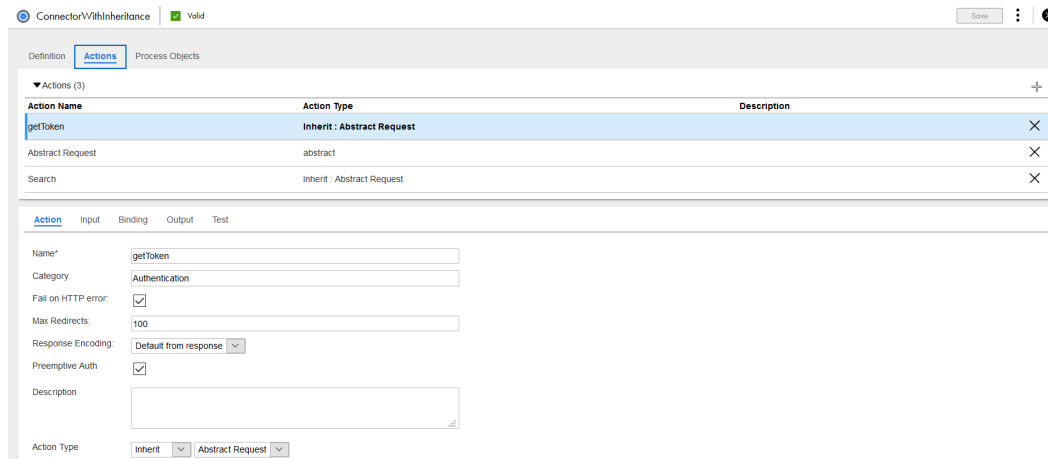
Choose the row you want to edit and then enter the following information in the Action details tab:

Click **+** to add a new row.

- **Action Name**: Enter the name that appears in lists when referencing this action in service connectors and connections. The name must start with a letter or number, and can contain only alphanumeric characters, spaces, and underscores (_). This is a required field.

- **Category**: If you have many service connectors, each with multiple actions, you can create categories to help users navigate within processes.

- **Fail on HTTP error**: Check this option if you want Process Designer to acknowledge that an error occurred when you send a request to a server using this action. For more information see "Checking for HTTP Errors" on page 251.

- **Max Redirects**: Specifies the maximum number of redirects that an action can make. When an action accesses an endpoint, the endpoint might redirect the action to another endpoint. Enter a value to control the number of redirects.
  The default **Max Redirects** value is 100.

  Enter **0** to disable redirects.

  For example, if you enter **3**, the service connector action makes a maximum of three redirects and receives three redirect responses from the endpoint. If the endpoint requests a fourth redirect response, you see an error message.

- **Preemptive Auth**: Sends authentication details, that is, user credentials, along with a request to an endpoint. Select **Preemptive Auth** when you know that an endpoint requires authentication. If you do not select **Preemptive Auth** and the endpoint requires authentication, the following events occur:

  1. The service connector sends a request to the endpoint.

  2. The endpoint requests authentication.

  3. The service connector sends user credentials to the endpoint.

  If you select **Preemptive Auth** and the endpoint requires authentication, you avoid an extra request.

- **Action Type**: You can select one of the following action types for each action. Refer to "Shared Service Actions" on page 242 for more information.
  - **General** actions are published for use only with a particular service connector.
  - **Abstract** actions are not published but can be shared with other actions, that is, made available for reuse as a template for other actions.
  - **Inherited** actions inherit properties from an Abstract action.

- **Description**: Enter a description or notes about this action.

For each action, specify additional details on the **Input**, **Binding**, **Output**, and **Test Results** tabs.

Shared Service Actions

Using shared service actions, you can define common inputs, bindings, and outputs, then reuse these definitions in other actions that "inherit" from the parent "abstract" action.

For example, you might have multiple actions which all share the same:

- Input parameter session-id
- HTTP Header such as content-type
- Binding method (GET/POST) and URL
- Authentication Credentials
- Output fields

In that case, you can:

1. Define an Abstract type action that contains all of the common definitions:
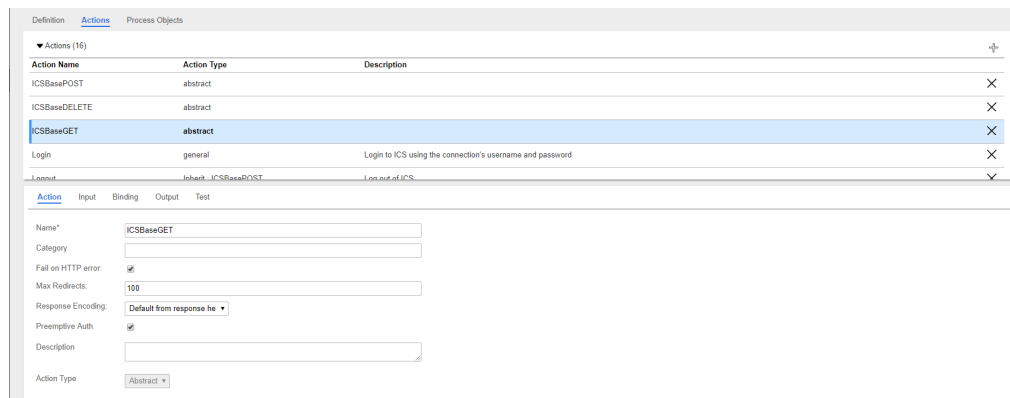
2. Create multiple Inherit type actions that use those common definitions and extend or add to certain parameters.
3. For example, an Inherit action can:
   - Define a new input parameter (for example, $messageId).
   - Extend the binding URL and append the new parameter to the base with "/{$messageId}".

For example:
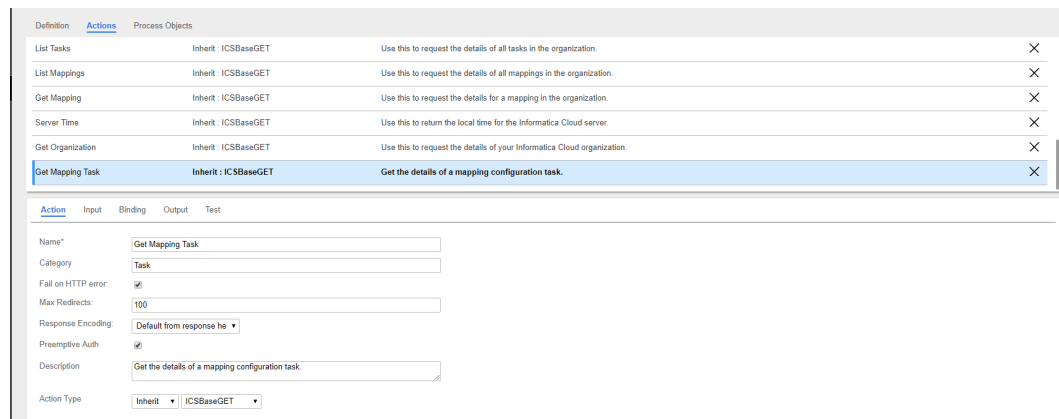
Notes on Usage

Shared service actions:

- Are not available at design time within IPDs and Guides.
- Inherit inputs, bindings, and outputs from a single parent (Abstract) action.
- Do not allow abstract actions to inherit from other abstract actions.
- Allow each Service Connector to have multiple abstract actions.

After you specify the Inherit Action Type, many fields are disabled.

However, for specific Binding and other options, you can choose to:

- **Inherit**: Use all input parameters from the Abstract action, as is. This is the default.
- **Exclude**: Exclude the parameter so it is not available to the action at design time or runtime.
- **Overwrite**: Specify parameters to overwrite what was inherited.

Input Tab

Use the **Input** tab to define input data items that are unique to the service to which you are sending data.

For example:

Depending on the service, you might have no data items or many.

For each item, enter the following properties:

- **Name**: The name of the input data item. The name must start with and can contain letters, numbers, or the special characters .@$^_. The name can also contain spaces. This is a required field.
- **Label**: The name of the item to display within a process. If not specified, the Name is displayed.
- **Type**: Select the data type of the item from the list. If you select the Reference or Object List type, you can select a Process Object.
  You can also set a list of simple type such as text, integer, number, check box, date, date time, or time for an input field. For the initial values, use a sequence of values based on the list of simple type as shown in the following examples:

  - List of text: ("Bob","John","Valery")

  - List of integer: (1,2,3)

  - List of number: (22.5, -23.654, -45.876500)

- List of check box: (true, false, true)

- List of date: ("2022-01-03", "2021-12-11", "2020-03-04")

- List of date time: ("2022-01-03T11:23:01Z", "2021-12-11T23:11:55", "2020-03-04T22:10:45")

- List of time : ("12:30:00", "22:45:23", "05:01:02")

To set the type as a list of simple type, select the **Text**, **Integer**, **Number**, **Checkbox**, **Date**, **Date Time**, or **Time** type as required, and then select the **This is a list** check box.

- **Required**: Select if a value must be set for this parameter.

- **Description**: Enter a description of the parameter.

- **Parameter**: When selected, the service connector passes this parameter to the service. If the parameter is only used when constructing another parameter and does not need to be passed to the service, clear this field. For example, the service might expect a date to be in the RFC 1123 format but you want the process that calls it to pass an XSD date in the Process Designer's normal format. This option is enabled for all data types except XML.

- **Test with**: Enter a value that Process Designer will use to test the action. If the input is an attachment, you can also upload a sample attachment file.
  You can also test the input fields in the JSON format with the values set to a list of simple type as shown in the following example:

```
{
    "ListText": [
        "textJson1",
        "textJson2"
    ],
    "ListInt": [
        1,
        2,
        3
    ],
    "NumberList": [
        12.1,
        12.12,
        12.3
    ]
}
```

After you enter a value in the input field, you must click anywhere outside the row to save the value.

Use the **+** icon to add a new item. Click **X** to delete the current row.

Service Connector Bindings

For each action in a service connector, the Binding options enable you to specify the interface and parameters required to communicate with a service.
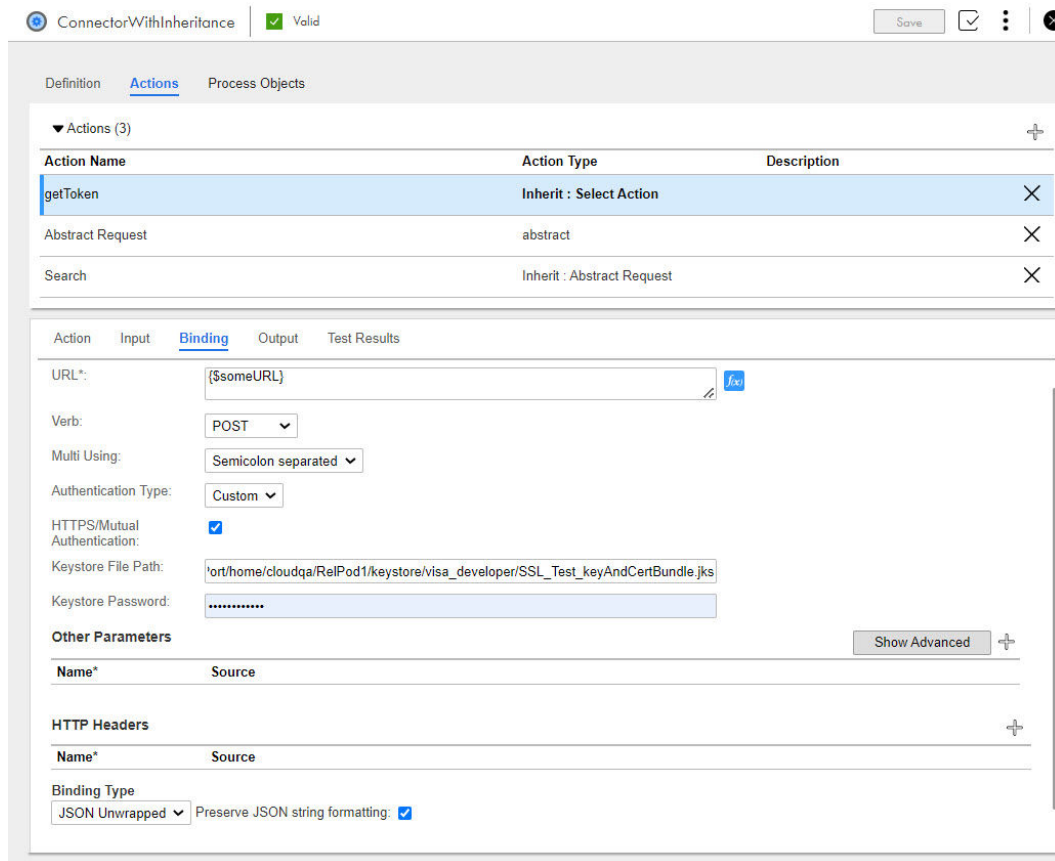
Binding Properties

Specify values for these options:

- **URL**: Enter the URL of the REST Service.

- **Verb**: Choose how to send data to the service:

  - **GET**, **HEAD**, **OPTIONS**, and **TRACE**: Process Designer automatically generates the query parameters.

  - **POST**, **PUT**, **PATCH**, and **DELETE**: Process Designer adds a **Binding Type** field from which you can then select the value as **JSON**, **JSON Unwrapped**, **Form**, **URL**, or **Custom**. If you select **Custom**, a **Body** field appears where you can type the request that is sent to the service. By default, **JSON** is selected for the **POST**, **PUT**, and **PATCH** verbs, and **None** is selected for the **DELETE** verb. Consult the API documentation to determine what you should enter.

- **Multi Using**: This option determines how query string parameters are generated when you need to specify multiple values for a parameter. Choose one of the following:

  - **Semicolon separated**: for a semicolon-separated multi-select list, Process Designer adds input parameter values as a single string. (For example, *?param=a;b;c*). This is the default.

  - **Comma separated**: for a comma-separated multi-select list, Process Designer adds input parameter values as a single string. For example, *?param=a,b,c*.

  - **Append brackets**: for a semicolon-separated multi-select list, Process Designer maps semicolon-separated values to multiple query parameters of the same name, and appends [] to the parameter name. For example, *colors[]=red&colors[]=blue&name=JW*.

- **Append numbers**: for a semi-colon separated multi-select list, Process Designer maps semicolon-separated values to multiple query parameters of the same name and adds a number to the end of the field name, that is, 1..N. For example, *colors1=red&colors2=blue&name=JW*.

- **Authentication Type**: You have two options to define the authentication information that the service requires: Basic or Custom. If you choose:

  - **Basic**, enter the user name and password, if required.
    Enter text or an XQuery expression in the user name and password fields. If you enter an XQuery expression, you can pass the password value through a connection instead of hard coding it in the service connector. Click the glass icon to view the password.

    Select the **Expression in Password** check box to retain the password that you enter and also export the password when you export the service connector.

    Note: If the password contains an ampersand (&) character, when you test the service connector, the action fails. Additionally, if you use the same service connector in an app connection and process, the process invocation also fails.

  - **Custom**, add the security parameters based on the service requirements. For example, you might need to send signed data using an HTTP header.

- **HTTPS/Mutual Authentication**: Select this check box if the action must use a custom client certificate for mutual authentication. You can configure one custom client certificate for each action. If selected, the **Keystore File Path** and **Keystore Password** fields appear. You must place the keystore file on the Secure Agent machine.

  Note: You can configure custom client certificates for service connectors that run on Secure Agents. When you configure a custom keystore file for a service connector and also specify a client certificate using the `javax.net.ssl.keyStore` property in the Process Server properties, the custom keystore file configured in the service connector takes precedence. If the service connector authentication fails, the client certificate configured in the `javax.net.ssl.keyStore` property is ignored and the request fails.

- **Keystore File Path**: Enter the path and file name of the Java KeyStore (JKS) file on the Secure Agent machine.

- **Keystore Password**: Enter the password to open the keystore file.

Other Parameters

Use this section to add input parameters required by the service. The value may be specified as a literal or expression. Refer to documentation from the service you are using.

To add a row and enter multiple parameters, click the **+** sign. To remove a row, click the **X**.

For each row, enter:

- **Name**: the name of the parameter as specified within the service being called.
- **Content**: the value to specify for the parameter, if any.

Click **Show Advanced** to select these optional parameters for the Binding:

- **Sign With**: The value that is computed will first be signed using SHA1, and the result of the signature will be this value.
- **Sign Using**: Choose to sign the field using SHA1 or SHA256.
- **Encode Using**: Choose to encode the field as Base64 and Hex64Upper.
- **Temp**: If checked, Process Designer will not generate a parameter when using GET or POST. For more information, see the Parameters described here: "Input Tab" on page 243.
- **Attachment Base64**: The base64 encoding of the attachment associated with the <inputName>.
- **Attachment Name From**: The file name of the attachment associated with the <inputName>.

Click **Hide Advanced** to hide the fields.

HTTP Headers

If the service requires one or more HTTP headers, enter the Name and Content (value of the parameter) for each header in this section.

To define a SOAP-based service connection, you must add both the "SOAPAction" and "Content-Type" headers. The value of SOAPAction can be empty, dynamically set using GET, or defined in the WSDL.

If the service does not set the content-type in the response header, the service connector attempts to infer the content type from the payload. For example, if the response starts and ends with either "{...}" or "[...]", the response is parsed as JSON. If the response payload starts with an angle bracket ("<"), it is treated as XML. If the response payload cannot be parsed due to malformed JSON or XML, it is processed as a regular string.

If the response Content-Type header contains the term **application** in the first part and the term **json** in a subsequent part, Application Integration infers the response payload type as JSON. Similarly, if the response Content-Type header has the term **text** in the first part and the term **xml** in a subsequent part, Application Integration infers the payload type as XML.

Note: As you type in the Name field, Process Designer displays commonly used values, including:

- Accept
- Accept-Charset
- Accept-Encoding
- Accept-Language
- Content-Type
- SOAPAction

Some services request an MD5 signature. You can generate an MD5 signature in the **HTTP Headers** section of service connector actions.

To generate an MD5 signature, perform the following steps:

- Enter a name for the HTTP header. For example, enter **Signature**.
- Go to **source** > **Expression Editor**.
- Under **type**, select **XQuery**.
- Under **Insert Field**, select **PAYLOAD_DIGEST_MD5**.

When you test or run the service connector, the used variable contains an MD5 checksum for the generated request payload.

Content Type and Character Sets

You can determine the charset used for a request payload as noted here:

- Specify a charset value in the Content-Type header. In this case, the header determines the charset of the request payload. In this case, the charset is based on the value specified in the header unless the request format is an attachment type, in which case each part of a multipart/form-data request will be treated separately. If you set the Content-Type and want the service connector to append the default charset automatically, add a semi-colon after the specified Content-Type, as shown here: Content-Type=application/json;
- Do not specify a charset value in the Content-Type header so this value can be set automatically based on the request format. To ensure that the content can be decoded correctly, the service connector appends the default charset (UTF-8) to the Content-Type. For example, the value might be set as follows: Content-Type=application/json;charset=UTF-8

Expression Editor for Service Connector Properties

To define the binding parameters, you can use the Expression Editor accessible for the URL, Other Parameters, and HTTP Headers fields.

To open the Expression Editor, click *f(x)* next to the field you want to edit. When it opens, the Expression Editor gives you access to a list of available functions and the fields defined for the service connector, as shown in this image:
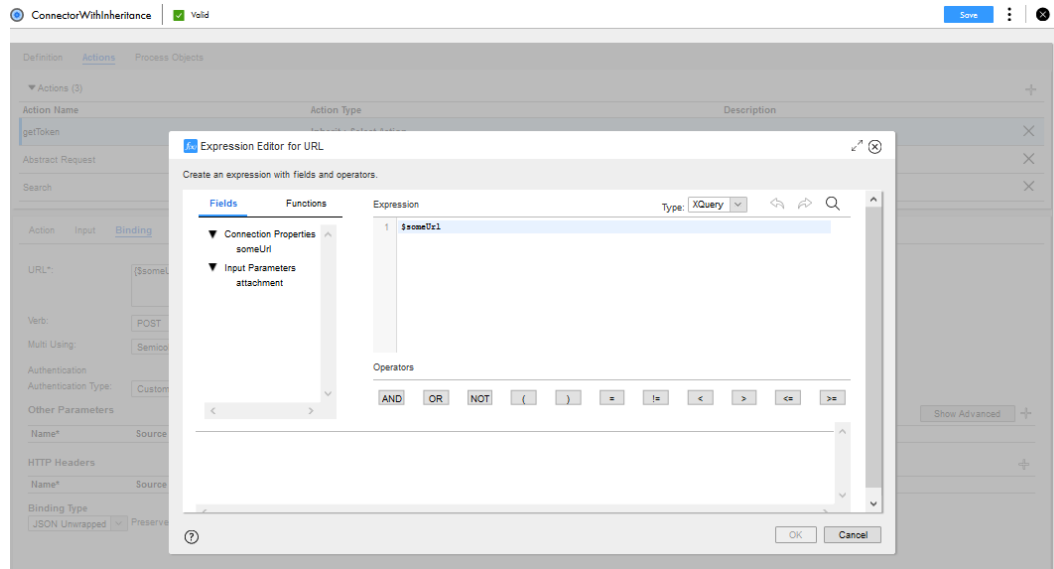
Based on the type selected for the expression, the Expression Editor applies syntax validation. In this case, the post_url field is validated as an XQuery variable.

For more information, see "Using the Expression Editor" on page 23.

Output Tab

Use the **Output** tab to define how the service connector must parse data returned from a service and place it in variables. Typically, a service returns data in the XML format. If the service returns data in the JSON format, the service connector converts it to the XML format. You can map the XML to output fields in two ways:

1. Directly map the data to a property based on XML tag names or JSON properties.
2. Use Expression to extract elements.

   Note: Informatica recommends parsing the XML response by using XML parsing tools such as JDOM and Woodstox, and then extracting values rather than performing a plain string search to match namespaces and tags. If you match plain strings, you might encounter an error because the namespace prefixes are not constant.

The following image shows the **Output** tab:
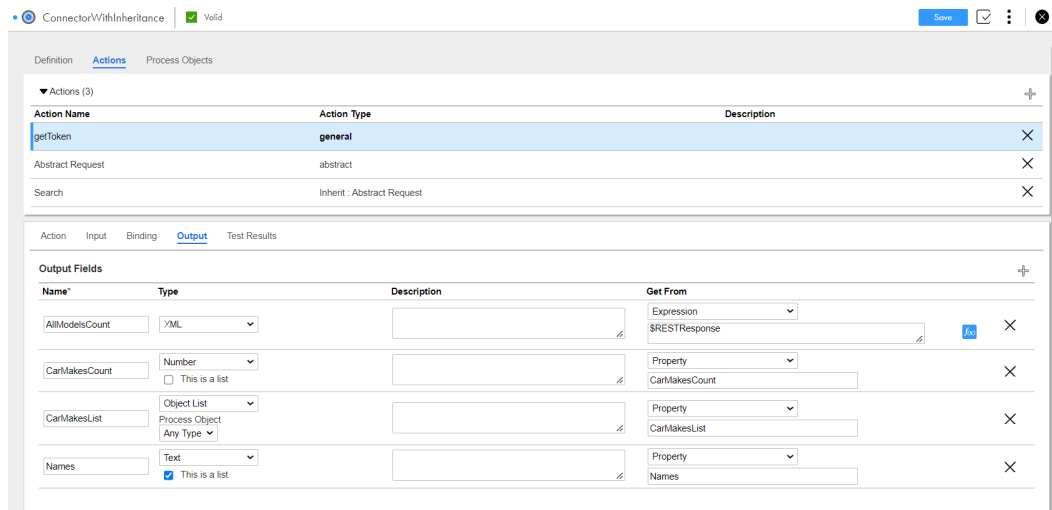
For each output data item, specify:

- **Name**: The name of a variable into which a returned value is placed. The name must start with a letter or number, and can contain only alphanumeric characters, spaces, and underscores (_). This is a required field.

- **Type**: The data type of the value being written to the variable. If the type is Object List or Reference, Process Designer displays a list of Process Objects so you can choose one of the objects defined within the **Process Objects** tab.
  You can also set a list of simple type such as text, integer, number, check box, date, date time, or time for an output field. To set the type as a list of simple type, select the **Text**, **Integer**, **Number**, **Checkbox**, **Date**, **Date Time**, or **Time** type as required, and then select the **This is a list** check box.

- **Description**: Text describing the variable

- **Get From**: Select one of the following options:
  - **Property**: To enter a named value (which is the name used within the XML returned by the service) to be placed within the variable.

    Note: You must select **Property** only for unique fields. If you have duplicate fields, select **Expression** and specify an XQuery expression. For more information, see "Rules and guidelines for using list of simple types" on page 250.

  - **Expression**: To write an expression to parse the XML returned by the service. In the above image, we use the $RESTResponse output field to return one value for the AllModelsCount variable. The expression used for the CarMakesList returns a list. Click **f(x)** to open the Expression Editor where you can type the expression.

  - **HTTP Response Status Code**: To check the HTTP response status code.

  - **HTTP Response Header**: To enter the part of the response header to assign to the field. See the details below.

  - **Entire Response**: To assign the complete contents of the response payload to the field.

  - **Simplified XML**: To rearrange data so it can be used by process objects. For more information, see "Simplified XML" on page 251.

  - **Entire Response As Attachment**: To handle the entire response as an attachment.

    Note: The **Entire Response As Attachment** option preserves backward compatibility. If you had created a service connector with a single attachment earlier, it will continue to work as is, and no manual changes are needed. However, Informatica recommends that you redesign your service connector to handle

multipart responses as multiple attachments so that you do not have to manually split attachments. You can also use the processing logic for attachments within the service connector itself instead of doing it within a process. Therefore, you can use a service connector within multiple processes without having to design the processing logic in multiple processes. You can also use the functions available to work with output attachments.

- **Attachments**: To work with multiple attachments and pass the entire list of attachments to the selected variable except the part used as the payload.

After you enter a value in the output field, you must click anywhere outside the row to save the value.

For details on handling HTTP errors, see .

Rules and guidelines for using list of simple types

Consider the following rules and guidelines when you use a list of simple types in a service connector:

- You must use the **Property** option for an output field only if the payload response contains unique fields. If you have duplicate fields or a field with a list of simple type, there might be instances of one or more values with the same field name in the XML document. In that case, you must select **Expression** and use **XQuery** to get the correct field.
  For example, assume that you select the **Property** option for the payload response as shown in the following sample:

```
<users>
<user1>
    <ID>1</ID>
    <ID>2</ID>
    <ID>3</ID>
</user1>
<user2>
    <ID>4</ID>
    <ID>5</ID>
    <ID>6</ID>
</user2>
<users>
```

When you use a field with a simple type and set the output as ID. Application Integration identifies an instance of the ID and returns a single output value such as the ID with the value as 4. However, if you use a field with a list of simple type, Application Integration finds the output instance. Then, Application Integration returns all the instances from the same level within the element. In this example, if the ID with value 1 in user 1 is identified, Application Integration also returns the IDs 2 and 3 within user 1.

To avoid this issue, when you have duplicate fields such as ID and you want to use the second ID from user 2, use **XQuery** to return the correct field as shown in the following sample:

```
.//user2/ID[2]/text()
```

Consider the following scenarios to use the sequence of values for the list of simple type variables:

If you have text output variables that contain a list of strings, to build a comma-separated string, use XQuery as shown in the following sample:

```
{string-join($TextsOut , ",")}
```

If you have a list of date-time values, and you want to access the second value from a list of dates, use XQuery as shown in the following sample:

```
{$DateTimesOut[2]}
```

If you want to iterate all the integers from the list of integer values, use XQuery as shown in the following sample:

```
{for $j in $Integers
return $j}
```

- When you generate a JSON payload with an input field that contains a list of simple types, the output is in the form of a JSON array. When you parse the JSON response, the fields are also converted into a list of simple types.
  For example, the generated JSON payload appears as shown in the following sample:

  ```
  {
   "userNames": ["Bob","John","Valery"]
  }
  ```

  When you generate an XML payload, the request appears as shown in the following sample:

  ```
  <root>
  <userNames>Bob</userNames>
  <userNames>John</userNames>
  <userNames>Valery</userNames>
  </root>
  ```

Simplified XML

Much of the complexity in standard XML, with a mixture of namespace, attributes, siblings, and children, is not needed within Process Designer in order to present data to people designing processes. While you sometimes need to use the XQuery option for Get From, this is not necessary for many applications.

Simplified XML rearranges data within XML so that it can be used by process objects. This rearrangement treats attributes as children, removes namespaces, makes siblings with the same name consecutive, and places text into CDATA sections.

If you select Simplified XML for the Get From option, Process Designer displays a text field. If you do not enter anything in the field, Process Designer processes all of the received XML. If you enter the name of an element, Process Designer begins simplifying at this element, only processing this element and elements nested within it.

If you type a period ("."), this is treated as an empty field. (This is the default.)

Checking for HTTP Errors

When you send a request to a server and it cannot complete the request, it returns an error status code. There are two ways in which a service connector can handle HTTP errors (4xx and 5xx status code in the HTTP response).

1. For each Action defined on the **Actions** tab, you can select Fail on HTTP error, as shown in the illustration below. By default, this option is enabled. If the HTTP request returns an HTTP error, the service connector fails.

2. Specify how to handle the HTTP errors for each Action on the **Output** tab. To implement this option, disable Fail on HTTP error.

Fail on HTTP Error

If you enable Fail on HTTP error and an HTTP error is returned:

- A process faults and you can view the process details in the Application Integration Console.

- A process displays a dialog to the end user with message text showing the HTTP error received.

- The service connector throws faults in the same format a process throws faults:

  ```
  <sf:faultResponse>
      <sf:code>CODE</sf:code>
      <sf:reason>REASON</sf:reason>
      <sf:details>DETAILS</sf:details>
  </sf:faultResponse>
  ```

HTTP error codes follow the format HTTP_NNN, for example, HTTP_404. The reason string contains the HTTP status message.

Note: Any internal or system errors return SERVICE_CONNECTOR_ERROR and one of the following reason strings:

- CATALOG_ERROR
- OTHER_PARAMETERS_ERROR
- AUTHENTICATION_ERROR
- CUSTOM_HEADERS_ERROR
- BINDING_URL_ERROR
- ALTER_REQUEST_AUTHENTICATION_ERROR
- PROCESS_RESPONSE_AUTHENTICATION_ERROR
- OUTPUT_PARAMETERS_ERROR

If you disable Fail on HTTP Error and an HTTP error is returned:

- A process does not fault and the service connector passes the 4xx or 5xx status codes to the process so that you can handle the error in the process.
- A or process faults and displays a failure message.

If the target service uses error codes that users might encounter as a matter of course, enable this option to provide process users with meaningful information.

The following image shows the Fail on HTTP error option enabled:
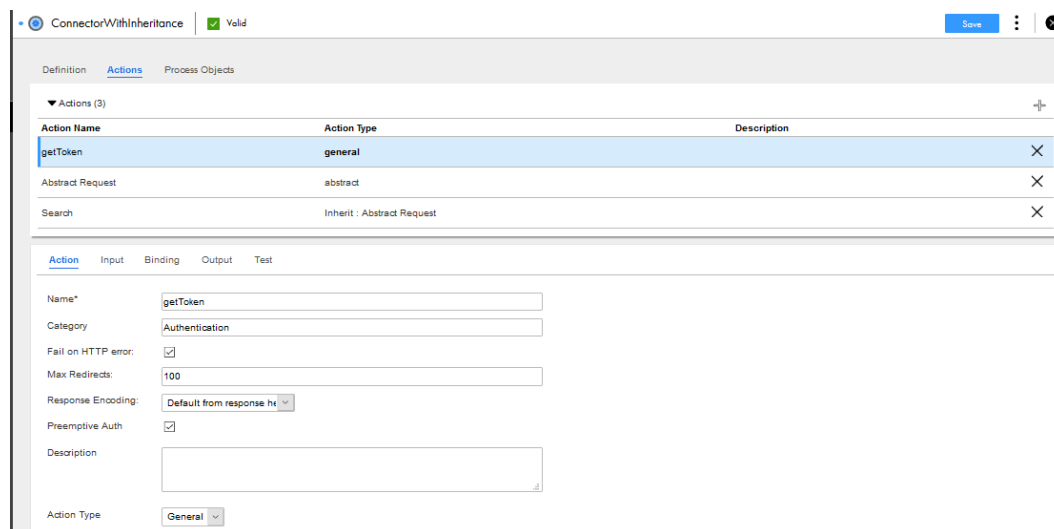
Handle HTTP Errors

To handle HTTP errors for a process, first be sure that you have disabled **Fail on HTTP error**.

You then need to check the HTTP response status code within the process.

To do this, define an "HTTP Response Status Code" as a variable in the **Output** tab. You can also specify a variable using $ResponseStatusCode in Expression fields (see the available functions below). In this way, you can pass the status code as part of the service connector's output and handle the response in the process.

These variables, like all variables you create in the **Output** tab, display when you click **Test**.

If you check Fail on HTTP error and an HTTP error status code is returned, the output contains a message.

HTTP Response Header Information Using Expression

You can use one of the following to get details from the response headers:

$ResponseHeaders[@*name* = "Content-Type"]/text();**

This constant contains the HTTP response header.

fn:getResponseHeader($ResponseHeaders, *header_name***[,** *default_value***]);**

Returns a response header value. If the header parameter is not defined, this function returns the optional default value.

fn:responseHeaderExists($ResponseHeaders, *header_name* **);**

Returns a Boolean value that indicates if a header parameter exists within the response.

fn:getResponseHeaderNames($ResponseHeaders);

Returns a list that contains the names of all the parameters within a response header.

Creating Service Connector Process Objects

On the **Process Objects** tab, you can define one or more process objects for a service connector to group data and create a structured object. When defined in the service connector, the process objects are available to processes that use the service connector. If, for example, a service returns demographic information such as name, address, and phone number, you might create a single Demographic process object that contains this information.

You can associate each process object with one or more of the data elements returned by the service using the **Actions/Output** tab.

To create service connector process objects:

1. Create or open an existing service connector.

2. Click the **Process Objects** tab.

3. Click the **+** sign to add a new process object or select an existing process object from the list. Each process object appears as a line item on the tab.
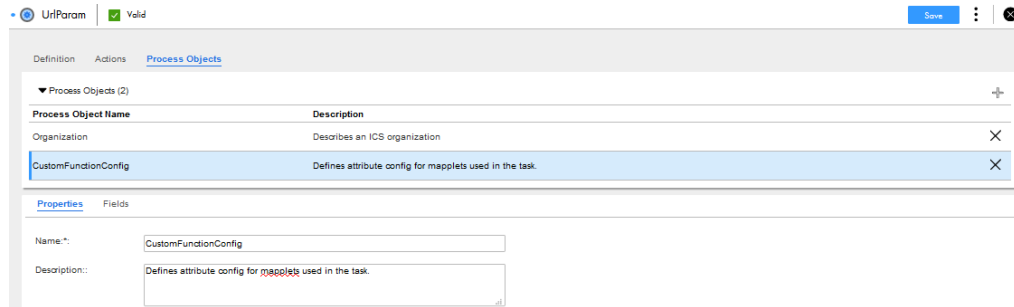
4. On the **Properties** tab, specify the following details for each process object:

- **Name**. Enter a process object name that identifies the process object. This name appears in the lists where the process object is available for selection. The name must start with a letter, and can contain only alphanumeric characters, underscores (_), and hyphens (-). This is a required field.

- **Description**. Enter an optional description.

5. On the **Fields** tab, specify the following details for each process object:

- **Name**. Enter a name for each field in the process object.

- **Type**. Select the data type of the value being written to the variable using one of the built-in data types. If you select the Reference or Object List type, you can select a process object.
  You can also set a list of simple type such as check box, date, date time, integer, number, text, or time for a process object field.

  To set the type as a list of simple type, select the **Checkbox**, **Date**, **Date Time**, **Integer**, **Number**, **Text**, or **Time** type as required, and then select the **This is a list** check box. For more information about using a list of simple type, see "Creating a field with a list of simple type" on page 103.

- Optionally, expand the row after you select a data type such as text, date, date time, time, number, or integer. You can enter the following values for each of the data types:

  - character length and text format for the text data type

  - date range for the date data type

  - date time range for the date time data type

  - time range for the time data type

  - minimum and maximum values for the number data type

  - minimum and maximum values for the integer data type

- **Required**. Select this check box to set the field as a required field.
  A required field must be included in the request payload. It can contain a value, be empty, or be null.

- **Nullable**. Clear this check box if the field must not accept null values. By default, the **Nullable** check box is selected.

For more information about using process objects in a process, see "Rules and Guidelines for using Process Objects in a Process" on page 199.

Testing the Service Connector

When you use the service connector editor, click **Test** to send a request to the service and display the response data on the **Test** tab:

- **Generate Process Objects**: Click to see the process objects that you have defined for the service connector.
- **Result**: Displays the status of the test.
- **HTTP Status**: Displays the HTTP status code.
- **URL**: Displays the URL where test data was sent. Any query parameters are added to the URL shown here.
- **Output** and **Value**: Shows the data returned from the service that is assigned to the variables defined on the **Output** tab. If you are using XSLT, it is used to extract the data from the payload.
- **Response Payload**: Shows the payload sent to Process Designer from the service (not the full response sent by the service).
- **REST Request**: Shows the data sent to the service from Process Designer.
- **REST Response**: Shows all of the data sent back to the Process Designer from the service (whereas the response payload is only a portion of the returned data).

Here, you can see the differences between the payload, request, and response data (for illustration purposes only; the actual test results display only request *or* response data):
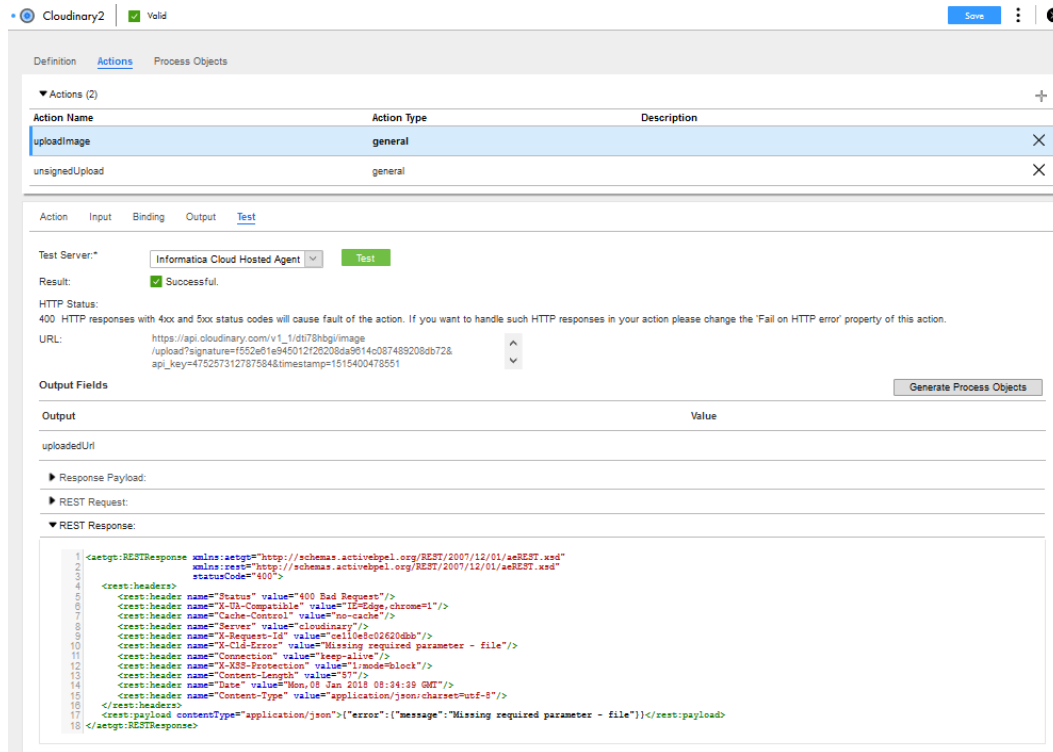
If the response includes one or more attachments, you can also download the attachments. The following image shows a multipart response with options to download all the attachments:

Generate Process Objects

A process object is usually tied to a specific set of elements. You may sometimes have a large number of identical objects, each of which applies to one set of elements. For example, when you define the data being returned, you create a process object whose sole purpose is to define a subset of returned data. These objects are *non-reusable* and can only be used by a single process object field. The object names are also the name of a field.

You can also create objects that are *reusable*. For example, the refType element in NetSuite has two fields: a name and an internalID. Instead of creating many objects, each of which contains these two fields, you can create a single, reusable process object.

1. Click **Generate Process Objects** to show the process objects that you have defined for the service connector.

2. Select the process objects that you want to make reusable and then click **Next**.
   Process Designer displays a list of generated process objects.

3. Click **Finish**.

Service Connector Timings

You can see the HTTP Response Parsing time, the HTTP Execution Time, and the Redirection Count in the tab of a service connector.

You can see the following HTTP headers in the **Rest Response** section of the tab:

HTTP Response Parsing Time

The time that a service connector takes, in milliseconds, to parse a response from the URL you entered in the **Binding** tab. The HTTP Response Parsing time is usually zero for small requests. For large requests, especially with requests with attachments, the HTTP Response Parsing time increases.

See the HTTP header `X-AE-HTTP-RESPONSE-PARSING-TIME-IN-MILLIS` for the HTTP Response Parsing Time.

Redirection Count

The number of redirects, if any, made by the URL that you entered in the **Binding** tab.

For example, if a service request redirects to a service that in turn redirects to another service, the Redirection Count is two. If a service request goes through with no redirects, the Redirection Count is zero.

See the HTTP header `X-AE-REDIRECTION-COUNT` for the Redirection Count.

Note: If a GET HTTP request redirects to another GET HTTP request, the Redirection Count remains zero. This is a limitation.

HTTP Execution Time

The response time, in milliseconds, of the URL that you entered in the **Binding** tab. The HTTP Execution Time does not include the response parsing time, or the time taken to by the service connector to perform other tasks.

See the HTTP header `X-AE-HTTP-EXECUTION-TIME-IN-MILLIS` for the HTTP Execution Time.

The following image shows the `X-AE-HTTP-RESPONSE-PARSING-TIME-IN-MILLIS`, `X-AE-REDIRECTION-COUNT`, and `X-AE-HTTP-EXECUTION-TIME-IN-MILLIS` HTTP headers:

Option 2: Generating a Service Connector by Importing a File

You can import a Web Service Definition Language (WSDL) file, a Swagger 2.0 JSON file, or an OpenAPI 3.0 JSON or YAML file to create an Application Integration service connector.

Creating a Service Connector from a WSDL File

You can import a WSDL file to easily create a service connector with multiple operations. Use the **New Service Connector from WSDL** wizard to import a WSDL file and create a service connector. After you specify the WSDL file, Application Integration parses the file and loads the operations.

If an error occurs during import, the wizard displays the error under **Error/Warnings** on the **Progress** tab. You can take corrective actions and import the WSDL file again.

1. In Application Integration, click **New** on the left navigation bar.

   The **New Asset** dialog box appears.

   The following image shows the **New Asset** dialog box:
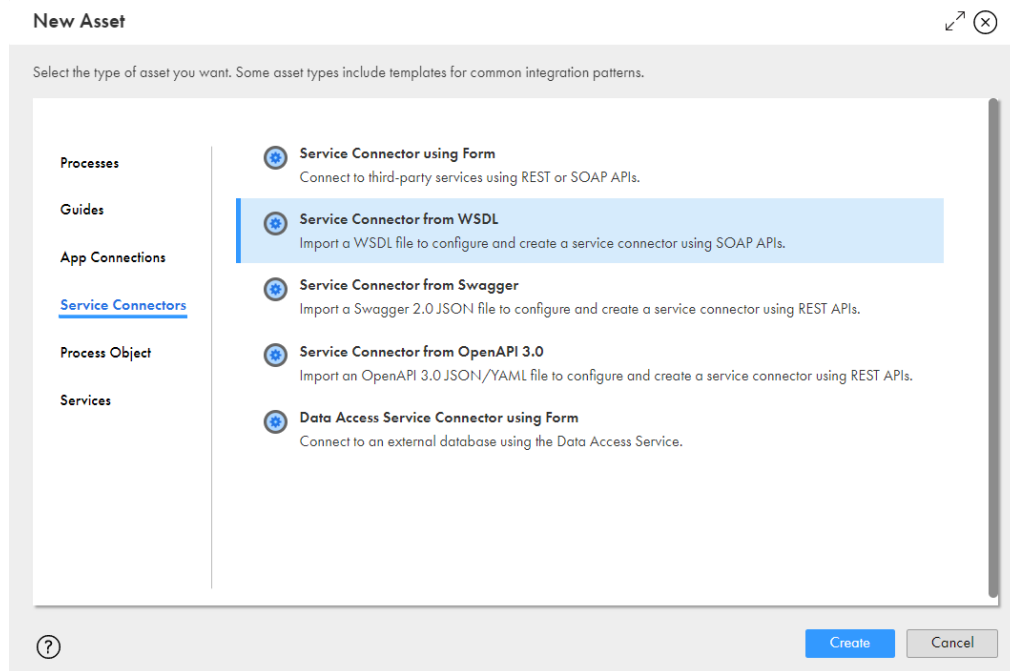
2. Click **Service Connectors** > **Service Connector from WSDL**, and then click **Create**.

   The **New Service Connector from WSDL** dialog box appears with the **WSDL Source** tab open as shown in the following image:
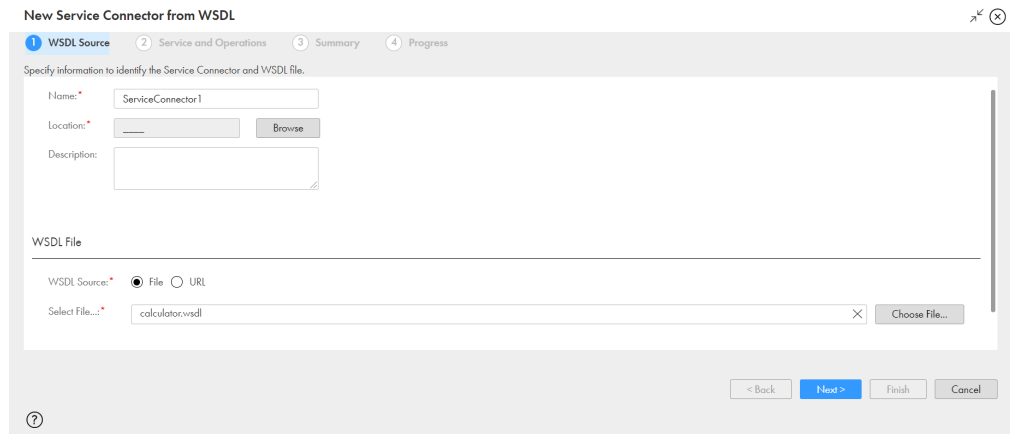
3. Specify a name, location, and description for the service connector.

4. In the **WSDL File** section, use one of the following options to specify the WSDL file that you want to import:

- To use a WSDL file from your local system, select **File**, and then select the WSDL file.
  You can either select a single `.wsdl` file or a single `.zip` file that contains self-contained `.wsdl` files and `.xsd` schema files.

- To use a URL that contains a WSDL file, select **URL**. The **WSDL URL** and **Use authentication** fields appear. In the **WSDL URL** field, enter the URL that contains the WSDL file. Select the **Use authentication** option to specify the user name and password to access the WSDL URL.

  Note: If the password contains an ampersand (&) character, when you test the service connector, the action fails. Additionally, if you use the same service connector in an app connection and process, the process invocation also fails.

5. Click **Next**.

   The **Services and Operations** tab appears.

   The following image shows the **Services and Operations** tab:
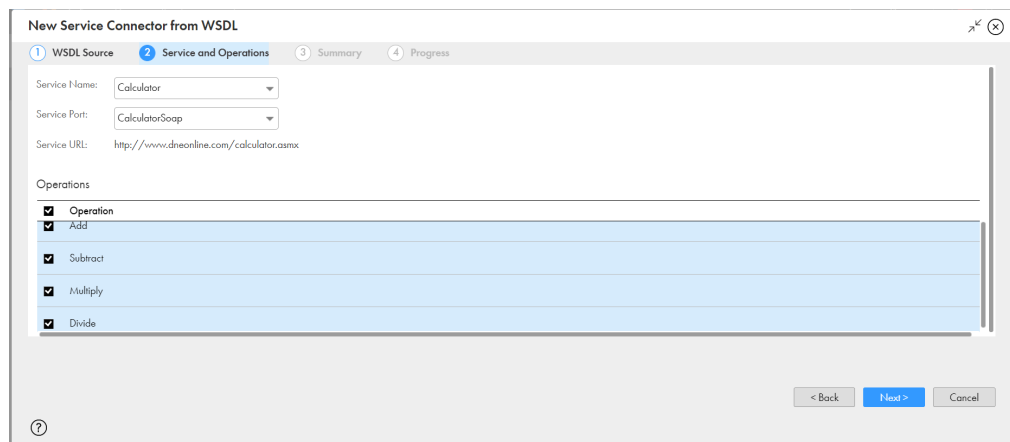
6. From the **Service Name** list, select the service that you want to use from the imported WSDL file.

   The **Service URL** field displays the URL to which the service connector sends requests. You cannot edit this URL.

7. From the **Service Port** list, select the port that you want the service connector to use.

8. In the **Operations** section, select the operations that you want the service connector to provide.

   The operations that you select will be available as actions when you create a process and use the service connector.

9. Click **Next**.

   The **Summary** tab appears.

The following image shows the **Summary** tab:
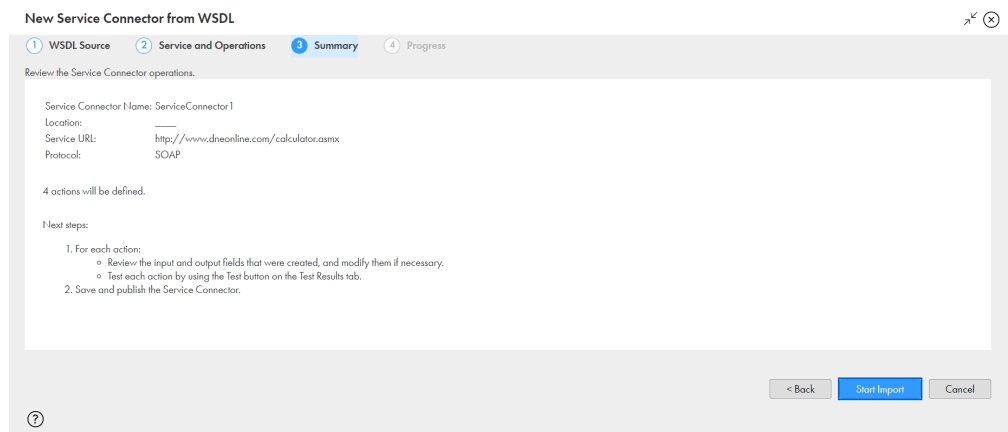
10. Review the service connector configuration.

11. Click **Start Import**.

Application Integration starts importing the WSDL file and displays the status on the **Progress** tab. The **Progress** tab indicates whether the import completed successfully, failed, or completed with warnings.

The following image shows the **Progress** tab:
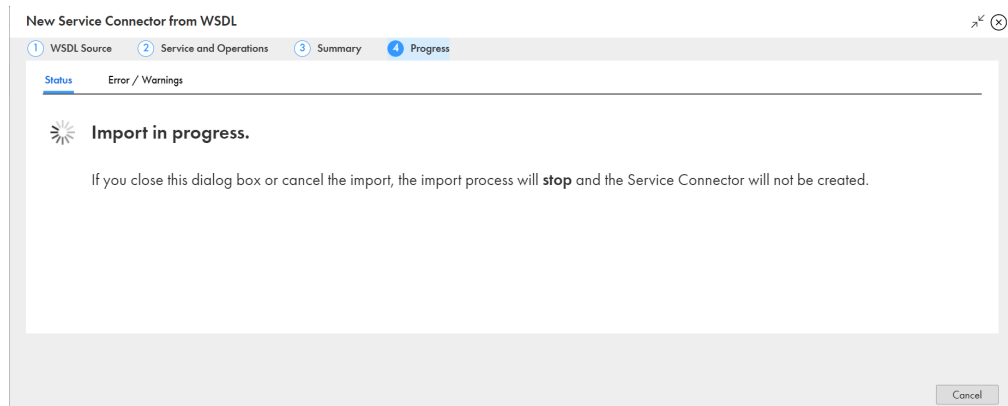
Note: If you close the dialog box or cancel the import, the import process stops and the service connector is not created.

If the import fails or completes with warnings, click the **Error/Warnings** tab to understand why the import did not complete successfully. You can fix the issues and import the WSDL file again.

The following image shows a successful import:
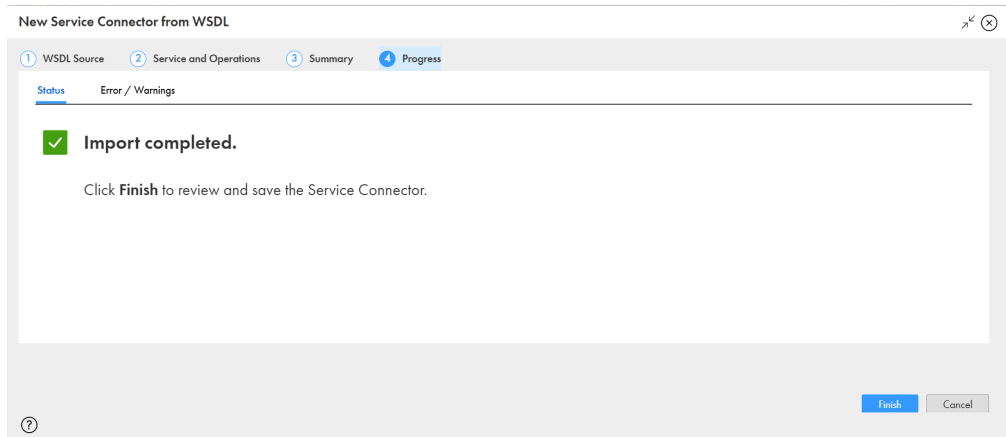
12. Click **Finish** to review and save the service connector.

WSDL Files with Qualified Elements

If you have a WSDL file with user defined data types, you can add namespace prefixes and type names to create a WSDL file with qualified elements. You can use this WSDL file to create a service connector.

You can create a service connector from a WSDL file that contains user defined data types. Add namespace prefixes and type names to the user defined data to create a WSDL file with qualified elements. You can use this WSDL file to generate a service connector. The namespace prefix does not appear in the process object and you see a correct payload.

For example, consider a WSDL file that has the user defined elements RecordId and AddressComplete.

First, define a schema with the type name 'Address'. The following is a sample schema with a complexType name Address:

```
xs:complexType name="Address">
    <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="1" name="RecordId" type="xs:string" />
        <xs:element minOccurs="0" maxOccurs="1" name="AddressComplete"
type="xs:string" />
    </xs:sequence>
</xs:complexType>
<xs:element minOccurs="0" maxOccurs="1" name="Address" type="tns:Address" />
```

Next, in the WSDL file, add the namespace prefix 'tns', as in the following sample:

```
<wsdl:message name="InputRequest">
    <wsdl:part name="parameters" element="tns:Address" />
</wsdl:message>
<wsdl:portType name="AddressValidationSoap">
    <wsdl:operation name="Process">
      <wsdl:input message="tns:InputRequest" />
      ...
    </wsdl:operation>
</wsdl:portType>
```

You can use this WSDL file to create a service connector. You will get an Address process object with two fields, RecordId and AddressComplete.

WSDL Files with Repeating Elements

You can create a service connector from a WSDL file with nested and repeating elements.

For example, consider the following WSDL and WSDL schema files with the nested, repeating elements 'shelf' and 'book':

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:libSchema="www.library.com/library.xsd"
    targetNamespace="www.library.com/library.wsdl"
    xmlns:lib="www.library.com/library.wsdl"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    name="library">

    <wsdl:types>
        <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:import
                namespace="www.library.com/library.xsd"
                schemaLocation="./schema/library.xsd">
            </xsd:import>
        </xsd:schema>
    </wsdl:types>

    <wsdl:message name="request">
        <wsdl:part name="request" element="libSchema:library" />
    </wsdl:message>
    <wsdl:message name="response">
        <wsdl:part name="response" element="libSchema:bookCount" />
    </wsdl:message>

    <wsdl:portType name="Library">
        <wsdl:operation name="countBooks">
            <wsdl:input message="lib:request" />
            <wsdl:output message="lib:response"></wsdl:output>
        </wsdl:operation>
    </wsdl:portType>

</wsdl:definitions>
```

```
1    <?xml version="1.0"?>
2    <xs:schema elementFormDefault="qualified"
3         targetNamespace="www.library.com/library.xsd"
4         xmlns:xs="http://www.w3.org/2001/XMLSchema"
5         xmlns:ns="www.library.com/library.xsd">
6
7        <xs:element name="book">
8            <xs:complexType>
9                <xs:attribute name="name" type="xs:string" use="required" />
10           </xs:complexType>
11       </xs:element>
12       <xs:element name="shelf">
13           <xs:complexType>
14               <xs:sequence>
15                   <xs:element ref="ns:book" minOccurs="0" maxOccurs="100" />
16               </xs:sequence>
17               <xs:attribute name="name" type="xs:string" use="required" />
18           </xs:complexType>
19       </xs:element>
20       <xs:element name="library">
21           <xs:complexType>
22               <xs:sequence>
23                   <xs:element ref="ns:shelf" minOccurs="0" maxOccurs="20" />
24               </xs:sequence>
25               <xs:attribute name="name" type="xs:string" use="required" />
26           </xs:complexType>
27       </xs:element>
28
29       <xs:element name="bookCount" type="xs:integer" />
30
31   </xs:schema>
```

You can use this WSDL and WSDL schema file to create a service connector using the WSDL to Service Connector tool. When you import the service connector, you see three process objects: Library, Shelf, and Book. When you select a parent object, you also see the nested object in the right side of the bottom panel.

elementFormDefault

You can use WSDL files that contain the elementFormDefault attribute in the WSDL schema.

The WSDL file can contain elementFormDefault="qualified" or elementFormDefault="unqualified".

Choice Elements

You can use choice schema elements to create a payload in a WSDL file.

Use the If element to show what should appear in the request.

The WSDL to Service Connector considers top level choice elements. This means that if a WSDL schema file has nested choice elements, the WSDL to Service Connector takes the first choice element.

Creating a Service Connector from a Swagger JSON File

You can import a Swagger 2.0 JSON file to easily create a service connector with multiple operations. Use the **New Service Connector from Swagger** wizard to import a Swagger JSON file and create a service

connector. After you specify the Swagger JSON file, Application Integration parses the file and loads the operations.

If an error occurs during import, the wizard displays the error under **Error/Warnings** on the **Progress** tab. You can take corrective actions and import the Swagger JSON file again.

1. In Application Integration, click **New** on the left navigation bar.

   The **New Asset** dialog box appears.

   The following image shows the **New Asset** dialog box:
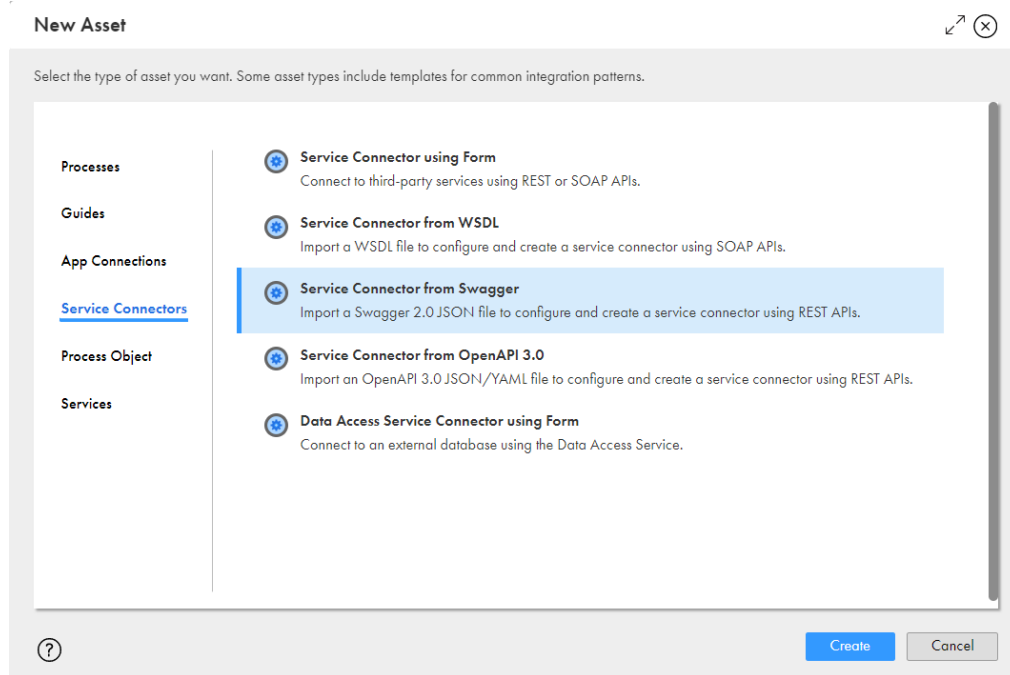
2. Click **Service Connectors** > **Service Connector from Swagger**, and then click **Create**.

   The **New Service Connector from Swagger** dialog box appears with the **Swagger Source** tab open as shown in the following image:
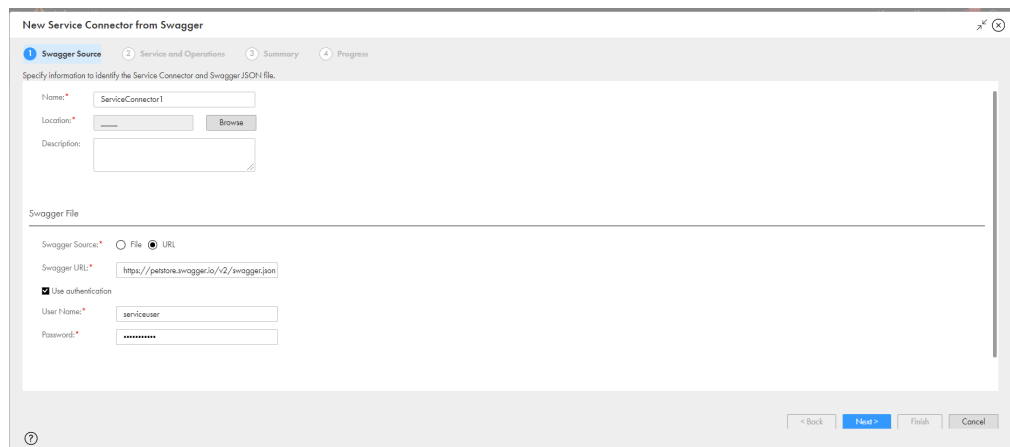
3. Specify a name, location, and description for the service connector.

4. In the **Swagger File** section, use one of the following options to specify the Swagger JSON file that you want to import:

- To use a Swagger JSON file from your local system, select **File**, and then select the Swagger JSON file.

- To use a URL that contains a Swagger JSON file, select **URL**. The **Swagger URL** and **Use authentication** fields appear. In the **Swagger URL** field, enter the URL that contains the Swagger JSON file. Select the **Use authentication** option to specify the user name and password to access the Swagger JSON URL.

  Note: If the password contains an ampersand (&) character, when you test the service connector, the action fails. Additionally, if you use the same service connector in an app connection and process, the process invocation also fails.

5. Click **Next**.

   The **Services and Operations** tab appears.

   The following image shows the **Services and Operations** tab:
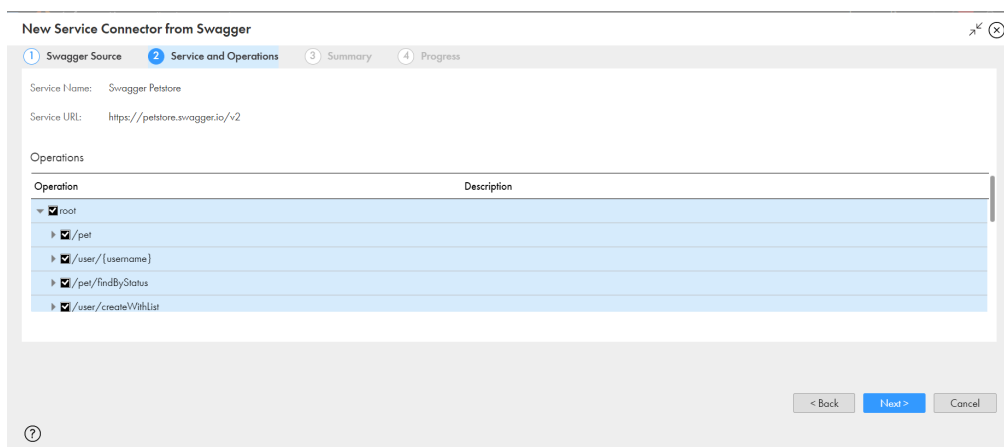
6. In the **Operations** section, select the operations that you want the service connector to provide.

   The operations that you select will be available as actions when you create a process and use the service connector.

7. Click **Next**.

   The **Summary** tab appears.

   The following image shows the **Summary** tab:
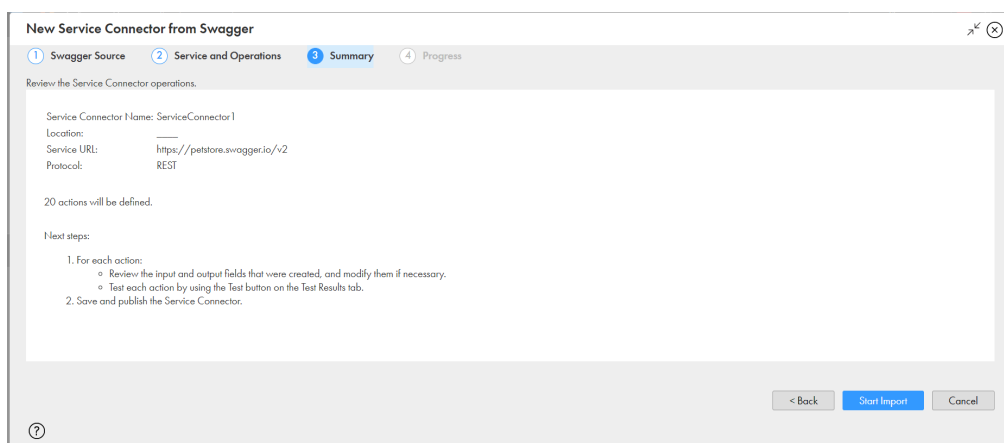
8. Review the service connector configuration.

9. Click **Start Import**.

    Application Integration starts importing the Swagger JSON file and displays the status on the **Progress** tab. The **Progress** tab indicates whether the import completed successfully, failed, or completed with warnings.

    The following image shows the **Progress** tab:
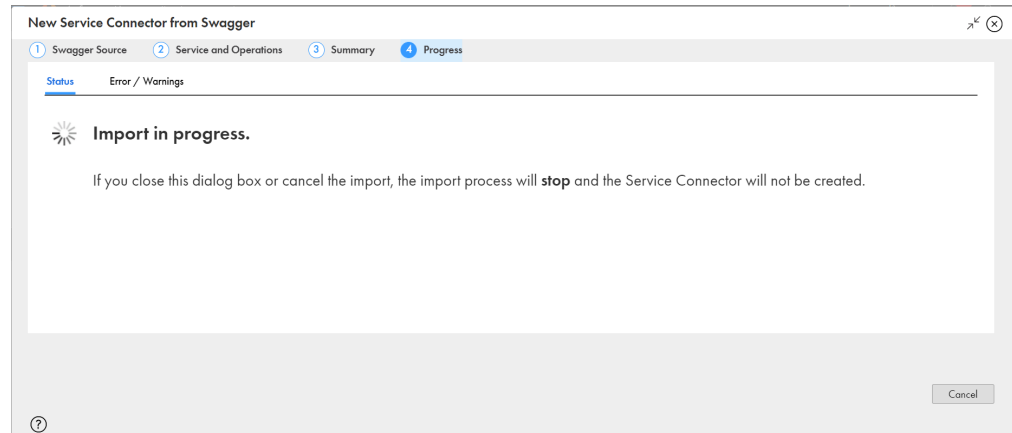
    Note: If you close the dialog box or cancel the import, the import process stops and the service connector is not created.

    If the import fails or completes with warnings, click the **Error/Warnings** tab to understand why the import did not complete successfully. You can fix the issues and import the Swagger JSON file again.

    The following image shows a successful import:
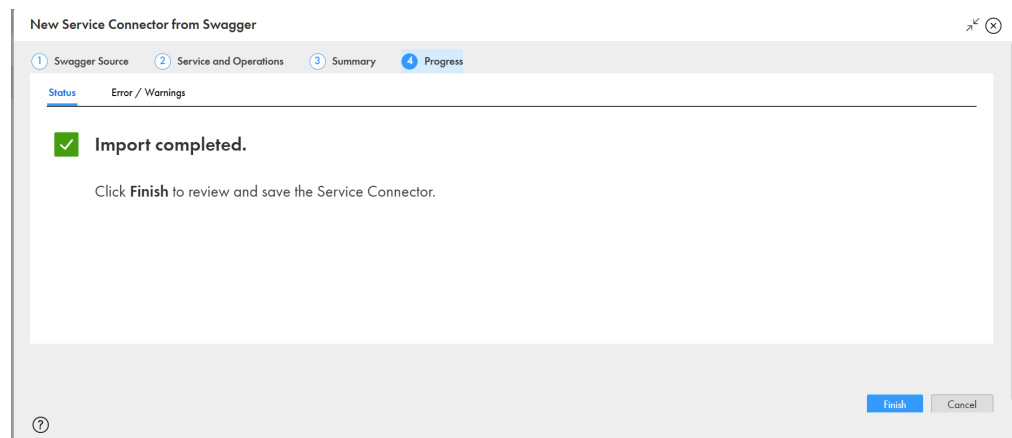
10. Click **Finish** to review and save the service connector.

Rules and guidelines for Swagger files

Consider the following rules and guidelines when you import a Swagger file to create a service connector:

*   You must import a valid Swagger file. You can use the following URL to validate the syntax of the Swagger file:
    https://editor.swagger.io/
*   Application Integration supports Swagger files of the JSON format. It does not support other formats such as YAML.

- Application Integration supports only single-document Swagger files. If you import a multi-document Swagger file, Application Integration does not load the dependencies.

- If you create a service connector by importing a Swagger file that has a large number of complex types and operations, the service connector editor might be affected, because the generated service connector will be complex and bulky in size.

- If you have a large Swagger file to create a service connector with multiple types and operations, you can split the Swagger file into several smaller Swagger files for improved performance and better usability. You can use a Swagger splitter tool to split the Swagger file. For more information, see the following community article:
  https://knowledge.informatica.com/s/article/DOC-18606

  You can also manually select a subset of operations in the import wizard to create service connectors with fewer actions.

Creating a Service Connector from a OpenAPI 3.0 File

You can import an OpenAPI 3.0 JSON or YAML file to easily create a service connector with multiple operations. Use the **New Service Connector from OpenAPI 3.0** wizard to import a OpenAPI 3.0 file and create a service connector. After you specify the OpenAPI 3.0 file, Application Integration parses the file and loads the operations.

If an error occurs during import, the wizard displays the error under **Error/Warnings** on the **Progress** tab. You can take corrective actions and import the OpenAPI 3.0 file again.

1. In Application Integration, click **New** on the left navigation bar.

   The **New Asset** dialog box appears.

   The following image shows the **New Asset** dialog box:
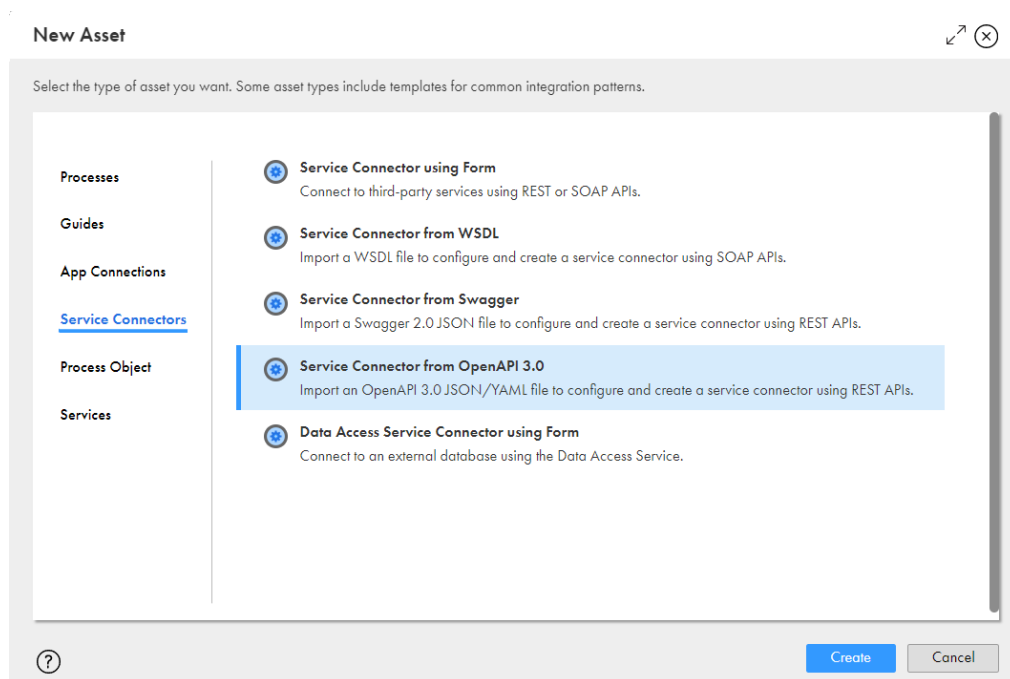
2. Click **Service Connectors** > **Service Connector from OpenAPI 3.0**, and then click **Create**.

The **New Service Connector from OpenAPI 3.0** dialog box appears with the **OpenAPI 3.0 Source** tab open as shown in the following image:

3. Specify a name, location, and description for the service connector.

4. In the **OpenAPI 3.0 File** section, use one of the following options to specify the OpenAPI 3.0 file that you want to import:

   • To use an OpenAPI 3.0 file from your local system, select **File**, and then select the OpenAPI 3.0 JSON or YAML file.
   You can either select a single `.json` or `.yaml` file, or a single `.zip` file that contains self-contained `.json` or `.yaml` files.

   • To use a URL that contains an OpenAPI 3.0 file, select **URL**. The **OpenAPI 3.0 URL** and **Use authentication** fields appear. In the **OpenAPI 3.0 URL** field, enter the URL that contains the OpenAPI 3.0 JSON or YAML file. Select the **Use authentication** option to specify the user name and password to access the OpenAPI 3.0 URL.

   Note: If the password contains an ampersand (&) character, when you test the service connector, the action fails. Additionally, if you use the same service connector in an app connection and process, the process invocation also fails.

5. Click **Next**.

   The **Service and Operations** tab appears.

   The following image shows the **Service and Operations** tab:
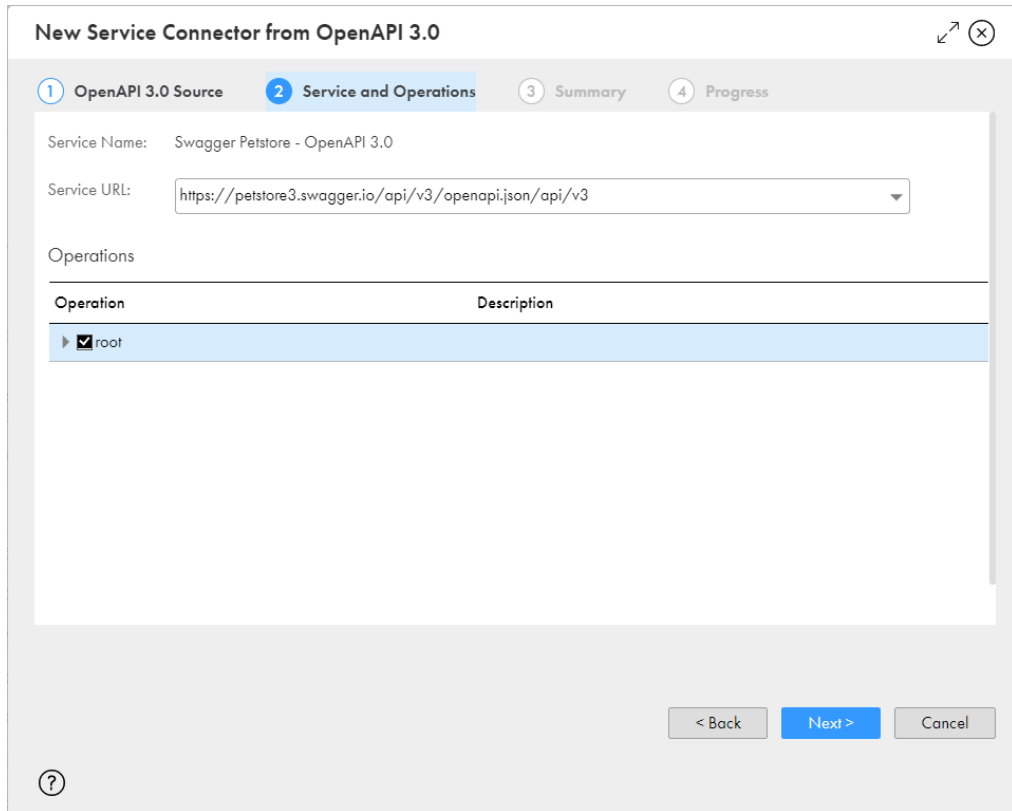
6. From the **Service URL** list, select the endpoint that you want to use from the imported OpenAPI 3.0 file to create a service connector.

7. In the **Operations** section, select the operations that you want the service connector to provide.

    The operations that you select will be available as actions when you create a process and use the service connector.

8. Click **Next**.

    The **Summary** tab appears.

The following image shows the **Summary** tab:
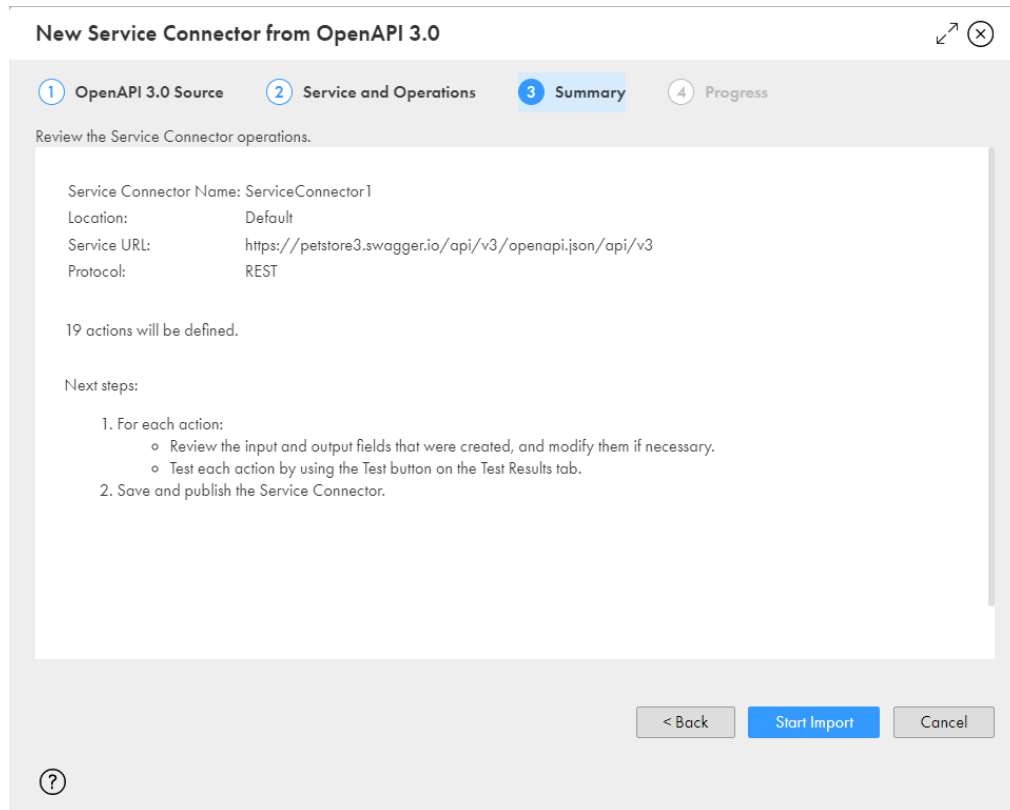
9. Review the service connector configuration.

10. Click **Start Import**.

Application Integration starts importing the OpenAPI 3.0 file and displays the status on the **Progress** tab. The **Progress** tab indicates whether the import completed successfully, failed, or completed with warnings.

The following image shows the **Progress** tab:
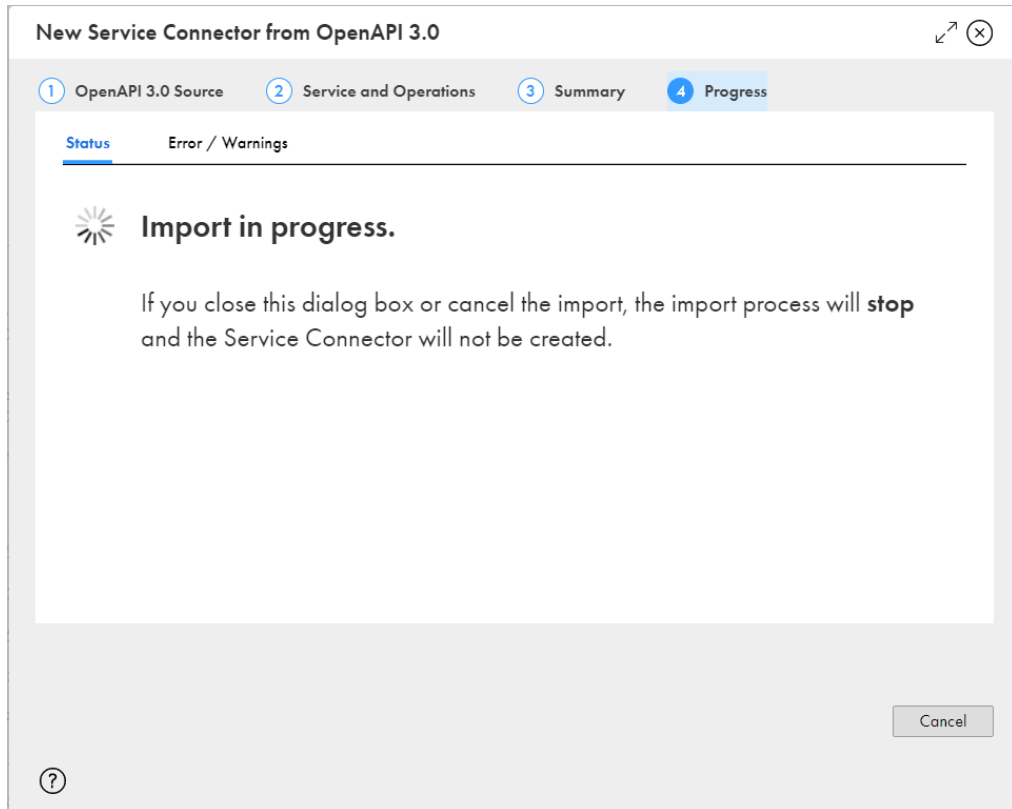
Note: If you close the dialog box or cancel the import, the import process stops and the service connector is not created.

If the import fails or completes with warnings, click the **Error/Warnings** tab to understand why the import did not complete successfully. You can fix the issues and import the OpenAPI 3.0 file again.

11. Click **Finish** to save the service connector.

HTTP Operations

You can import a Swagger file that contains the DELETE, PUT, POST, PATCH, and GET HTTP operations.

The service connector that you generate has actions corresponding to the HTTP operation is the Swagger file. If the Swagger file has multiple HTTP operations, the service connector will have multiple actions.

For example, if you use a Swagger file with a POST HTTP operation, you get a service connector with a POST action.

The following image shows a single POST HTTP operation within a Swagger file:

```
- paths: {
    - /pet: {
        - post: {
            - tags: [
                "pet"
            ],
            summary: "Add a new pet to the store",
            description: "",
            operationId: "addPet",
            - consumes: [
                "application/json",
                "application/xml"
            ],
            - produces: [
                "application/xml",
                "application/json"
            ],
            - parameters: [
                - {
                    in: "body",
                    name: "body",
                    description: "Pet object that needs to be added to the store",
                    required: true,
                    - schema: {
                        $ref: "#/definitions/Pet"
                    }
                }
            ],
            - responses: {
                - 405: {
                    description: "Invalid input"
                }
            },
            - security: [
                - {
                    - petstore_auth: [
                        "write:pets",
                        "read:pets"
                    ]
                }
            ]
        },
    }
```

The following image shows the action in service connector that you get when you use the preceding Swagger file:

You see that the Add Pet action corresponds to the verb POST.

JSON Payload

You can import a Swagger file with different types of JSON payloads. The Service Connector you generate reflects the JSON payload of the Swagger file.

The following table shows the JSON payload features that you can use, a sample Swagger excerpt, and the Input field of the corresponding service connector:

JSON Payload Feature	Sample Swagger Content	Service Connector Input Field

Lists of Objects	```schema: {
	    type: "array",
	  - items: {
	        $ref: "#/definitions/User"
	    }
	}```
		Name* body, Label body, Type Reference / Process Object User

References	```schema: { $ref: "#/definitions/Pet" }```	Name* body, Label body, Type Reference / Process Object Pet

Single Select Enums	```items: {
	    type: "string",
	  - enum: [
	        "available",
	        "pending",
	        "sold"
	    ],
	    default: "available"
	},```
		Name* status, Label status, Type Multi-Select Picklist / Options: available, pending, sold

Simple types	```{
	    name: "orderId",
	    in: "path",
	    description: "ID of pet that needs to be fetched",
	    required: true,
	    type: "integer",
	    maximum: 10,
	    minimum: 1,
	    format: "int64"
	}```
		Name* orderId, Label orderId, Type Integer

Attachments	```- {
	    name: "file",
	    in: "formData",
	    description: "file to upload",
	    required: false,
	    type: "file"
	}```
		file, file, Attachment

Lists of simple types	```- parameters: [
	  - {
	        name: "tags",
	        in: "query",
	        description: "Tags to filter by",
	        required: true,
	        type: "array",
	      - items: {
	            type: "string"
	        },
	        collectionFormat: "multi"
	    }
	],```
		Name* tags, Label tags, Type Object List / Process Object string

JSON Payload Feature	Sample Swagger Content	Service Connector Input Field

Multi Select Enums	parameters: [
	- {
	    name: "status",
	    in: "query",
	    description: "Status values that need to be considered for filter",
	    required: true,
	    type: "array",
	    - items: {
	        type: "string",
	        - enum: {
	            "available",
	            "pending",
	            "sold"
	        },
	        default: "available"
	    },
	    collectionFormat: "multi"
	    }
	],
		Name* / Label / Type: Multi-Select Picklist ▾ — status / status / Options: available, pending, sold

Objects with References to Other Object Lists	tags: {
	    type: "array",
	    - xml: {
	        name: "tag",
	        wrapped: true
	    },
	    - items: {
	        $ref: "#/definitions/Tag"
	    }
	},
		Name* / Label / Type: Object List ▾ — tags / tags / Process Object: string ▾

Required and nullable elements

You can define process object fields as required and nullable in the WSDL, Swagger JSON, and OpenAPI 3.0 JSON or YAML files before you import the files to generate a service connector.

In a WSDL file, to define a field as nullable, set the `nillable` attribute to "`true`" in the field element. If you do not specify the `nillable` attribute, the field is set to not nullable by default. If you do not want to set a field as required, set the `minOccurs` attribute to "`0`" in the field element. If you do not specify the `minOccurs` attribute, the field is considered as required by default.

For example, consider a WSDL file that has a schema named ProcessObjectReq. The schema is of a complex type and contains two elements called as Name and ID. To set the ID field as nullable and not required, set the `nillable` attribute to "`true`" and the `minOccurs` attribute to "`0`" as shown in the following sample:

```
<xsd:complexType name="ProcessObjectReq">
    <xsd:sequence>
        <xsd:element name="Name" nillable="false" type="xsd:string"/>
        <xsd:element name="ID" minOccurs="0" nillable="true" type="xsd:integer"/>
    </xsd:sequence>
</xsd:complexType>
```

When you import the WSDL file, the nullable and required field elements defined in the WSDL file are propagated to the process object as shown in the following image:

Similarly, you can set the required and nullable field elements in the Swagger JSON file and create a service connector as shown in the following sample:

```
"definitions" : {
  "Process1-3Request" : {
    "type" : "object",
    "required" : [ "Test" ],
    "properties" : {
      "Test" : {
        "$ref" : "#/definitions"/ProcessObjectReq"
      }
    }
  },
  "ProcessObjectReq" : {
    "type" : "object",
    "properties" : {
      "Name" : {
        "type" : "string"
      },
      "ID" : {
        "type" : "integer",
        "format" : "int32"
      }
    },
    "required": [
      "Name"
    ],
  "x-nullable":["ID"]
  }
}
```

You can also set the required and nullable field elements in the OpenAPI 3.0 file and create a process object as shown in the following sample:

```
"ApiResponse": {
  "type": "object",
  "properties": {
    "code": {
      "type": "integer",
      "format": "int32",
      "required": true,
      "nullable": false
      },
    },
...
```

If you use a process object with nullable fields in a process, and generate the Swagger JSON file for the process, the generated Swagger JSON file might not contain the `x-nullable` field defined in the process object. The field is set to not nullable by default. Therefore, Informatica recommends that you use a WSDL file or OpenAPI 3.0 file to preserve the nullable field values.

The following image shows the required and nullable fields selection from the imported Swagger JSON file:
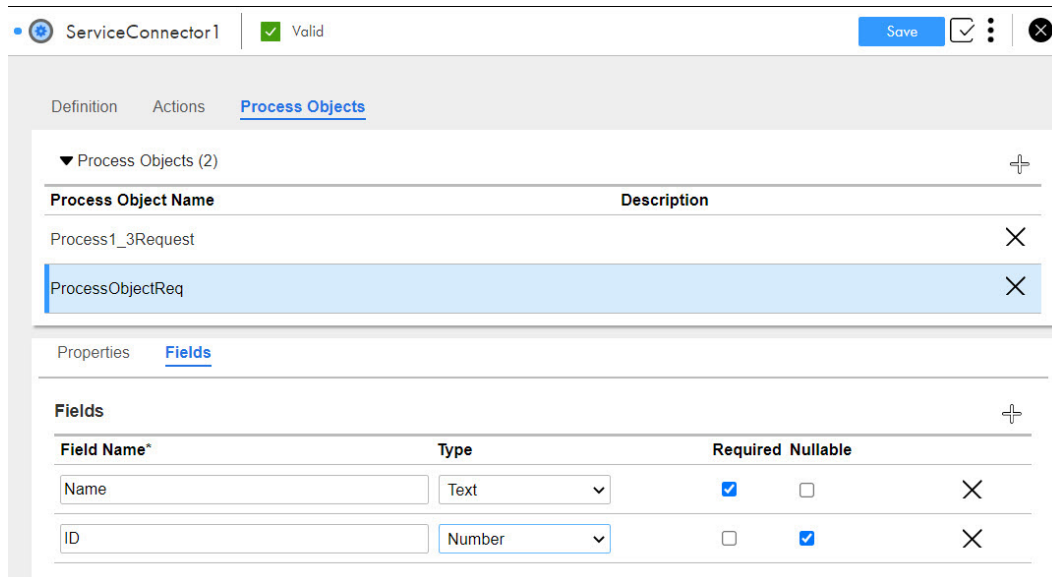
When you use this process object in a process, the fields marked as required and nullable use the same restrictions in the generated WSDL file, Swagger file, and OpenAPI 3.0 file.

Option 3. Downloading a Service Connector from GitHub

Informatica has uploaded several REST and SOAP API-based service connectors on GitHub. You can download the Informatica service connectors for free and import them into your organization. You can also upload your service connector files and contribute towards enriching the repository.

For more information about Informatica service connectors and downloadable process samples, click the following URL:
https://network.informatica.com/community/informatica-network/products/cloud-integration/cloud-

application-integration/blog/2018/10/16/registry-of-service-connectors-your-gateway-to-building-composite-api-using-cloud-application-integration

1. Access the following URL to download the service connectors:

    https://github.com/InformaticaCloudApplicationIntegration/Service-Connectors

    The following **Service-Connectors** page appears displaying the Informatica service connectors.
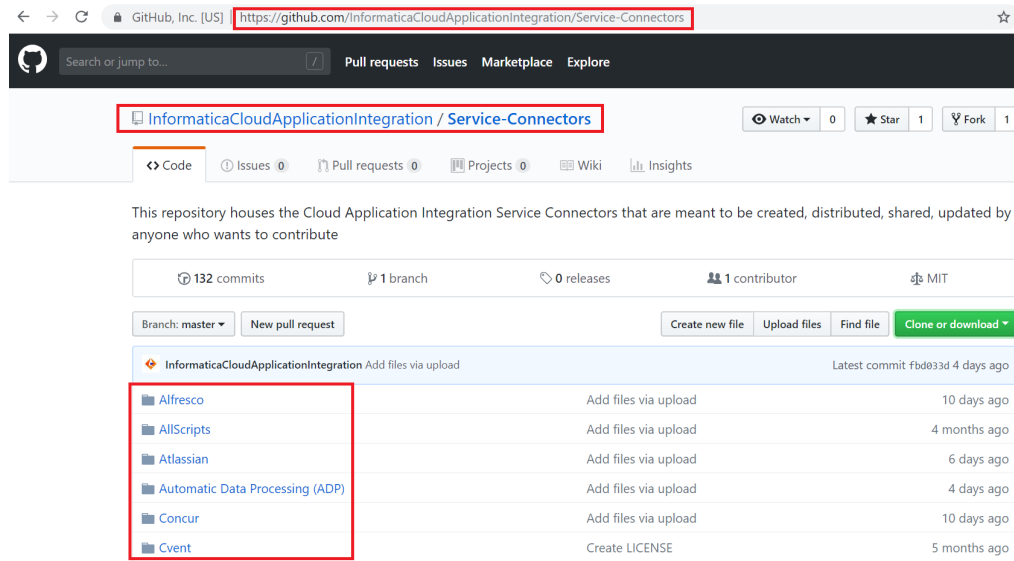
2. Download the service connector `.zip` file that you want to use.

3. Log in to Informatica Intelligent Cloud Services℠ and click **Application Integration**.

4. On the **Explore** page, click **Import** and select the service connector `.zip` file that you downloaded from GitHub.

    The service connector appears in the **Select Assets** section as shown in the following image:
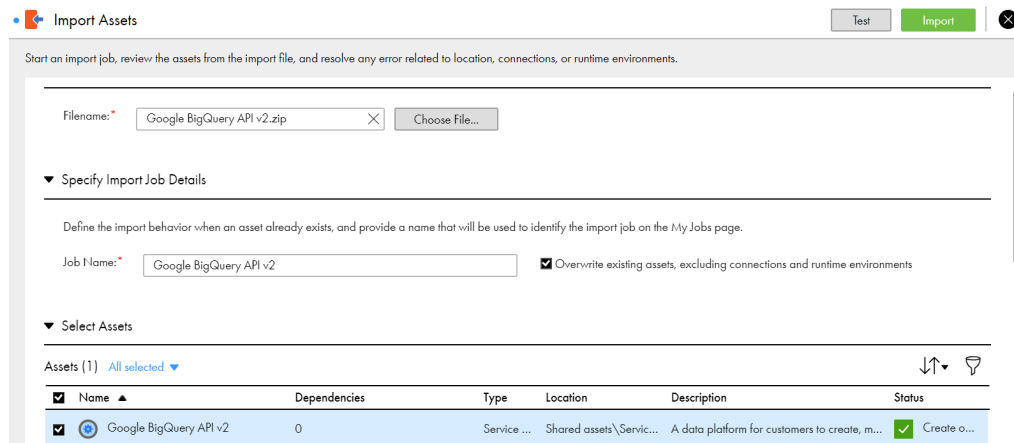

5. Click **Import** and open the **My Import/Export Logs** page to check the import status.

    After the import completes, the service connector is available in your organization in the following location:

    ```
    Shared assets\Service Connectors
    ```

Option 4. Downloading a Service Connector from Informatica Marketplace

Informatica has uploaded several REST and SOAP API-based service connectors on Informatica Marketplace. After you create an account in Informatica Marketplace, you can download the Informatica service connectors and import them into your organization.

1. Access the following URL: https://marketplace.informatica.com/

2. Create an account and log in.

3. From the **Product Categories** list on the left pane, click **Cloud Application Integration**.

   A page appears displaying the available service connectors.

   The following image shows some sample service connectors:
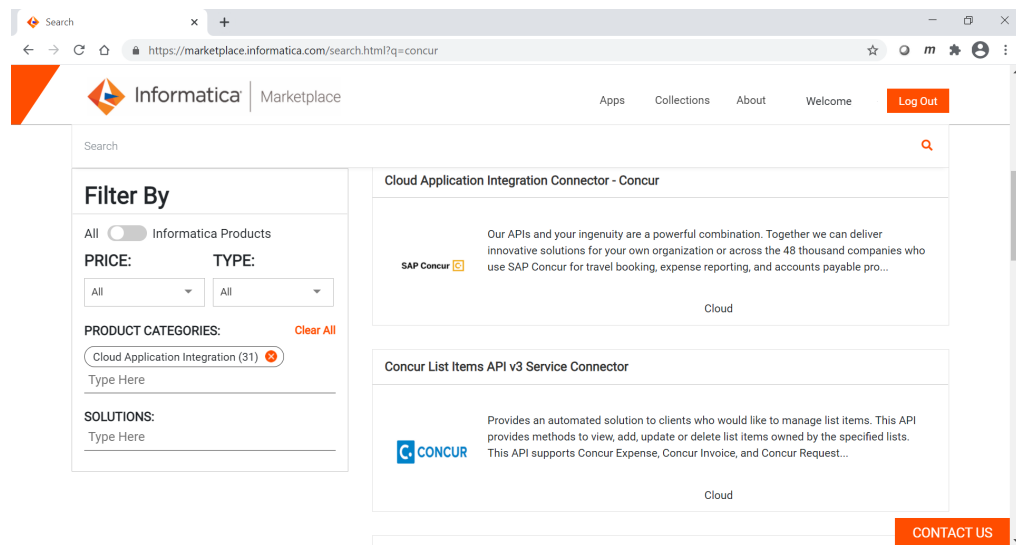
4. In the **Search** box, enter a name for the service connector that you want to download. You can also use the **Price** list and **Type** list to filter the service connectors.

5. Click the name of the service connector that you want to download.

   The service connector details appear.

   If the service connector contains specific API-based service connectors, you see all the connector details along with links to access the API documentation and API specification as shown in the following image:
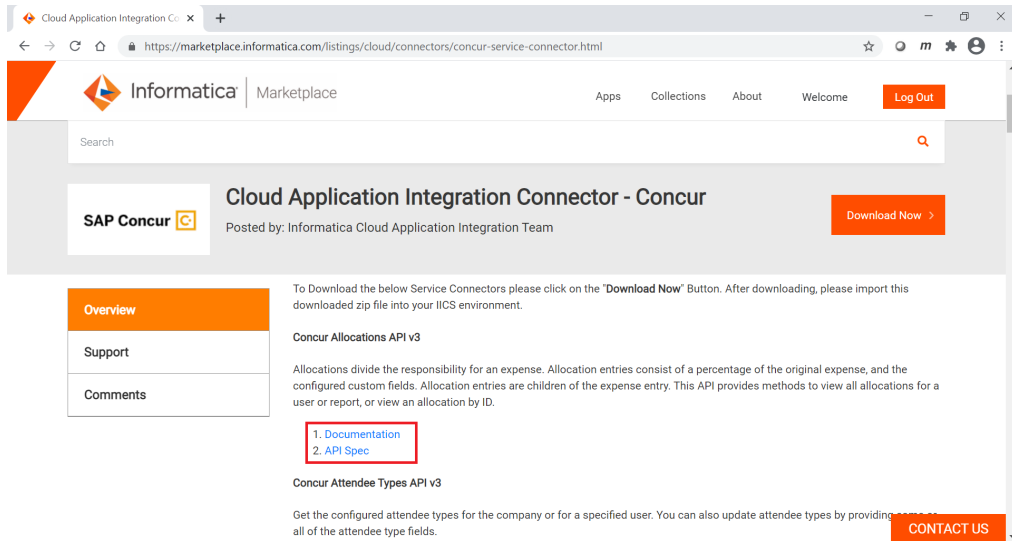
6. Click **Download Now**.

   A form appears prompting you to enter the user details.

7. Fill in the form, accept the terms and conditions, and submit the form.

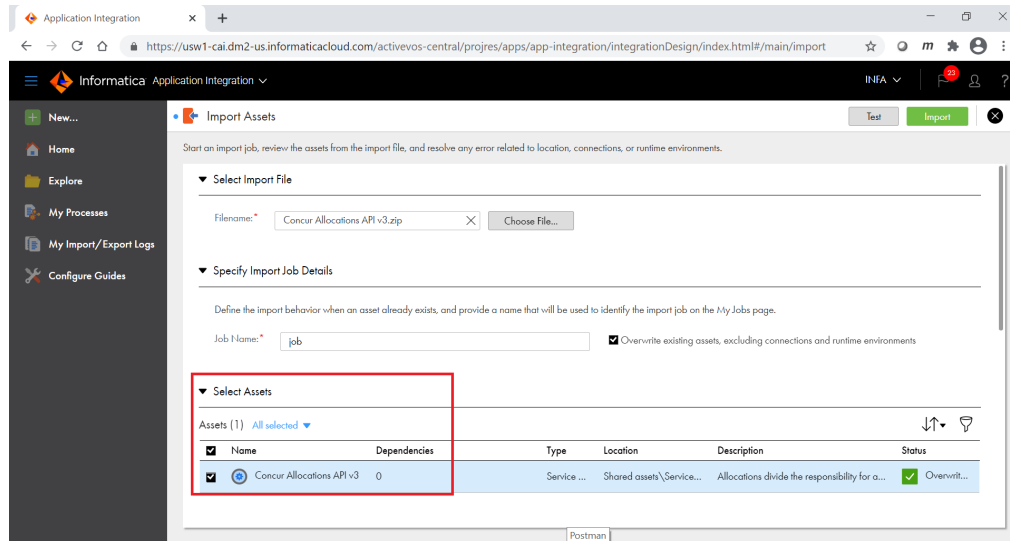   The service connector is downloaded.

   If the service connector contains specific API-based service connectors, you can choose to download specific connectors as shown in the following image:
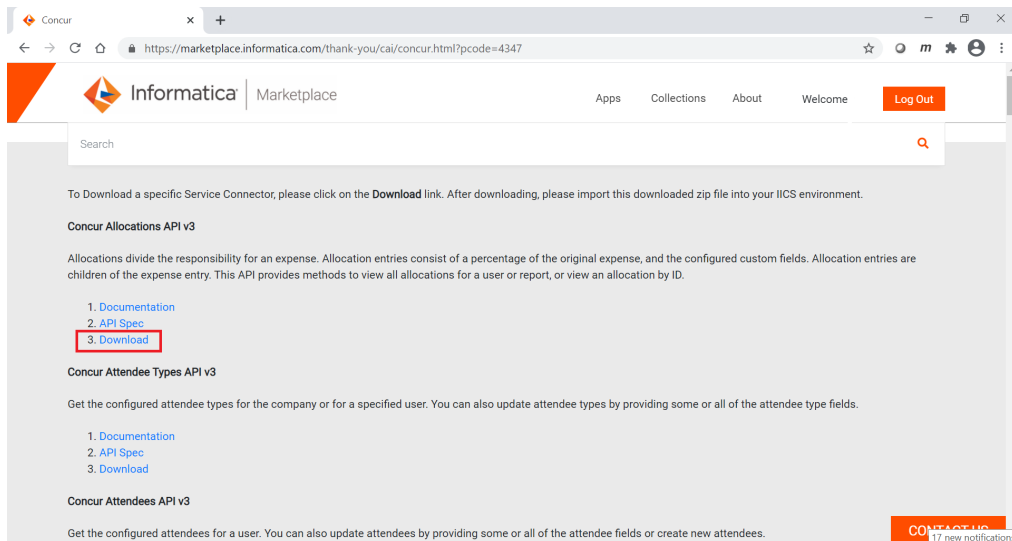
8. Log in to Informatica Intelligent Cloud Services and click **Application Integration**.

9. On the **Explore** page, click **Import** and select the service connector `.zip` file that you downloaded from Informatica Marketplace.

   The service connector appears in the **Select Assets** section as shown in the following image:
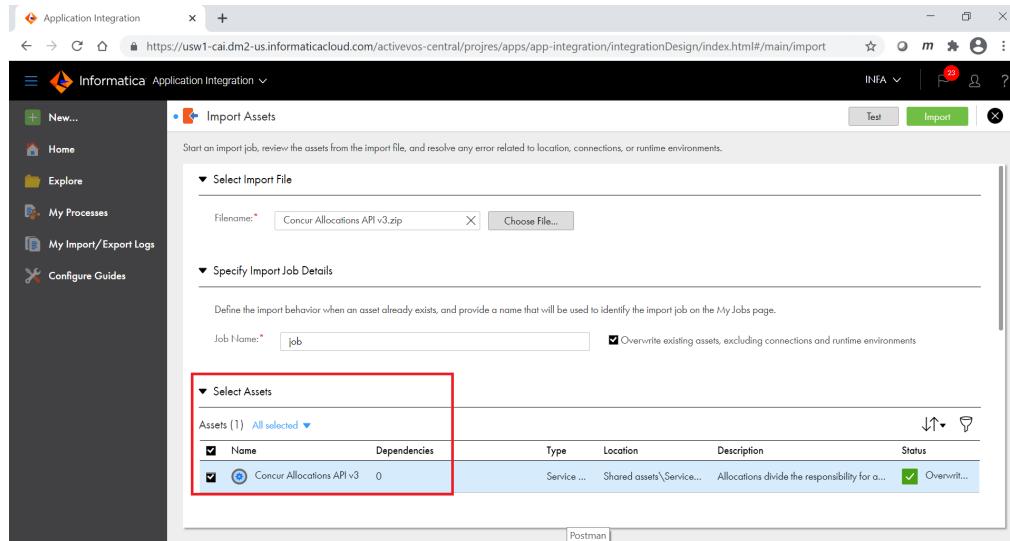
10. Click **Import** and open the **My Import/Export Logs** page to check the import status.

    After the import completes, the service connector is available in your organization in the following location:

    ```
    Shared assets\Service Connectors
    ```

Option 5. Downloading a Service Connector from Administrator

You can download REST and SOAP API-based service connectors from Informatica Intelligent Cloud Services Administrator and import them into your organization.

1. Log in to Informatica Intelligent Cloud Services and click **Administrator**.

2. On the left pane, click **Add-On Connectors**.

3. On the right pane, enter the search keyword as **cloud application integration connector** in the **Search** box.

   The Cloud Application Integration service connectors that you can download are displayed as shown in the following image:
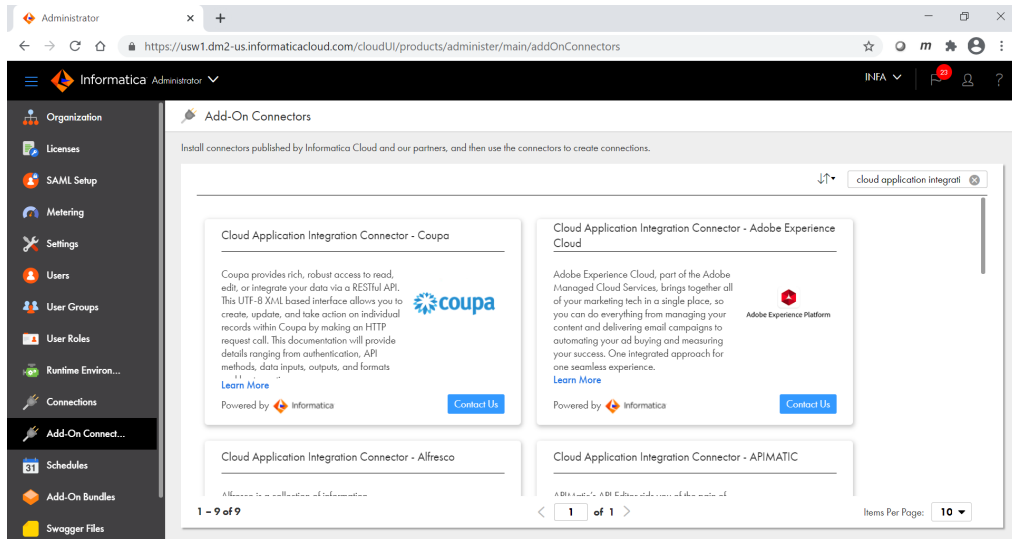
Note: The Cloud Application Integration service connectors are prefixed with the term **Cloud Application Integration Connector**.

4. Click **Contact Us** against the service connector that you want to download.

   The Informatica Marketplace page for the service connector opens.

5. Create an account and log in to Informatica Marketplace.

6. From the **Product Categories** list on the left pane, click **Cloud Application Integration**.

   A page appears displaying the available service connectors.

   The following image shows some sample service connectors:
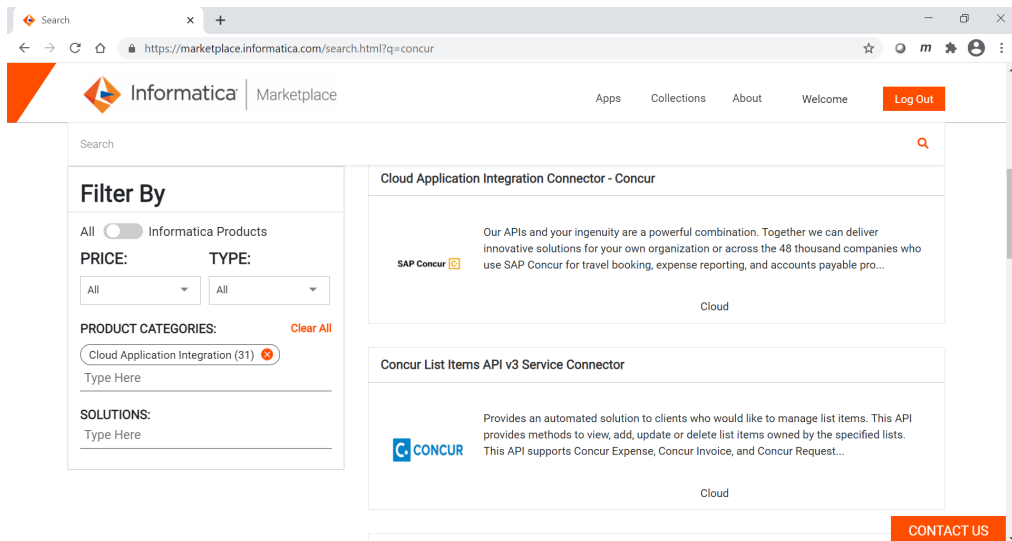
7. In the **Search** box, enter a name for the service connector that you want to download. You can also use the **Price** list and **Type** list to filter the service connectors.

8. Click the name of the service connector that you want to download.

   The service connector details appear.

If the service connector contains specific API-based service connectors, you see all the connector details along with links to access the API documentation and API specification as shown in the following image:
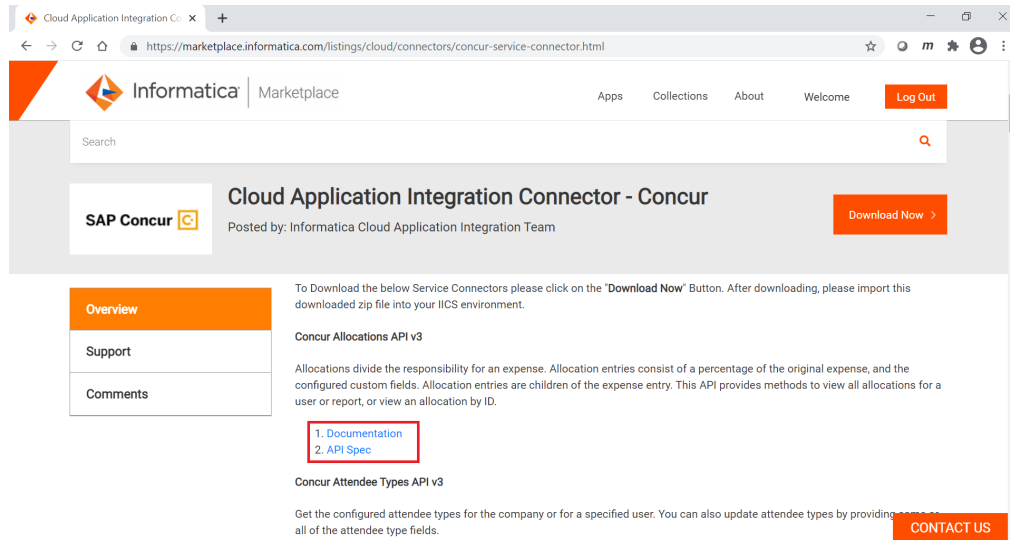
9. Click **Download Now**.

    A form appears prompting you to enter the user details.

10. Fill in the form, accept the terms and conditions, and submit the form.

    The service connector is downloaded.

    If the service connector contains specific API-based service connectors, you can choose to download specific connectors as shown in the following image:
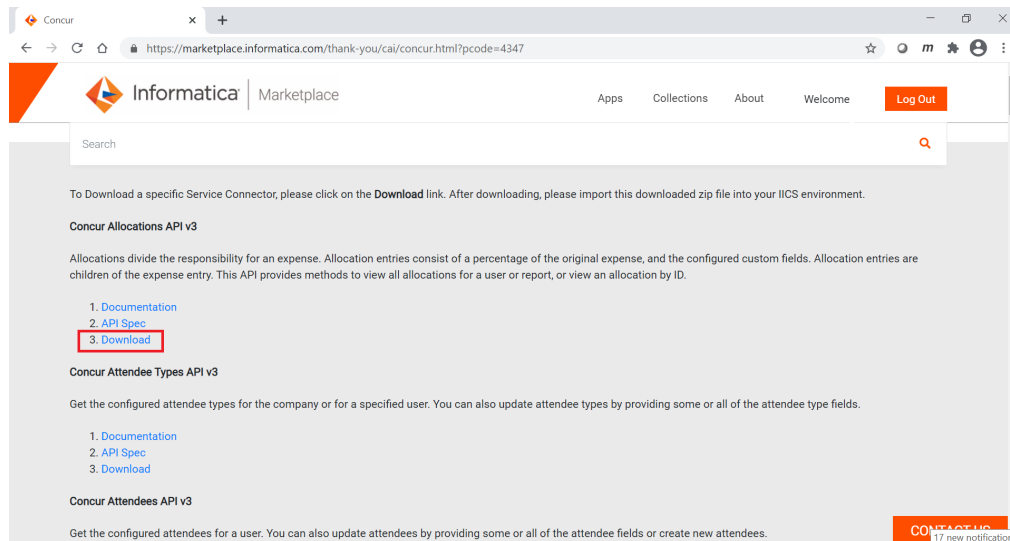
11. Log in to Informatica Intelligent Cloud Services and click **Application Integration**.

12. On the **Explore** page, click **Import** and select the service connector `.zip` file that you downloaded from Informatica Marketplace.

    The service connector appears in the **Select Assets** section as shown in the following image:

13. Click **Import** and open the **My Import/Export Logs** page to check the import status.

    After the import completes, the service connector is available in your organization in the following location:

    ```
    Shared assets\Service Connectors
    ```

Creating a Data Access Service Connector

You can create a data access service connector on a Secure Agent to access data directly from a database.

You can use a data access service connector to connect to the following databases:

- IBM DB2
- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL

You can create a data access service connector by defining the properties and specifying variables to interact directly with the database. You can also define the actions associated with a data access service connector.

Before you create a data access service connector, you must copy the required JDBC JAR files to the following location and restart the Secure Agent:

```
<Secure Agent installation directory>/apps/process-engine/ext
```

Download the following JDBC JAR files based on the database that you use:

```
Database name                    JAR files
IBM DB2                          db2jcc4.jar
MySQL                            mysql-connector-java-5.1.40-bin.jar
Oracle                           ojdbc8.jar
SQL Server                       sqljdbc41.jar
```

Note: The JAR files must be compatible with the database version that you use.

To create a data access service connector, perform the following steps:

1. In Application Integration, click **New** > **Service Connectors**.

2. In the **New Asset** dialog box, click **Data Access Service Connector using Form**, and then click **Create**.
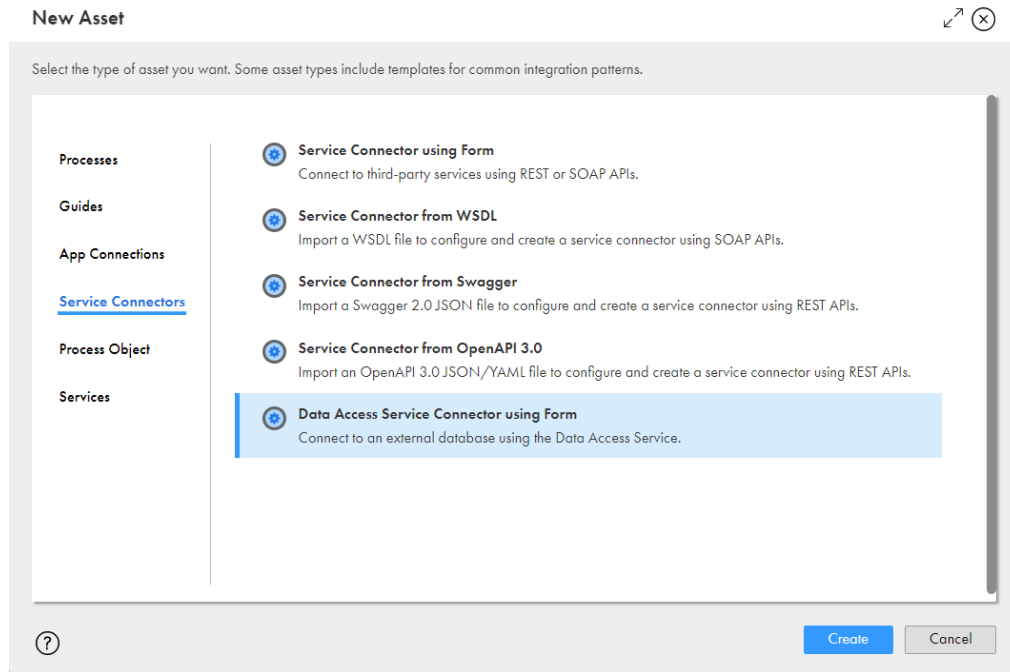
Defining Properties

To create a data access service connector and define the properties, perform the following steps:

1. In Application Integration, click **New** > **Service Connectors** > **Data Access Service Connector using Form** > **Create**.
   The data access service connector editor appears.

   The following image shows the data access service connector editor:

2. On the **Definition** tab, define the following basic properties for the data access service connector:

   - **Name**: The name by which the data access service connector is made available for processes. The name must start with a letter or number, and can contain only alphanumeric characters, multibyte characters, underscores (_), and hyphens (-). The name must not exceed 128 characters. This is a required field.

   - **Location**: Specify the project or folder to save the data access service connector. This is a required field.

- **Description**: Optionally, enter a description for the data access service connector.
- **Agent Only**: Selected by default because a data access service connector can only run on a Secure Agent.
- **Use OData**: Select this option to access data from a web service that uses OData. After you enable the **Use OData** option in a data access service connector, you can enable OData and specify the allowed users and groups for OData in the app connection that uses the data access service connections.

For more information about properties related to app connections, see *Design*

After you publish the app connection, you can view the OData Service URL and OData Swagger URL in the **Properties Detail** page of the app connection.

3. In the **Connection Properties** section, define the following general properties to connect to an external database:
   - **JDBC Driver**: The fully qualified Java class name of the JDBC driver.
     Based on the database, you can specify one of the following driver class names:
     - **IBM DB2**: com.ibm.db2.jcc.DB2Driver
     - **Microsoft SQL Server**: com.microsoft.sqlserver.jdbc.SQLServerDriver
     - **MySQL**: com.mysql.jdbc.Driver
     - **Oracle**: oracle.jdbc.OracleDriver
     - **PostgreSQL**: org.postgresql.Driver

     This is a required field.
   - **JDBC URL**: The connection URL to connect to an external database.
     Based on the database, you can specify one of the following URLs:
     - **IBM DB2**: jdbc:db2://<server>:<port>/<database>
     - **Microsoft SQL Server**: jdbc:sqlserver://<host>:<port>;databaseName=<database>
     - **MySQL**: jdbc:mysql://<host>:<port>/<database>
     - **Oracle**: jdbc:oracle:thin:@//<host>:<port>/<service>
     - **PostgreSQL**: jdbc:postgresql://<host>:<port>/<database>

     This is a required field.
- **User Name**: The user name to connect to the database. This is a required field.
- **Password**: The password to connect to the database. This is a required field. The value entered in this field is encrypted by default. In the Developer Console, the password that you enter is visible when you save the connector. After you save, close, and reopen the connector, the password appears encrypted.
- **Schema**: Enter the name of the schema that contains the tables that you want to include or exclude in the metadata. To view the appropriate results, you must also enter a schema name in the app connection that uses the data access service connector.
- **Include Tables**: Enter the names of the tables that you want to include in the metadata. To include a list of tables, use a comma to separate multiple table names. You can also use '.*' for pattern matching. To view the appropriate results, you must also mention the included table names in the app connection that uses the data access service connector.
- **Exclude Tables**: Enter the names of the tables that you want to exclude from the metadata. To exclude a list of tables, use a comma to separate multiple table names. You can also use '.*' for pattern matching. To view the appropriate results, you must also mention the excluded table names in the app connection that uses the data access service connector.

4. Optionally, in the **Connection Properties** section, define the following advanced properties :

- **Check SQL Injection**: Set this field to **Yes** to prevent or mitigate an SQL injection attack that can happen through malicious input parameters. Default is **No**. You must also configure this field in the app connection that uses the data access service connector.

- **Initial Connection Pool Size**: The initial number of connections created when the pool is started. Default is 5.

- **Max Active**: The maximum number of active connections that can be allocated from the pool simultaneously. If the value is negative, there is no limit. Default is 10.

- **Min Idle**: The minimum number of connections that can remain idle in the pool without additional connections being created. A value of 0 means that none will be created. Default is 1.

- **Max Idle**: The maximum number of connections that can remain idle in the pool without additional connections being released. If the value is negative, there is no limit. Default is 5.

- **Max Wait**: The maximum number of milliseconds that the pool waits for a connection if there are none available. A value of -1 means to wait indefinitely.

- **Transaction Isolation Level**: The default state of connections created by the pool, which is one of the following states:

  - 0 - None

  - 1 - Read Uncommitted

  - 2 - Read Committed

  - 4 - Repeatable Read

  - 8 - Serializable

- **Min Evictable Idle Time**: The minimum amount of time an object may sit idle in the pool before it is evicted (if an evictor exists).

- **Num Tests per Eviction Run**: The number of objects to test during each run of the idle object evictor thread (if an evictor exists).

- **Test on Borrow**: When selected, objects are validated before being borrowed from the pool. If the object cannot be validated, it is dropped from the pool, and an attempt is made to borrow another. Default is false.

- **Test on Return**: When selected, the borrowed objects are validated before being returned to the pool. Default is false.

- **Test while Idle**: When selected, the borrowed objects are validated by the idle object evictor (if one exists). If the object cannot be validated, it is dropped from the pool. Default is false.

- **Validation Query**: The SQL query that validates connections from the pool before returning them to the caller. If specified, the query must be an SQL SELECT statement that returns at least one row.

Specifying Variables

To configure a data access service connector, you might need to specify variables to define output fields.

Built-in Variables

You can reference the following variables in a data access service connector using an XQuery expression:

Variable	Variable Type	Description
$VariableName	All connection properties	Input and output fields can be specified using this format.
$Response	Output field mapping	Contains the Response XML data. It is displayed on the **Test Results** tab of the data access service connector.

Defining Actions

On the **Actions** tab, you can create and describe one or more actions associated with a data access service connector.

The following image shows the **Actions** tab:
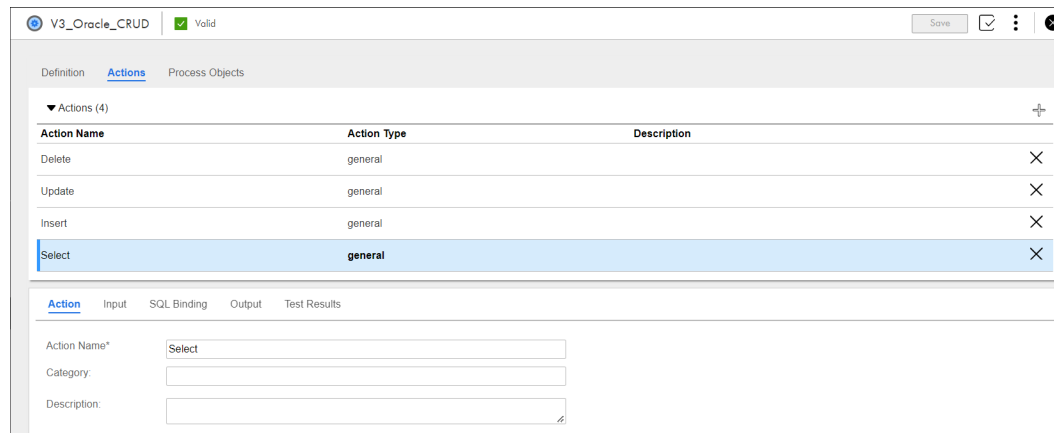
Click the row that you want to edit or click **+** to add a new row, and then enter the following information on the **Action** tab:

- **Action Name**: Enter the name that appears in lists when referencing this action in data access service connectors and connections. Do not use spaces or special characters in the action name. This is a required field.
- **Category**: If you have many data access service connectors, each with multiple actions, you can create categories to help users navigate within processes.
- **Description**: Enter a description or notes about this action.

For each action, specify additional details on the **Input**, **SQL Binding**, **Output**, and **Test Results** tabs.

Input Tab

Use the **Input** tab to define input data items that are unique to the service to which you are sending data.

The following image shows the **Input** tab:
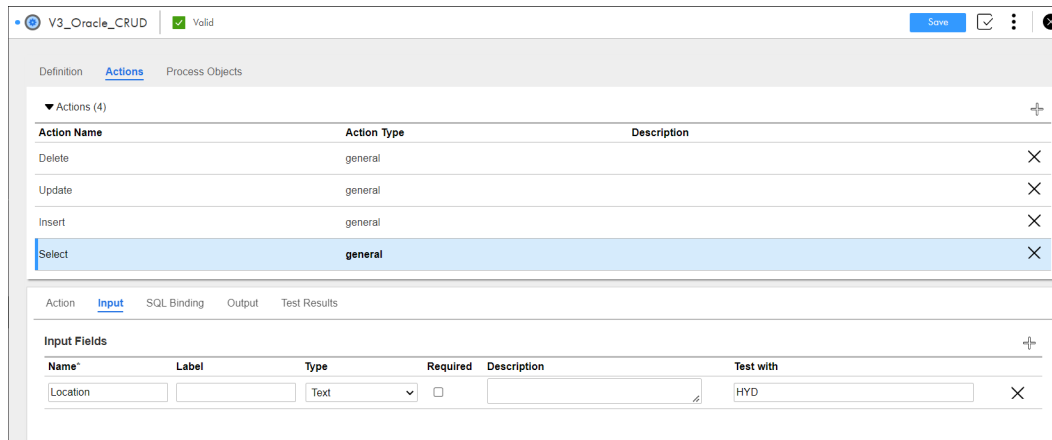
For each item, enter the following properties:

- **Name**: Required. The name of the input data item.

  Note: If you use a space character in the input field name and use the input field in the SQL query on the **SQL Binding** tab, you might see an error.

- **Label**: The name of the item to display within a process. If not specified, the name is displayed.

- **Type**: Select the data type of the item from the list. If you select the Reference or Object List type, you can also select a Process Object.

- **Required**: Select if a value must be set for this parameter.

- **Description**: Enter a description of the parameter.

- **Test with**: Enter a value that Process Designer will use to test the action. If the input is an attachment, you can also upload a sample attachment file.

Use the **+** icon to add a new item. Click **X** to delete the current row.

SQL Binding Tab

For each action in a data access service connector, use the **SQL Binding** tab to specify multiple SQL queries to access the data from the data source. After the SQL entries are made, the tab displays details such as the SQL name, SQL text, column case of the expression, query type, maximum rows returned by the SQL query, and maximum wait time in seconds.

When you add multiple queries, the statements are executed in the order in which the queries are listed in the table. You can use the arrow buttons to move a query up or down. You can also delete SQL queries.

To add a row of an SQL entry, click **+** on the **SQL Binding** tab. Enter the following details for each SQL query:
SQL Name

> Name of the SQL query.

SQL Text

> A valid SQL statement that returns at least one row.

Column Case

> Specify whether the characters in the expression must be converted into lowercase or uppercase, or remain unchanged. Default is **Lowercase**.

Type

Select one of the following values:

- **Data Manipulation**. Select this value to query information and apply logic to it to generate a completely different set of data.
- **Stored Procedure**. Select this value to use a prepared SQL code that you can save and reuse.

Has Result Set

Reserved for future use.

Maximum Rows

The maximum number of rows returned by the SQL query that can be displayed in the SQL response payload. The maximum number of rows can't exceed 10,000 rows.

Max Wait (Seconds)

The maximum length of time in seconds that a database client can wait for a response from the database before timing out the session.

Note: Application Integration does not perform a commit after it executes every SQL query. To help with a rollback in case of an SQL query failure, the commit is done only at the end after all the SQL queries are executed. In a few databases, implicit commit occurs for DDL statements such as CREATE TABLE, DROP TABLE, and so on. These executions can't be rolled back.

Configuring multiple SQL queries

Perform the following steps to add and edit SQL queries:

1. On the **SQL Binding** tab, click the **+** icon to add an SQL entry.

   An SQL entry is added to the table.

2. Click the **Edit** icon to edit the SQL query.

   The **Create SQL** page appears. The following image shows the **Create SQL** page where you can configure multiple SQL queries:

3. In the **Create SQL** page, configure the SQL query properties such as the SQL name, SQL query, column case of the query, query type, maximum rows, and maximum wait time.

   You can use SQL queries to perform various data manipulation operations such as SELECT, INSERT, DELETE, and UPDATE. You can parameterize an input field in an SQL query. Use the following syntax to parameterize an input field at run time:

   ```
   '{$<input_field_name>}'
   ```

   For example, if empId is the name of an input field, enter the following SQL query to parameterize the empId field:

   ```
   select * from employee.contact where id = '{$empid}'
   ```

4. Click **Create**.

SQL query response format

In a data access service connector, the XML response for single and multiple SQL queries begins with the `multiDataAccessResponse` tag.

The following sample snippet shows the format as the process or data access service output:

```
<multiDataAccessResponse xmlns:aetgt="http://schemas.informatica.com/socrates/data-
services/2014/05/business-connector-model.xsd"
 xmlns:bconn="http://schemas.informatica.com/socrates/data-services/2014/05/business-
connector-model.xsd">
    <dataAccessResponse>
        <result>
```

```
            <statementId>SQL Name 1</statementId>
                <row>
                <updatedRows>0</updatedRows>
                </row>
        </result>
        </dataAccessResponse>
        <dataAccessResponse>
            <result>
            <statementId>SQL Name 2</statementId>
                <row>
                <updatedRows>1</updatedRows>
            </row>
        </result>
        </dataAccessResponse>
</multiDataAccessResponse>
```

The following sample snippet shows the XML format as the response payload:

```
<multiDataAccessResponse>
    <dataAccessResponse>
        <result statementId="SQL Name 1">
            <row>
            <updatedRows>0</updatedRows>
            </row>
        </result>
    </dataAccessResponse>
    <dataAccessResponse>
        <result statementId="SQL Name 2">
            <row>
            <updatedRows>1</updatedRows>
            </row>
        </result>
    </dataAccessResponse>
</multiDataAccessResponse>
```

Output Tab

Use the **Output** tab to define how the data access service connector must parse data returned from the database.

The following image shows the **Output** tab:
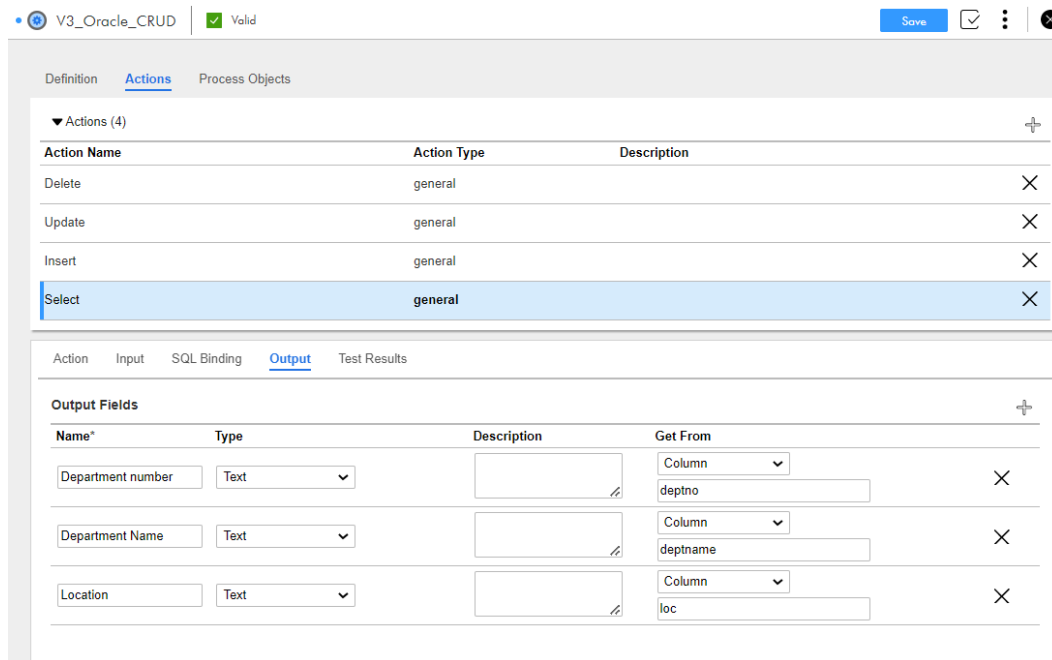
For each output data item, specify the following properties:

- **Name**: The name of a field into which a returned value is placed.
- **Type**: The data type of the value being written to the field. If the type is Object List or Reference, Process Designer displays a list of process objects from which you can choose one of the objects defined within the **Process Objects** tab.
- **Description**: Enter a description of the output field.
- **Get From**: Select one of the following options:
  - **Column**: To enter a column name (which is the column name in the database table used within the output returned by the database) to be placed within the output field.
  - **Expression**: To write an expression to parse the output returned by the database. Click **f(x)** to open the Expression Editor where you can type the expression.
  - **Result Set**: To assign the complete contents of the response from the database to the output field.
  - **Entire Response As Attachment**: To handle the entire response as an attachment.
  - **Attachments**: To work with multiple attachments and pass the entire list of attachments to the selected variable except the part used as the payload.

Creating Process Objects

On the **Process Objects** tab, you can define one or more process objects for a data access service connector to group data and create a structured object. When defined in the data access service connector, the process objects are available to processes that use the data access service connector. For example, if a database returns demographic information such as name, address, and phone number, you might create a single demographic process object that contains this information.

You can associate each process object with one or more of the data elements returned by the database using the **Actions/Output** tab.

To create data access service connector process objects, perform the following steps:

1. Create or open an existing data access service connector.

2. Click the **Process Objects** tab.

3. Click **+** to add a new process object or select an existing process object from the list. Each process object appears as a line item on the tab.
    The following image shows the **Process Objects** tab:
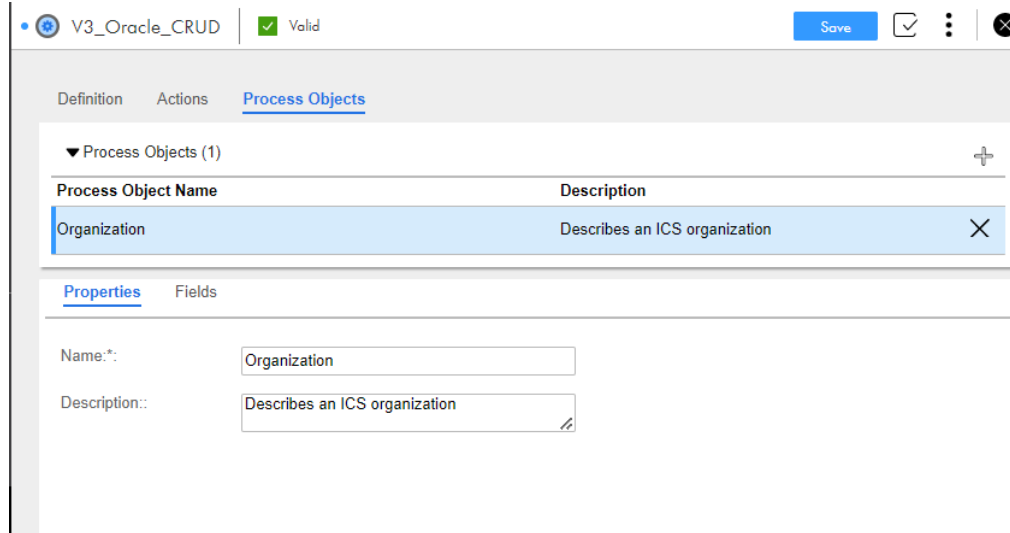
4. On the **Properties** tab, specify the following details for each process object:

    *   **Name**: Enter a name that identifies the process object. This name appears in the lists where the process object is available for selection. This is a required field.

    *   **Description**: Enter a description for the process object.

5. On the **Fields** tab, specify the following details for each process object:

    *   **Field Name**: Enter a name for each field in the process object.

    *   **Type**: Select the data type of the value being written to the field using one of the built-in data types such as Text, Integer, or Date Time. If the type is Object List, you also select a process object.

    *   **Required**: Select this check box to set the field as a required field.
        A required field must be included in the request payload. It can contain a value, be empty, or be null.

    *   **Nullable**: Clear this check box if the field must not accept null values. By default, the **Nullable** check box is selected.

Testing the data access service connector

When you use the data access service connector editor, click **Test** to send a request to the database and display the response data on the **Test Results** tab:

*   **Generate Process Objects**: Click to see the process objects that you have defined for the data access service connector.

*   **Result:** Displays the status of the test.

*   **Output** and **Value**: Shows the data returned from the database that is assigned to the fields defined on the **Output** tab.

*   **Response Payload**: Shows the payload sent to Process Designer from the database (not the full response sent by the database).

*   **SQL Request**: Shows the data sent to the database from the data access service connector. The tab displays a list of SQL names on the left pane and the corresponding data on the right pane.

- **SQL Response**: Shows all of the data sent back to the data access service connector from the database (whereas the response payload is only a portion of the returned data). The tab displays a list of SQL names on the left pane and the corresponding data on the right pane.

If the response includes one or more attachments, you can also download the attachments.

Note: The **Response Payload** and **SQL Response** sections show a maximum of 10,000 records. You can't specify a limit greater than 10,000 on the **SQL Binding** tab.

The following image shows the **Test Results** tab:

Generate Process Objects

A process object is usually tied to a specific set of elements. You may sometimes have a large number of identical objects, each of which applies to one set of elements. For example, when you define the data being returned, you create a process object whose sole purpose is to define a subset of returned data. These objects are *non-reusable* and can only be used by a single process object field. The object names are also the name of a field.

You can also create objects that are *reusable*. For example, the refType element in NetSuite has two fields: a name and an internalID. Instead of creating many objects, each of which contains these two fields, you can create a single, reusable process object.

1. Click **Generate Process Objects** to show the process objects that you have defined for the data access service connector.

2. Select the process objects that you want to make reusable and then click **Next**.
   Process Designer displays a list of generated process objects.

3. Click **Finish**.

Metadata parameters in response payload

To view the metadata details for a specific app connection, use the metadata parameter. The output includes contextual information about a piece of data or a data set that is stored alongside the data.

Use the following URI to view the metadata details for an app connection:

```
https://<Informatica Intelligent Cloud Services URL>/active-bpel/odata/v4/<app connection
name>/$metadata
```

The following sample snippet shows the format of the response payload:

```
        ▼<Annotation Term="Informatica.OData.Screenflow.V1.Required">
            <Bool>true</Bool>
          </Annotation>
        </Property>
        <Property Name="psg_container" Type="Edm.Decimal" Precision="19"/>
        <Property Name="pog_name" Type="Edm.String" MaxLength="765"/>
      </EntityType>
    ▼<EntityType Name="StudentODataTest">
      ▼<Key>
          <PropertyRef Name="studentid"/>
        </Key>
      ▼<Property Name="studentid" Type="Edm.Int64" Nullable="false">
        ▼<Annotation Term="Informatica.OData.Screenflow.V1.Required">
            <Bool>true</Bool>
          </Annotation>
        ▼<Annotation Term="Informatica.OData.Screenflow.V1.Readonly">
            <Bool>true</Bool>
          </Annotation>
        </Property>
      ▼<Property Name="firstname" Type="Edm.String" MaxLength="255">
        ▼<Annotation Term="Informatica.OData.Screenflow.V1.Required">
            <Bool>true</Bool>
          </Annotation>
        </Property>
      ▼<Property Name="lastname" Type="Edm.String" MaxLength="255">
        ▼<Annotation Term="Informatica.OData.Screenflow.V1.Required">
            <Bool>true</Bool>
          </Annotation>
        </Property>
        <Property Name="sectionid" Type="Edm.Int64" MaxLength="10"/>
        <Property Name="subjectid" Type="Edm.Int64" MaxLength="10"/>
      ▼<NavigationProperty Name="SectionODataTest" Type="Informatica.OData.AppConnectionTestSqlServer1.Models.SectionODataTest">
          <ReferentialConstraint Property="sectionid" ReferencedProperty="sectionid"/>
        </NavigationProperty>
      ▼<NavigationProperty Name="SubjectODataTest1" Type="Informatica.OData.AppConnectionTestSqlServer1.Models.SubjectODataTest1">
          <ReferentialConstraint Property="subjectid" ReferencedProperty="subjectid"/>
        </NavigationProperty>
      </EntityType>
```

OData expand parameter in response payload

After you publish an app connection, you can use the OData expand parameter to perform nested queries on the data. You can query up to one level of a navigation property in a service call.

Use the following URI with the OData expand parameter:

```
https://<Informatica Intelligent Cloud Services URL>/active-bpel/odata/v4/<app connection
name>/<table name>?$expand=*
```

The following sample snippet shows the output of the expand query:

```
 3        "value": [
 4            {
 5                "studentid": 1,
 6                "firstname": "FirstName1",
 7                "lastname": "FisrtName2",
 8                "sectionid": 11,
 9                "subjectid": 111,
10                "sectionodata": {
11                    "sectionid": 11,
12                    "numberofstudents": 10
13                }
14            },
15            {
16                "studentid": 2,
17                "firstname": "FirstName2",
18                "lastname": "FisrtName2",
19                "sectionid": 22,
20                "subjectid": 222,
21                "sectionodata": {
22                    "sectionid": 22,
23                    "numberofstudents": 20
24                }
25            }
26        ]
```

Validating a Service Connector

When you create a service connector or a data access service connector, you can use the Validation panel to verify if there are validation errors and view the error details.

1. Click the **Validation** icon in the toolbar as shown in the following image:

The **Validation** panel appears on the right pane. In the **Validation** panel, errors are grouped based on the tabs where they occur.

2. Click the arrow corresponding to a tab name as shown in the following image:

A list of errors that occur in the tab is displayed.

3. Click an error row as shown in the following image:

The cursor moves to the erroneous field. Take the required corrective action to resolve the error.

As and when you correct errors, the error list in the **Validation** panel is refreshed.

Saving and Publishing a Service Connector

After you create a service connector or a data access service connector, save and publish the service connector.

You can also publish multiple service connectors in bulk. For more information, see *Invoke*.

In the service connector designer page, click **Save** to save the service connector, and then click **Publish** to publish the service connector.

To disable a published service connector, you must unpublish the service connector. After you make the necessary changes, publish the service connector again for the changes to get reflected.

CHAPTER 7

Using Connectors

After you create a service connector, create an app connection to use in a process.

The following image shows the steps that you must follow to create an app connection:

Creating an App Connection

Perform the following steps to create an app connection:

1.   In Application Integration, click **New**.

2. In the **New Asset** dialog box, click **App Connections** > **App Connection**, and then click **Create** as shown in the following image:

Defining App Connection Properties

For each app connection, you enter descriptive properties, identify the scope, and set authentication parameters, if required.

These options display for all app connections:

- **Connection Type**: From this list, select a published service connector, native toolkit connector (such as JDBC), or web service connector (such as Salesforce). When you create a connection, the connection type that you select determines the connection properties that are displayed.

  Note: After you publish an app connection, you cannot change its connection type.

- **Name**: Enter a name that identifies an app connection within processes and in the Designer. The name must start with a letter, and can contain only alphanumeric characters, multibyte characters, and hyphens (-). The name must not exceed 128 characters. This is a required field.

  Important: If you define an app connection to be used for publishing processes in Salesforce, the **Connection Name** must be `Salesforce` to ensure that the processes are accessible to users in Salesforce. Only one Salesforce app connection is permitted for each Application Integration organization.

- **Description**: Enter a description to identify an app connection.

- **Run On**: Select the agent where you want to access a service behind your firewall or select **Cloud Server or any Secure Agent**. This list displays the agents available for your organization.

- **Connection Test**: Displays the results of any connection test run on the Cloud Server or the agent named in the **Run On** field.

- **OData Enabled**: Select **Yes** to enable OData feeds. If you select **Yes**, you must specify either the allowed users or the allowed groups that can access the connection at design time. You can also enable or disable Cloud access to the OData endpoint URLs.

After you enable OData in an app connection and publish the connection, you can click **Actions > Properties Detail** to view the generated OData Service URL and OData Swagger URL.

The OData Service URL is available on an Informatica Cloud endpoint and uses the following format:

```
https://<Informatica Intelligent Cloud Services URL>/active-bpel/odata/v4/
<OData_connection_name>
```

The OData Swagger URL is available on an Informatica Cloud endpoint and uses the following format:

```
https://<Informatica Intelligent Cloud Services URL>/active-bpel/odata/v4/
<connection_name>/$metadata?swagger
```

For more information about enabling the OData protocol in a Data Access Service Connector, see *Defining Properties*.

- **OData Cloud Access Enabled**: If you enable OData and configure the connection to run on a Secure Agent machine or a Secure Agent group, you can choose to disable Cloud access to the OData endpoint URLs for security purposes.

- **Allowed Users for OData:**
  The users that have access to this app connection at run time. You can enter more than one user in this field. After you specify the first value, press the Enter key or the Comma key, and then specify the next value.

  Allowed Roles for OData:
  The roles that have access to this app connection at run time. Users assigned these roles have access to an OData endpoint URL. You can enter a custom role or a system-defined role. You can enter more than one role in this field.

  IMPORTANT: If you enter a custom role in the **Allowed Roles for OData** field, a user invoking the OData endpoint URL must have or belong to groups that have the custom role and the Service Consumer role.

  Example:

  Your organization has a custom role, Testers, with the Run privilege enabled for Application Integration. You create a process and enter 'Testers' in the **Allowed Roles** field.

  For a user to be able to use the OData endpoint URL, the following conditions need to be met:

  - Condition 1: The user has the Testers role, a role that this app connection requires.

  - Condition 2: The user has the Service Consumer role. Application Integration requires this role to invoke any endpoint URL.

  A user with the Testers role and the Service Consumer role will be able to invoke the OData endpoint URL. This is because *both*, Condition 1 and Condition 2, are satisfied.

  A user with *only* the Testers role will be unable to invoke the OData endpoint URL. This is because Condition 1 is satisfied but Condition 2 is not satisfied.

  A user with *only* the Service Consumer role will also be unable to invoke this OData endpoint URL. This is because Condition 2 is satisfied but Condition 1 is not satisfied.

- **Schema**: Enter the name of the schema that contains the tables that you want to include or exclude in the metadata. To view the appropriate results, you must enter a schema name in the app connection that uses the data access service connector.

- **Include Tables**: Enter the names of the tables that you want to include in the metadata. To include a list of tables, use a comma to separate multiple table names. You can also use '.*' for pattern matching. To view the appropriate results, you must also mention the included table names in the app connection that uses the data access service connector.

- **Exclude Tables**: Enter the names of the tables that you want to exclude from the metadata. To exclude a list of tables, use a comma to separate multiple table names. You can also use '.*' for pattern matching. To view the appropriate results, you must also mention the excluded table names in the app connection that uses the data access service connector.

The remaining properties are determined by the connector or the connection type that you selected. After you enter the property value, click anywhere outside the property row or press Enter to save the value.

For example, the Salesforce connection type in the following image requires a **Name**, **Type**, **Authorization URL**, and **Token Request URL**:

Specifying Event Sources

Event sources are available for some app connections based on the service connector.

For the following AMQP app connection that provides access to the AMQP listener service, you can add one or more event sources:

First, click **Add Event Source** to add a new event source for this app connection. Then enter the following information for each event:

- **Name**: Enter a unique name for each event source. The name must start with a letter, and can contain only alphanumeric characters, and hyphens (-). This is a required field.

- **Description**: Enter a description, if needed, for the event source.

- **Enabled**: Select **No** to disable the event source until you are ready to deploy it. This gives you the flexibility to configure multiple event sources and determine when to enable them.

The connection properties that you need to configure vary based on the connection type. In this case, the AMQP connection requires the **Destination Type**, **Destination**, and **Payload Format** fields to be configured.

A message displays on the tab to notify you if the preview option is not enabled or supported for the connection type.

Specifying Event Targets

Similar to event sources, you can specify one or more event targets for each app connection on a separate tab.

Each connector might define different types of event targets. For example, using the File connector, you can select either **File Writer** or **Delimited Content Writer File Writer** for each event target that you define. Each event target includes a set of properties unique to that event target type.

To create a new event target:

1. Click **Add Event Target** (and choose the target type, if applicable) a new row is added.

2. Highlight the row and add an event name and description.

3. Edit the properties for the specific connector.

Updating sensitive field values for app connections

You can use the update resource to update the sensitive field values for app connections. You can update fields for a single app connection or multiple app connections.

To update the app connections using the update resource, you must be assigned the Admin, Deployer, Designer, or Operator role.

When you export an asset, sensitive fields such as passwords, access keys, and secret keys are not exported. After you import an app connection, you can add the connection and the fields that you want to update as input data in a REST client such as Postman. The input data consists of the Global Identifier/Global Unique Identifier (GUID) of the connection and a key-value pair of attributes to be updated. In addition, you can provide attributes for the consumers and producers of the connection. Consumers refer to the event sources and producers refer to the event targets. For more information about GUID, see the lookup resource and finding assets resource in *REST API Reference* in the Data Integration help.

To update the field values, use the following URL:

```
<Informatica Intelligent Cloud Services URL>/active-bpel/asset/v1/update?
assetType=AppConnection
```

You can add the input data as shown in the following sample:

```
PUT <baseApiUrl>/active-bpel/asset/v1/update?assetType=AppConnection
Content-Type: application/json
Accept: application/json
INFA-SESSION-ID: <Infa-Session-ID>
{
  "connections": [
    {
      "guid": {
        "$t": "<guidvalue>"
      },
      "attributes": [
```

```
          {
            "key": {
              "$t": "Access Key"
            },
            "value": {
              "$t": "<AccessKeyvalue>"
            }
          },
          {
            "key": {
              "$t": "Secret Key"
            },
            "value": {
              "$t": "<SecretKeyvalue>"
            }
          }
        ],
        "consumers": [
          {
            "name": {
              "$t": "consumer1"
            },
            "attributes": [
              {
                "key": {
                  "$t": "attribute1"
                },
                "value": {
                  "$t": "<AttributeValue>"
                }
              }
            ]
          }
        ],
        "producers": [
          {
            "name": {
              "$t": "producer1"
            },
            "attributes": [
              {
                "key": {
                  "$t": "attribute1"
                },
                "value": {
                  "$t": "<AttributeValue>"
                }
              }
            ]
          }
        ]
      },
      {
        "guid": {
          "$t": "<guidvalue>"
        },
        "attributes": [
          {
            "key": {
              "$t": "Password"
            },
            "value": {
              "$t": "<Passwordvalue>"
            }
          }
        ]
      }
    ]
}
```

You can use the login resource to get the `INFA-SESSION-ID`. For more information about the login resource, see *REST API Reference* in the Data Integration help.

If the fields are successfully updated, you receive the following response:

```
{
    "status": {
      "$t": "SUCCESS"
    }
}
```

The fields are updated based on the GUID of the connection. If the GUID is not found, you receive the following response:

```
{
  "status": {
    "$t": "FAILED"
  },
  "errorDetails": {
    "error": {
      "$t": "Update failed for assets"
    },
    "failedAssets": {
      "updateAssetFaultInfo": {
        "xmlns": "https://com.informatica.icrt/2021/08/updateAssets.xsd",
        "xmlns$aetgt": "https://com.informatica.icrt/2021/08/updateAssets.xsd",
        "name": {},
        "GUID": {
          "$t": "<Incorrect guid value>"
        },
        "fault": {
          "$t": "Entry not found for GUID."
        }
      }
    }
  }
}
```

Validating an App Connection

When you create an app connection, you can use the Validation panel to verify if there are validation errors and view the error details.

1. Click the **Validation** icon in the toolbar as shown in the following image:

The **Validation** panel appears on the right pane. In the **Validation** panel, errors are grouped based on the tabs where they occur.

2. Click the arrow corresponding to a tab name as shown in the following image:

A list of errors that occur in the tab is displayed.

3. Click an error row as shown in the following image:

The cursor moves to the erroneous field. Take the required corrective action to resolve the error.

As and when you correct errors, the error list in the **Validation** panel is refreshed.

Saving and Publishing an App Connection

After you create an app connection, save and publish an app connection.

You can also publish multiple app connections in bulk. For more information, see *Invoke*.

In the app connection page, click **Save** to save the app connection, and then click **Publish** to publish the app connection.

After you publish an app connection, you can view the published metadata and use the connection in a Service step of a process.

Viewing Published Metadata

After you publish an app connection, you can see the objects associated with an app connection.

In the following image, you see the published metadata of an app connection:

Note: For some connection types, published metadata is not available.

Unpublishing an App Connection

To disable a published app connection, you must unpublish the app connection. After you make the necessary changes, publish the app connection again for the changes to get reflected.

Click **Actions** > **Unpublish** as shown in the following image:

CHAPTER 8

System Services, Listeners and Connectors

This chapter includes the following topics:

Using the OData Provider

OData is a REST-based and standardized protocol that provides access to data over the Web. You can use the OData protocol to access data services from the Cloud, including internal data sources like those available with a JDBC connection.

OASIS has standardized on OData V4. Informatica recommends that you use version to use OData V4, and it is the default version.

Your organization can expose OData feeds on an endpoint such as:

https://[host].rt.informaticacloud.com/active-bpel/odata/[version]/[connection name]/[data source name]

For example, these two endpoints expose the *sampleparts* table:

- **OData Version 2**: https://[host].rt.informaticacloud.com/active-bpel/odata/v2/Parts/sampleparts

~or~

- **OData Version 4**: https://[host].rt.informaticacloud.com/active-bpel/odata/v4/Parts/sampleparts

Data is available in Atom XML format or JSON, including support for the XML content-type.

When the OData schema is generated, note the following:

- The IID fields are excluded.
- The key fields are added to the OData Key definition.
- The schema uses the same type as the native data type for each field.

You can enable OData in connections configured to run on Secure Agents. The Secure Agent, rather than opening a port, opens up an outbound connection to the Informatica Cloud servers through which all communication occurs. The Secure Agent then has access to any on-premises applications or data sources.

Note: If you publish an OData enabled connection to a Secure Agent, the OData URL on the **Connection** page is an Informatica Cloud URL.

You do not see a Secure Agent URL. To construct the Secure Agent URL, replace all text `odata/v4/Oracle` in the Informatica Cloud URL with `https://<host>:<port>/process-engine/`.

For example, if the Informatica Cloud URL is `https://ps1w2.rt.informaticacloud.com/active-bpel/odata/v4/Oracle`, the Secure Agent URL is `https://localhost:7443/process-engine/odata/v4/Oracle`.

Supported OData V4 and OData V2 URI Conventions

You can use many OData V4 and OData V2 URI conventions to access data.

Supported OData Version 4 Conventions

You can use the following OData V4 URI conventions to access data:

- Section 2. URL Components
- Section 3 Service Root URL
- Section 4.1 Addressing the Model for a Service
- Section 4.3 Addressing Entities
  - You can use only Canonical URLs
- Section 4.4 Addressing References between Entities
- Section 4.6 Addressing a Property
- Section 4.7 Addressing a Property Value
- Section 5 Query Options
- Section 5.1 System Query Options
- Section 5.1.1 System Query Option $filter
- Section 5.1.1.1 Logical Operators
- Section 5.1.1.1.1 Equals
- Section 5.1.1.1.2 Not Equals
- Section 5.1.1.1.3 Greater Than
- Section 5.1.1.1.4 Greater Than or Equal
- Section 5.1.1.1.5 Less Than
- Section 5.1.1.1.6 Less Than or Equal
- Section 5.1.1.1.7 And
- Section 5.1.1.1.8 Or
- Section 5.1.1.4.1 contains
- Section 5.1.1.4.2 endswith
- Section 5.1.1.4.3 startswith
- Section 5.1.1.6 Literals
- Section 5.1.1.6.1 Primitive Literals

- Section 5.1.3 System Query Option $select
- Section 5.1.4 System Query Option $orderby

    - You can use $orderby only with a JDBC connector.

- Section 5.1.5 System Query Options $top and $skip

    - You can use $skip only with a JDBC connector.

- Section 5.1.6 System Query Option $count
- Section 5.1.7 System Query Option $search.

    - You can do simple searches like http://host/service/Products?$search=blue

    - You cannot use $search and $filter at the same time

- 5.1.8 System Query Option $format

See here for the complete OData V4 specification for URI conventions.

Supported OData Version 2 Conventions

Informatica recommends that you use OData V4. However, you can use the following OData V2 URI conventions to access data:

- Section 1. URI Components
- Section 2. Service Root URI
- Section 3. Resource Path
- Section 3.1. Addressing Entries

    - You cannot use relationships and complex entities

    Section 3.3. Addressing Service Operations

    Section 4. Query String Options

    Section 4.1. System Query Options

    Section 4.3. Top System Query Option ($top)

    Section 4.5. Filter System Query Option ($filter)

    •With the $filter option, you can use only 'endswith' and 'startswith'

    Section 4.7. Format System Query Option ($format)

    Section 4.8. Select System Query Option ($select)

    Section 4.9. Inlinecount System Query Option ($inlinecount)

See here for the complete list of OData V2 specification for URI conventions.

Custom Composite Keys

When you use OData to access a JDBC connector, you can define and edit custom composite keys. You define a custom composite key when a database entity does not have a primary key. With this custom composite key that you create, you can perform read operations on the JDBC database.

To use custom composite keys, you must use OData V4.

Primary Keys and Custom Composite Keys

A primary key is an object field (a column in a database table or a database view) or a set of object fields (columns in a database) that can uniquely identify each record in a database table or database view. You use primary keys to query objects in the database.

If an entity does not have a primary key, you can define a custom composite key for the entity. To create a custom composite key, manually select a field or multiple fields for that key. The custom composite key takes the place of the primary key, and you can use it to read data from a JDBC database.

Example: You connect to a JDBC database of employee records and you see that the entity *empaccount* does not have a primary key. You can manually select the fields *email*, *ID*, and *name* to define a custom composite key for *empaccount*.

You can also edit custom composite keys that you created. If you defined a custom composite key that references an entity that no longer exists, you can remove the key. If a custom key contains fields that no longer exist, you can remove these fields.

Viewing the Custom OData Entity Keys Section

1. Enable OData and publish a JDBC connection.

2. Click the **Published Metadata** tab.

   You see a new section, Custom ODataEntity Keys, with the following information:

   - Error messages, if any, with information about entities without a primary key and custom keys assigned to entities that no longer exist.

   - A list of entities and their corresponding key fields.

3. Click **Edit Custom Keys**.

   The Custom OData Entity Keys dialog box opens.
   The following image shows the Custom OData Entity Keys dialog box:

On the left side, you see three types of entities:

- Entities that do not have a primary key.

- Entities that do not exist but still have a custom key mapped to them.

- Entities that have a primary key defined, and do not require a custom composite key.

On the right side, you see a list of fields that you can map to create a custom composite key.

Creating OData Custom Composite Keys

To create OData custom composite keys, you select entities that you want to use as primary keys.

1. Click **Edit Custom Keys**.

2. Select an entity from the list of entities.

3. Select the fields that you want to use as a custom key. You can select multiple fields.

   Important: Ensure that you analyze the data and select fields that uniquely identify each record in the database table or view. For example, if you want to use the fields **FirstName** and **LastName** to create a custom composite key, you cannot have two employees with the same first and last names.

4. Click **Apply**.

   You see the custom composite key that you created in the list under Custom ODataEntity Keys.

Editing Custom Composite OData Keys

You can edit and delete custom composite OData keys.

Publish Validation

When you publish a connection, Informatica Process Designer checks the custom composite keys you created and displays a warning if it encounters any potential issues.

The following image shows a sample warning message:

Using Salesforce System Services Listeners and Connectors

Salesforce Outbound Messages

Use a Salesforce outbound message (OBM) to trigger an Application Integration process.

For example, design an OBM such that a change to the Account Salesforce object triggers the CreateNewID process.

To create a Salesforce OBM, perform the following tasks:

1. Download and install the Informatica Cloud Real Time for Salesforce package (the 'managed package'). For more information, see the *Installing the Salesforce Managed Package* topic in the *Salesforce Managed Package* section.
2. Log in to Informatica Intelligent Cloud Services℠ and open the Application Integration service.
3. Create and publish the process you want the Salesforce OBM to trigger.
4. Log in to your Salesforce developer account and create a Salesforce OBM.

Salesforce OBM URL

When you invoke a process through a Salesforce OBM, Salesforce sends a message to a specific endpoint called the Salesforce OBM URL.

To view the Salesforce OBM URL for a process, perform the following steps:

1. In Application Integration, create a Salesforce connection and suffix the connection name with 'Salesforce'.
   For example, name the connection 'AccountDetails-Salesforce' or 'TestSalesforce' and perform steps 2 through 5 to see the Salesforce OBM URL.

   Note: You cannot view the Salesforce OBM if you use 'Salesforce' anywhere else in the name except at the end.
2. Save and publish the Salesforce connection.
3. Create, save, and publish a process that uses the Salesforce connection..
4. From the **Explore** page or from the **Process Designer** page of the process, click **Actions** > **Properties**.

5. In the **Properties** window that appears, click the **Process Properties** tab to view the Salesforce OBM URL. The following image shows a sample **Process Properties** tab with a Salesforce OBM URL visible:

Creating a Salesforce OBM

To create a Salesforce OBM, log in to your Salesforce developer account, create a workflow rule, and then define an OBM.

Before you create a Salesforce OBM to trigger an Application Integration process, you must perform the following tasks:

- Install the managed packaged in the Salesforce organization in which you want to create an OBM.
- Create and publish the Application Integration process you want the Salesforce OBM to trigger.

1. Log in to your Salesforce developer account.
2. Go to **Create** > **Workflow & Approvals** > **Workflow Rule**.
3. Click **New Rule**.

4. On the **New Workflow Rule** page, select the object you want the rule to apply to and then click **Next**.

5. On the **Configure Workflow Rule** page, enter a rule name and description, set evaluation and rule criteria, and then click **Save & Next**.

6. Go to **Add Workflow action** > **New Outbound Message**.

7. On the **Configure Outbound Message** page, enter the required information, including the **Endpoint URL**. See the Salesforce OBM URL on page 313 topic for information on how to obtain the Endpoint URL.

8. Click **Save**.

The Salesforce OBM creation is complete. The Salesforce OBM you created triggers an Application Integration process according to the criteria you set.

For detailed information about configuring a Salesforce OBM, see the Salesforce documentation.

Configuring OBMs from multiple Salesforce organizations to single IICS organization

The Salesforce Managed Package uses a single sign-on mechanism by default. There must be a one-to-one relationship between a Salesforce organization and the corresponding Informatica Cloud organization. However, you might want to have more than one Salesforce organization generate OBMs intended for specific processes that run in a single Informatica Cloud organization.

To do this, configure Salesforce OBMs with the `authType` query parameter. The `authType` parameter represents the type of authorization to use and establish a relationship between an Informatica Cloud organization and multiple Salesforce organizations through the configuration of user accounts.

Note: You cannot use the `authType` parameter to have multiple Salesforce organizations use the same process. There must be a one-to-one relationship between the Salesforce organization and the specific process to which you are sending an OBM.

To enable the Salesforce OBM with the `authType` query parameter, perform the following steps:

1. For each Salesforce organization that will send OBMs, create a dedicated Salesforce user account. Ensure that the Informatica Cloud Real Time for Salesforce managed package is installed in these organizations. However, you do not need to configure the package and provide an Informatica Cloud Real Time Host URL.

2. To associate the Salesforce user with a specific Informatica Cloud user, perform the following steps:

    a. Log in to Informatica Intelligent Cloud Services (IICS), and select **Administrator** > **Users**.

    b. Navigate to the Informatica Cloud user that you want to associate with the Salesforce user.

    c. In the **Salesforce User Name** and **Confirm Salesforce User Name** fields, enter the user name that you created in Step 1.

3. Configure OBMs in Salesforce to use the `authType=ICSUser` query parameter. This parameter indicates the type of authorization to use when validating the Salesforce session ID and session URL in Informatica Cloud. The query parameter is a literal value. For example, you can use the following URL:

    ```
    https://use4-cai.dm-us.informaticacloud.com/active-bpel/services/REST/
    SalesforceNotificationRESTPort?processName=TestOuboundMessage&authType=ICSUser
    ```

    Note: You can use the `authType` parameter only for processes.

4. In the Salesforce Outbound Message definition, perform the following steps:

    a. Enter the Salesforce user name in the **User to send as** field. This is the user name that is used while sending the message.

    b. Select **Include Session ID**.

    c. At run time, verify that the user name mentioned in step 4a matches with the Salesforce user name entered in step 2c, and the process must exist in the organization.

Note: If you use the `authType=ICSUser` query parameter, only one user per Salesforce organization is allowed to send the OBM. A different Salesforce user can have a corresponding user in the Informatica Cloud organization with all user roles, but any OBM sent from this user will not trigger any process. In the **OBM Monitoring** page in Salesforce, you see the HTTP error `403 Forbidden` for the OBMs sent from a newly created user. The OBM might trigger the same process or a different process.

For more information about how the Salesforce OBMs are routed to the respective Cloud Application Integration organisation even though the OBM URL does not contain the organization ID, see the following community article:

https://knowledge.informatica.com/s/article/513595.

Configuring OBMs from a single Salesforce organization to different IICS organizations

You can use a single Salesforce account to run different processes in different Informatica Intelligent Cloud Services (IICS) organizations.

Consider that you have a Salesforce account S, and you want to trigger process A created in IICS organization A and process B created in IICS organization B using the same Salesforce account.

Use the Salesforce account S to trigger process A in IICS organization A

1. Log in to the Salesforce account S and install the Informatica Cloud Real Time for Salesforce package. The managed package connects your Informatica Intelligent Cloud Services and Salesforce accounts.

2. In the **Guide Setup** page, enter the Informatica Cloud Application Integration Host URL in the following format:
   `<POD-specific Cloud Application URL for Salesforce>,<Informatica Organization ID>`

3. Create a Salesforce connection.

4. Log in to your Salesforce developer account and create a workflow rule to apply a Salesforce object. Create a Salesforce OBM to invoke process A in IICS organization A.

5. Trigger the workflow rule.
   Process A in IICS organization A is triggered using the Salesforce account S.

Use the Salesforce account S to trigger process B in IICS organization B

1. Log in to IICS, and select **Administrator** > **Users** > **Add User** and create a new user in IICS organization B with Salesforce authentication as shown in the following image:

   You will receive an email containing the user name and verification code to activate the user account.

   Note: The user name must be the same as the Salesforce account user name, that is, S.

2. Log in to the Salesforce organization S. To install a new Salesforce managed package, access the following URL:
   https://login.salesforce.com/packaging/installPackage.apexp?p0=04t3A000001AK3E

After the installation is complete, the installed managed package appears under Installed Packages as shown in the following image:
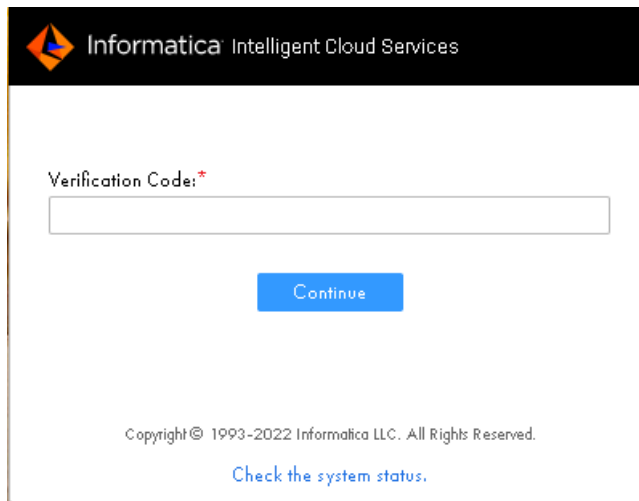

3. Click **All Tabs** at the top of the page.

The **Integration** tab appears on the **All Tabs** page as shown in the following image:

4. Click the **Integration** tab, and then click **Single Sign-On**.
   You are redirected to https://dm-us.informaticacloud.com URL and the IICS user verification page appears as shown in the following image:

Note: To verify Single Sign-On, the Salesforce user account must exist at any Point of Deployment (POD) within the https://dm-us.informaticacloud.com URL.

5. Enter the verification code that you received in the email.
   After successful verification, you are redirected to the IICS account.

6. Use the Salesforce account S to create a workflow rule to apply a Salesforce object. Create an OBM to invoke process B in IICS organization B. Configure an OBM in Salesforce to use the `authType=ICSUser` query parameter. This parameter indicates the type of authorization to use when validating the Salesforce session ID and session URL in Informatica Cloud. For example, you can use the following URL:

```
https://use4-cai.dm-us.informaticacloud.com/active-bpel/services/REST/
SalesforceNotificationRESTPort?processName=TestOuboundMessage&authType=ICSUser
```

Note: You can use the `authType` parameter only for processes.

7. Trigger the workflow rule.
   Process B in IICS organization B is also triggered using the same Salesforce account S.

For information about installing the Salesforce managed package, see *Salesforce and Application Integration*.

Using a Salesforce Connection

If you are using the Salesforce Managed Package, the Salesforce Connector allow you to define a connection to your Salesforce organization using one of these authentication methods:

- OAuth Connection
- OAuth JSON Web Token (JWT) Connection
- Password/Security Token Authentication

When you create a new Salesforce connection, choose the **Authentication Type** and provide the related connection properties.

Application Integration supports version 56.0 of the Salesforce API for password and OAuth based authentication when you create a new Salesforce connection. Version 56.0 offers various data objects and metadata types.

OAuth Connection

Salesforce supports OAuth to allow access to Salesforce.com through its API. OAuth is a standard protocol that allows secure API authorization. With OAuth, users do not need to disclose their Salesforce credentials and the Salesforce administrator can revoke the consumer's access at any time.

Application Integration supports version 56.0 of the Salesforce API for password and OAuth based authentication when you create a new Salesforce connection. Version 56.0 offers various data objects and metadata types. For example, you can use the ContactPointAddress object that represents a contact's billing or shipping address corresponding to an individual account.

Note: To use the various data objects and metadata types offered by Salesforce API version 56.0 for a Salesforce app connection, you must upgrade the managed package that is installed in Salesforce. For more information about Salesforce objects and services, see *Salesforce Connector Guide*.

To enable OAuth as the authentication type for a Salesforce connection, perform the following steps:
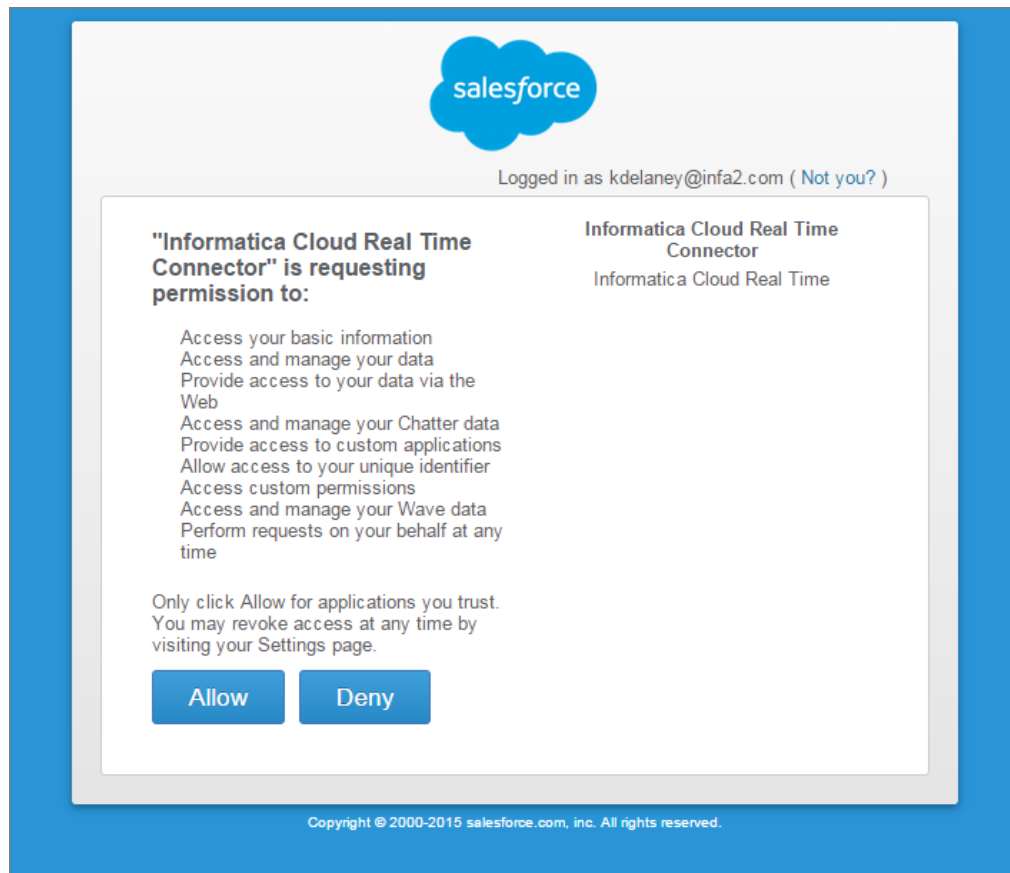
1. In Application Integration, click **New**.

2. In the **New Asset** dialog box, click **App Connections** > **Create**. The **AppConnection** page appears.

3. In the Authentication section, select **OAuth** from the list.

Authentication:		
Authentication Type:	OAuth	
Name	**Value**	**Description**
Authorization URL:*	https://login.salesforce.com/services/oauth2/authorize	Enter the Salesforce OAuth authorization URL. Default value for production is https://login.salesforce.com/services/oauth2/authorize. For Sandbox, use https://test.salesforce.com /services/oauth2/authorize.
Token Request URL:*	https://login.salesforce.com/services/oauth2/token	Enter the OAuth token request URL. For production, use https://login.salesforce.com/services/oauth2 /token. For sandbox, use https://test.salesforce.com/services/oauth2/token.
Session Duration:	60	Enter the number of minutes to wait before refreshing the session. Default is 60 minutes.
Authorization Status:	Not yet authorized.	Indicates the current status and the last time that authorization was completed.
Authorize Access:	Authorize	Click to initiate the authorization workflow using OAuth.

4. Enter the production or test URL as instructed on the screen for these required fields:

   - **Authorization URL**. Salesforce provides a dedicated URL that handles authorization. Enter the URL for either the production or test environment to access with this connection.

- **Token Request URL**. Salesforce provides a dedicated URL that handles token requests. Enter the URL for either the production or test environment to access with this connection.

5. In the **Session Duration** box, enter the number of minutes for which you want to maintain each session that was opened for this connection. Default is 60 minutes.

6. Click **Authorize** to start the authorization process.

   Note: To successfully launch the authorization process, pop-ups must be enabled in your browser. After you start the authorization process, you can cancel it prior to completion by closing the dialog box. In this case, the previous state (authorized or not authorized) is unchanged.

7. If prompted, enter the user name and password in the Salesforce login page and click **Log In to Salesforce**. After your login is validated, the following page appears:

8. Click **Allow** to complete the authorization. In the Salesforce connection properties, an **Authorization Status** message appears. The message includes the name of the last user who authorized the connection and the time of authorization.

Note: You must complete the authorization process within 3 minutes. Otherwise, the process times out. If that occurs, return to the connection properties and click **Authorize** again.

The authorization remains active as long as the connection is published, provided that the OAuth access was not revoked in Salesforce. After a session expires, it is automatically renewed in the background.

Salesforce Connected App Settings

Depending on the type of OAuth access enabled by the Salesforce administrator, you might be able to self-authorize or have pre-authorization. That option is determined by the Salesforce Connected App settings shown in the following image:

To learn more, refer to the Salesforce documentation.

OAuth JWT Connection

You can configure OAuth JSON Web Token (JWT) authentication in a Salesforce connection to connect to Salesforce.

Use OAuth JWT authentication to authorize servers to access data without logging in each time the servers exchange information. The OAuth JWT authentication uses a certificate to sign the JWT request and does not require explicit user interaction.

When you choose the OAuth JWT authentication type, configure the following properties:

- **User Name**. The Salesforce user name that has access to the connected app.
- **Keystore File**. The keystore file of the PKCS12 format.
- **Keystore Password**. The password to access the keystore file.
- **Session Duration**. The number of minutes after which the session expires.
- **Consumer Key**. The consumer key associated with the Salesforce connected app.
- **Token Request URL**. The OAuth token request URL.
- **Audience**. The authorization server URL of the intended audience for the token.

For more information about the OAuth JWT authentication in a Salesforce connection, see *Salesforce Connector Guide*.

Password/Security Token Authentication

When you choose the Password authentication type, you supply these properties:

- **User Name**: The username, for API access, to be used for the connection.
- **Password**: The password of the user associated with this connection.
- **Security Token**: The security token that provides this user with API access.

- **Service URL**: The Salesforce URL to use for the login. Be sure to specify the Soap/c WSDL (and not Soap/u).

At runtime, each connection is enabled based on the password and security token you provide.

Using the Email Service

Email Connection Properties

You can define an Email connection to send an email from an email server to a specified list of recipients. You can use OAuth 2.0 authentication or password-based authentication for an Email app connection. You can use OAuth 2.0 authentication only for a Microsoft Outlook email account.

In some cases, you might want to define multiple Email connections. For example, you might use different SMTP hosts for each school within a university or have multiple domains that each require a unique SMTP configuration. Each SMTP host requires a separate connection.

When you select **IICS Cloud Application Integration Email (Licensed for use)** as the connection type, you can configure the email-specific connection properties on the connection creation page.

To create the email connection using password-based authentication, configure the following properties on the connection creation page:

Property	Description
Authentication	Required. Specifies whether the Email connection must use password-based authentication. If you select **Enable**, Email Connector authenticates the user name and password that you enter in the email connection properties. If you select **Disable**, Email Connector does not perform authentication, and ignores the user name and password that you enter in the email connection properties. Default is **Enable**.
User Name	Required if you enable password-based authentication. If you disable password-based authentication, Email Connector does not perform authentication, and ignores any value that you enter in this field. Enter the user name to log in to the email server. The user name is either the account name or the email address. For example: `notifyme@mydomain.com` **Note:** If you have configured SMTP in your environment to work without password-based authentication, do not enter a user name and password.
Password	Required if you enable password-based authentication. If you disable password-based authentication, Email Connector does not perform authentication, and ignores any value that you enter in this field. Enter the password for the email address. **Note:** If you have configured SMTP in your environment to work without password-based authentication, do not enter a user name and password.

Property	Description
Security	Required. Specify the security protocol that the Email connection must use. Select one of the following values: - **None**. The Email connection does not use a security protocol. - **TLS**. The Email connection uses the TLS protocol. - **SSL**. The Email connection uses the SSL protocol. Default is **None**.
Connection Timeout	Optional. Specify the number of milliseconds to wait when attempting to connect to the email server. Default is **30000** milliseconds. If you encounter a timeout while using an Email connection, review your network and firewall configuration.

To create the email connection using OAuth authentication, configure the following properties on the connection creation page:

Property	Description
Authorization URL	Required. Enter the OAuth authorization URL for the email service that is used to authorize the user request. For example: `https://login.microsoftonline.com/xxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx/oauth2/v2.0/authorize`
Token Request URL	Required. Enter the OAuth token request URL that handles token requests. For example: `https://login.microsoftonline.com/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/oauth2/v2.0/token` The refresh token expires in 90 days. The user must authenticate again and publish the connection before the token expires.
Client ID	Required. Specify the identifier value from the OAuth provider.
Client Secret	Required. Enter the client secret to connect to the email application.
Scope	Required. Specify the scope. The scope in OAuth authentication limits an application's access to a user's account. You can select multiple scopes for a single client. To enter multiple scopes, separate each value with a space. For a Microsoft Outlook email account, enter the following scope: `https://outlook.office.com/SMTP.Send offline_access`
Authorization Status	Indicates the current status, the name of the user who authorized, and the last time when the authorization was completed.
Authorize Access	Click to initiate the authorization workflow using OAuth.

Note: In the OAuth provider, set the redirect URI to the host of the CAI POD in which you create a connection. The redirect URI must contain the `/oauthcallback` string. For example: `https://<Informatica Cloud Application Integration URL>/oauthcallback`

Configure the following common properties on the connection creation page:

Property	Description
Host	Required. Enter the email server's DNS name, such as `mail.mydomain.com`, or an IP address, such as `192.168.1.1`.
Port	Required. Enter the port for communication between the Process Server and the email server. Default is **25**.
Test Email Address	Optional. Enter the email address to which you want to send a test message when you test the Email connection. Separate multiple email addresses with a comma character.
From Email Address	Optional. Enter the email address that you want to appear in the **From** field of emails sent from the connection. For example: `no-reply@example.com`

Publishing Email Connections

After you create an Email connection, you can validate, save, and publish the connection. For more information, see *Design*.

When you publish an Email connection, Application Integration creates the following items on the **Metadata** tab:

sendEmailService

sendEmailService is the action that you must select in a Service step of a process or guide to send an email.

The following image shows the sendEmailService action:

AttachmentInformation

The AttachmentInformation object defines the attachments to be sent with the email. You can use XQuery attachment functions to construct attachments. For more information about creating and using attachments, see the following link: Attachments.

The following image shows the AttachmentInformation object:

EmailTaskInput

The EmailTaskInput object defines the input for the email. The object defines the recipients, email subject, email content type, email body, and attachment information.

The following image shows the EmailTaskInput object:

EmailTaskOutput

The EmailTaskOutput object defines the response of the sendEmailService action. The object defines whether the email was sent successfully, and also lists the following details:

- Email addresses to which the email was sent.
- Email addresses that are valid but to which the email was not sent.
- Email addresses that are not valid.

The following image shows the EmailTaskOutput object:

Using Email Objects in a Process or Guide

To use the email objects in a process or guide, perform the following steps:

1. Create a field to which you want to assign the email object.

2. From the **Type** list, select **More types**.

3. From the **Category** list, select **Connection defined Types**.

4. From the **Connection** list, select the email connection.

5. From the **Types** section, select the required email object as shown in the following image:

You can then assign values to the email object fields in the process or guide.

Using an Email Connection in a Process or Guide

After you create an Email connection, you can use it in a Service step in a process or guide.

Perform the following steps to use an Email connection in a process or guide:

1. Add a Service step to the process or guide.

2. Click the **Service** tab.

3. From the **Service Type** list, select **Connection**.

4. From the **Connection** list, browse and select the Email connection that you created.

5. From the **Action** list, select the **sendEmailService** action that was created when the Email connection was published. Application Integration populates the **emailTaskInput** object under input fields and the **emailTaskOutput** object under output fields.
    The following image shows a sample Service step that sends an email:

6. Click the **Input Fields** tab.

7. Set the content of the email message in the **emailTaskInput** field.

   To send an email to multiple recipients, separate the email addresses with a comma character.

8. Configure email attachments and output fields as needed.

9. Validate, save, publish, and run the process or guide.

Using Java Message Service (JMS)

A Java Message Service (JMS) enables messaging between loosely-coupled processes. "Loosely-coupled"-for purposes of messaging-means the messages are independent of one another. The interaction between the processes can be synchronous or asynchronous.

If the process sends a message while the message consumer is not running, the message waits in the JMS provider's queue until the message consumer is available. The contents of these messages can vary, but the data within the message is usually expressed as XML.

Taking advantage of the data flow that occurs with JMS, you can eliminate the need to invoke a service such as a database management system that runs on-premise.

Using the Shell Command Invoke Service Connector

You can create an invoke activity that calls a POJO service to execute a shell command script. The invoke activity is based on a shell command system service provided by Process Server. You can, for example, call a script that updates a database or adds a new user to an identity service. To use the shell command system service, you must create and locate your script in a working directory that is specified within the input message for the invoke activity.

The following message shows an example of input for the invoke activity.

```
<shl:execCommand xmlns:inv =
    "urn:com:activee:rt:shellcmd:services:invoke"
    xmlns:shl="urn:com:activee:rt:shellcmd:services:ishellcmdinvoker">
```

```
<inv:shellRequest>
    <inv:command>string</inv:command>
    <inv:workingDirPath>string</inv:workingDirPath>
    <inv:charset>utf-8</inv:charset>
    <inv:mergeErrorAndOutput>true</inv:mergeErrorAndOutput>
    <inv:stdInput>string</inv:stdInput>
</inv:shellRequest>
</shl:execCommand>
```

The elements of the message are described in the following table.

execCommand Input Element	Description

command	(Required) Name of the script to execute. Examples:
	`cmd.exe /C myScript`
	`concat('shell.bat', $requestVar)`

workingPathDir	(Required) Working directory for the execution of the script
charset	Character encoding of the script. The default is UTF-8.
mergeErrorandOutput	True is the default setting. If true, directs errors as well as execution output to the same console.
stdInput	Data for the script

CHAPTER 9

Designing Human Tasks

A human intervention becomes necessary in business processes where a decision needs to be taken. For example, a human action is required for approvals and exception management. In such cases, you can use a human task. A human task needs a human intervention to get completed. Create a human task asset to define the user task to include in a business process. You can create and edit a human task definition.

You first configure the human task asset in Application Integration. The asset defines the human task, the users and roles that will have access to the human task, and the task outcome. You can add the human task asset to an Application Integration process by using a Human Task step.

When the process runs, the human task that gets generated is sent to the Human Task Inbox of the Human Tasks service. Based on the role assigned to you, you can use the Human Task Inbox to view the tasks assigned to you and take appropriate actions. For more information about the Human Task Inbox, see the Human Tasks help.

After the task is complete, the process receives a callback and continues with the subsequent steps.

After a process completes, you can view the Human Task step execution details in Application Integration Console.

Example

You want to create a loan approval process. The process must allow a requestor to submit a loan application. The request is assigned to a bank employee or loan agent qualified to approve or reject the request based on various evaluation criteria. A human user then reviews, and approves or rejects the application. If the loan is approved, the loan is sanctioned to the requestor.

To achieve this use case, you perform the following tasks in Informatica Intelligent Cloud Services:

1. Create a loan approval process in Application Integration.

2. Create a human task asset in Application Integration. The asset defines the human task details.

3. Add a Human Task step to the process to include the human task asset and task.

4. When the process runs, it sends the task to the Human Task Inbox of the Human Tasks service.

5. The potential owners, administrators, and stakeholders that are defined in the human task asset can see the human task when they open the Human Task Inbox in the Human Tasks service.

6. The listed potential owners can claim the task. The user claiming the task becomes the task owner who can perform further actions on the task.

7. The task owner analyzes the loan application, approves or rejects the loan application, and completes the human task.

8. The process then continues to the next step in Application Integration and the loan is sanctioned to the requestor.

User roles for Human Tasks

A human task needs human intervention to get completed. When you create a human task asset in Application Integration, you assign users or roles to monitor and act upon the task. The actions that can be performed on a task vary based on the privileges assigned to a specific user or role.

You can configure the following roles for human tasks:

Task Owner

A task owner is the user that is assigned the task and can perform actions on it. The task owner can start, stop, suspend, resume, release, and complete a task. A task owner can also attach files and add comments to tasks. A task always has one task owner.

Potential Owners

A potential owner can become the actual owner of a task and belong to a user group that can work on a task. When a task is generated, all the potential owners can view the task. A potential owner can claim a task. After a task is claimed by a potential owner, the potential owner becomes the task owner, and the task can't be claimed by other potential owners.

A potential owner can claim, start, stop, suspend, resume, release, and complete a task. A potential owner can also attach files and add comments to tasks.

Excluded Owners

Excluded owners are the users or roles that are excluded from processing a human task. Excluded owners can't access, own, or perform any action on a task.

Administrators

Administrators are the users or roles that are authorized to assign and reassign tasks. Administrators can also attach files and add comments to tasks.

Stakeholders

Stakeholders are the users or roles that can oversee tasks and take necessary actions when a problem arises. Stakeholders can monitor the actions taken on a task by a task owner, claim a task, and reassign a task to another user when needed. Stakeholders can also attach files and add comments to tasks. A task can have multiple stakeholders.

Guide-rendered tasks

Guide-rendered tasks provide the user with a wizard that guides the task owner to enter data and complete a task.

A guide is a set of screens that prompts users to review, enter, or confirm data. You create a guide in Application Integration. When you create a human task asset in Application Integration, you can choose to use a guide to display a wizard for task owners' ease.

For example, a step might display a loan application or prompt the user to approve a loan amount that is requested by a customer. The steps in a guide interact with your application by extracting and storing data. Guides run within mobile apps or on traditional platforms such as a PC or a Mac.

If you select a guide at the time of creating a human task asset in Application Integration, the task in the inbox uses the screens configured in the guide to display data. The input fields, output fields, and outcomes of the selected guide must match the values that you specify in the human task asset. If you do not select a guide at the time of creating a human task, the task uses the default rendering.

Human Task creation

Use the human task asset to define the properties for the human task. When you create the asset, you can specify all the human task settings such as the task details, user or role assignments, input fields and output fields, task outcomes, and inbox display settings.

To create a human task asset, you perform the following steps:

1. Define the basic human task asset details.
2. Define the potential owners that can claim and act on a task.
3. Define the excluded owners that cannot access or work on a task.
4. Define the task administrators that can assign and reassign tasks.
5. Define the stakeholders that can monitor, assign, and reassign tasks.
6. Configure the input fields and output fields for the task.
7. Configure the task outcomes.
8. Configure the inbox display settings for the task.

Step 1. Define a Human Task asset

On the **Definitions** tab of human task asset, you can specify a name to identify the task, location to store the task properties, description for the task, and priority for the task. You can also configure the task to allow the user to skip the task if it is of low priority.

The task name and the description provided on the **Definitions** tab of the human task asset appear in the user's task panel of the Human Task Inbox so that the user can quickly identify the task without having to view the details.

1. In **Application Integration**, click **New**.

    The **New Asset** dialog box appears. The following image shows the **New Asset** dialog box:

2. Select **Human Task** and click **Create.**

    The **Definition** tab appears as shown in the following image:

3. Enter a name for the human task asset.

4. Use the default project or folder location or select a custom location. To select a custom location, click **Browse**, select a new location, and click **Select**.

5. Optionally, you can enter a description for the task.

6. Define the priority for the human task created by the process. The lesser the number, the higher the priority.

7. Select the **Skippable** option for non-critical tasks to allow reviewers to skip the task in the Human Workflow Inbox. After the task is skipped, the process continues with the next step.

Step 2. Define potential owners

On the **Assignments** tab of the human task asset, you can add a list of all the users and roles that can access the task and take actions. You can add a list of potential owners on the **Assignments** > **Owners** tab. The potential owners can become the actual owners of the task and can work on a task after claiming the task.

Add the users or roles that can claim the task and act on it. If you do not add any potential owner, all users can claim the task. If you add only one user as a potential owner, the task is assigned only to the configured owner.

1. Click the **Assignments** tab.

   The following image shows the **Owners** tab:

2. On the **Owners** tab, click **+** in the **Potential Owners** section to add users or roles from the list.

3. Perform one of the following steps:

   • To add a user, click **Users** and select a user from the list. Click **OK**.

   • To add a role, click **Roles** and select a role from the list. Click **OK.**
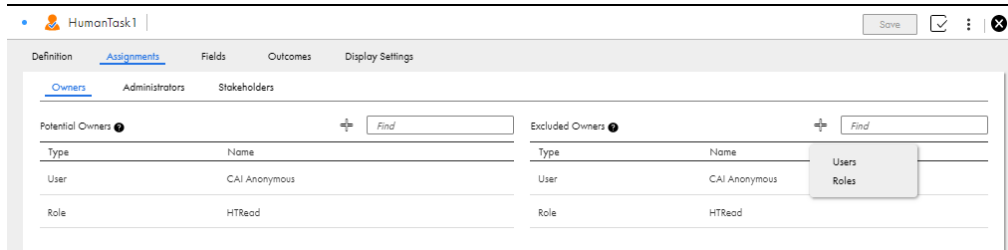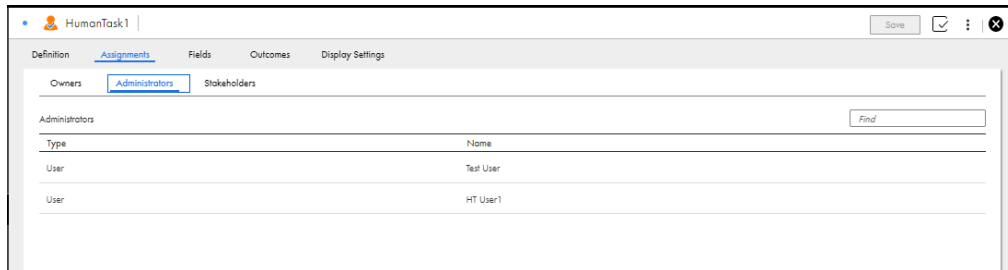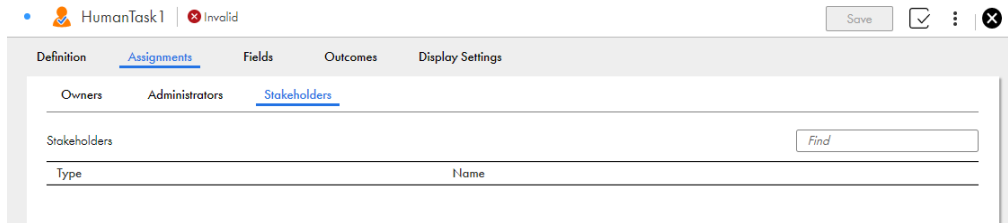
Step 3. Define excluded owners

On the **Assignments** > **Owners** tab of the human task asset, you can add a list of excluded owners who will not be able to access, own, or perform any action on the task.

Add the owners that you would like to exclude from claiming a task. If you add a user or role as both a potential owner and an excluded owner, the user or role cannot claim the task. If you add a potential user's role in the excluded owner list, the user cannot claim the task.

1. Click the **Assignments** tab.

   The **Excluded Owners** section appears as shown in the following image:

2. On the **Owners** tab, click **+** in the **Excluded Owners** section to add users or roles from the list.

3. Perform one of the following steps:

   - To add a user, click **Users** and select a user from the list. Click **OK**.
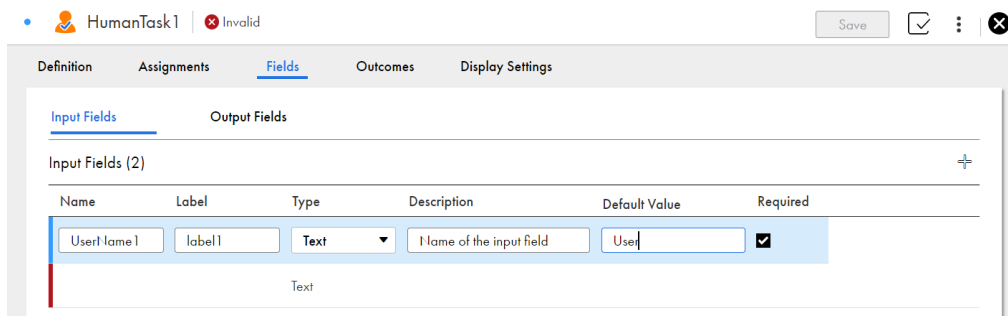   - .To add a role, click **Roles** and select a role from the list. Click **OK.**

Step 4. Define task administrators

On the **Assignments** > **Administrators** tab of the human task asset, you can add a list of administrators who are authorized to assign and reassign tasks.

1. Click the **Assignments** tab.

2. Click the **Administrators** section. The **Administrators** section appears as shown in the following image:

3. Click **+** to add the users or roles from the list.

4. Perform one of the following steps:

   - To add a user, click **Users** and select a user from the list. Click **OK**.
   - .To add a role, click **Roles** and select a role from the list. Click **OK.**

Step 5. Define stakeholders

On the **Assignments** > **Stakeholders** tab of the human task asset, you can add a list of stakeholders who are authorized to review a task. Stakeholders can take necessary actions when a problem arises. Stakeholders can also attach files and add comments to tasks. A task can have multiple stakeholders.

1. Click the **Assignments** > **Stakeholders** tab.

   The **Stakeholders** tab appears as shown in the following image:

2. Click **+** to add users or roles from the list.

3. Perform one of the following steps:

   - To add a user, click **Users** and select a user from the list. Click **OK**.

   - To add a role, click **Roles** and select a role from the list. Click **OK.**

Step 6. Configure input fields and output fields

On the **Fields** tab of the human task asset, you can specify all the input field and output field properties such as the field name, field type, field description, and default values. You can also specify whether the configured fields are required.

Configure the input fields to display information that the task owners will need to act on a human task. Configure the output fields that the task owners can use to provide information about the human task.
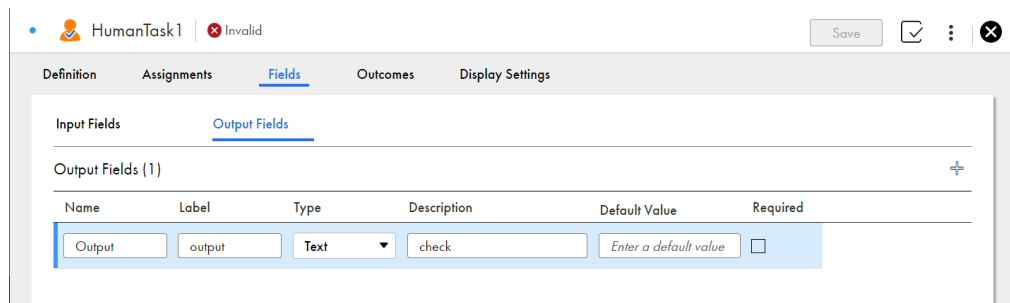
1. Click the **Fields** tab.

   The **Input Fields** section appears as shown in the following image:

2. To add input fields, perform the following steps:

   a. Click **+** to add an input field.

   b. Enter a name for the input field.

   c. Enter a label for the input field.

d. Select the field type from the list. You can select the integer, checkbox, text, date, date time, number, time, text area, and object. If you select the type as **Object**, you can select a process object. Process objects help in grouping information. For example, instead of creating separate fields for name, address, phone number, and so on, you can create one process object with different fields that contain demographic data. For more information about creating process objects, see *Designing Process Objects*. For rules and guidelines that you need to consider while using process objects in human task assets, see "Rules and guidelines for using process objects" on page 338.

e. Optionally, enter a description for the field.

f. In the **Default Value** field, enter the most used value for the field or provide guideline text about how to configure the field.

g. Select the **Required** check box to set the input field as required for the task.

3. Click the **Output Fields** section.

The **Output Fields** section appears as shown in the following image:

4. To add output fields, perform the following steps:

a. Click **+** to add an output field.

b. Enter a name for the output field.

c. Enter a label for the output field.

d. Select the field type from the list. You can select the integer, checkbox, text, date, date time, number, time, text area, and object. If you select the type as **Object**, you can select a process object. Process objects help in grouping information. For example, instead of creating separate fields for name, address, phone number, and so on, you can create one process object with different fields that contain demographic data. For more information about creating process objects, see *Designing Process Objects*. For rules and guidelines that you need to consider while using process objects in human task assets, see "Rules and guidelines for using process objects" on page 338.

e. Optionally, enter a description for the field.

f. In the **Default Value** field, enter the most used value for the field or provide guideline text about how to configure the field.

g. Select the **Required** check box to set the output field as required for the task.

Rules and guidelines for using process objects

Consider the following rules and guidelines when you use a process object in a human task asset:

* You can add a maximum of 10 process objects in the input fields and output fields altogether.

* The selected process object can contain a maximum of 5 nested objects.

* The selected process object can contain a maximum of 50 input fields and output fields including nested objects.
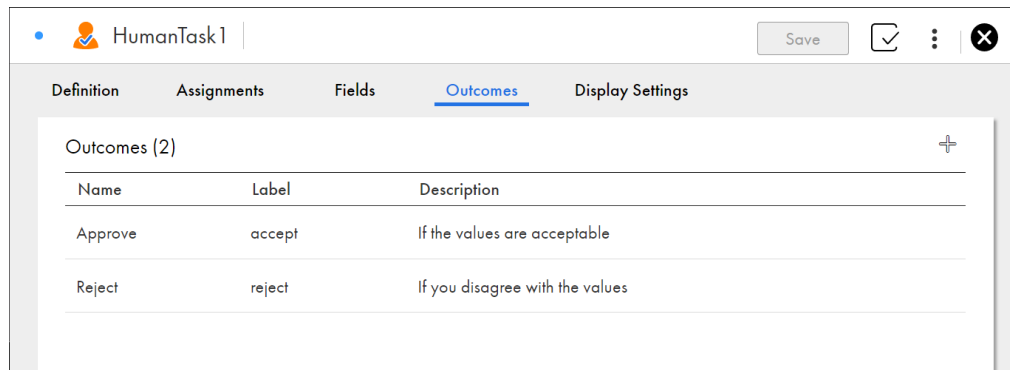
- The selected process object can't contain any cyclic reference.

- The selected process must contain at least one item.

Step 7. Configure task outcomes

On the **Outcomes** tab of the human task asset, you can specify how the task must be completed. You can create a list of possible decisions that a task owner can take on the task. For example, you can add **Approve** and **Reject** as possible outcomes for a task.

1. Click the **Outcomes** tab.

    The following image shows the **Outcomes** tab:

2. Click **+** and add the outcomes that must appear in the Human Task Inbox of the Human Tasks service.

Step 8. Configure display settings

On the **Display Settings** tab of the human task asset, you define the settings that the task uses in the Human Task Inbox. You can specify properties such as the display name for the task, task subject, and task description.

You can use a guide rendering method or default rendering method to display the fields in the Human Task inbox. If you choose a guide as a rendering method, the task uses the screens defined in the guide to display the fields. If you do not choose a guide as a rendering method in the **Guide** field, the task will use the default rendering method to display the fields in the Human Task Inbox.

There are two ways in which you can use the guide rendering method for the human task. You can either select a guide or generate a new guide that the task must use for rendering in the Human Task Inbox. If you click **Browse**, you can select an existing guide in Application Integration. However, the input fields, output fields, and outcomes of the selected guide must match the values that you specify in the human task asset.
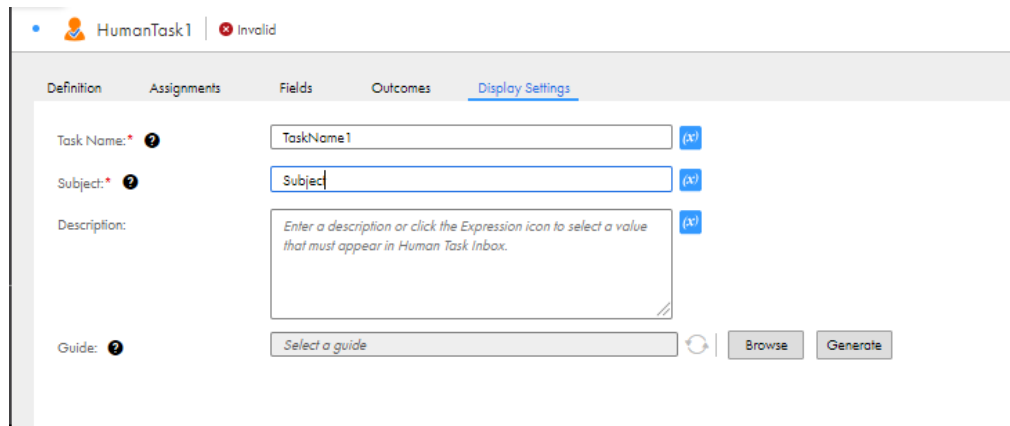
You can create a new guide by clicking **Generate**. With this feature, you can generate a guide with pre-populated values in the input fields, output fields, and outcomes based on the values that you define in the asset. You can further configure, save, and publish the guide to use it in the human task.

Note: When you generate a new guide, the output field will not appear as mandatory in the guide designer even if you have selected the **Required** field for the output field in the human task asset. You will be able to

complete the task successfully and continue with the next step in the process even if you do not provide a value for a required output field.

1. Click the **Display Settings** tab.

    The following image shows the **Display Settings** tab:

2. Enter a name for the human task that the process generates. Optionally, you can select the value by clicking the Expression Editor icon next to the **Task Name** field.

3. Enter the task subject that must be displayed in the Human Task Inbox. Optionally, you can select the value by clicking the Expression Editor icon next to the **Subject** field.

4. Optionally, enter the task description that must appear in the Human Task Inbox. You can also select the value by clicking the Expression Editor icon next to the **Description** field.

5. Optionally, select an existing guide or generate a new guide that you want to map to the human task in the **Guide** field. Click the **Generate** button and click **OK** on the confirmation message to generate a new guide.

6. You can further configure the guide as per the requirement. After you have generated a new guide, you must save and publish the new guide to use it in the human task. Any error that occurs during the guide generation will appear as a validation error in the human task asset.

Using a human task in a process

After you create a human task asset, you can use it in a process where human intervention is needed.

1. Drag a Human Task step to the process and select the human task asset that you created.
    For more information, see "Human Task Step" on page 127

2. Configure the other process steps as needed.
    You can also use human task utility functions when working with XQuery in Process Designer or in any field that uses a formula to set a value. For more information, see "Human task XQuery utilities " on page 101

3. Publish and run the process on Cloud Server.

    When the process runs the Human Task step, it generates the human task and sends it to the Human Task Inbox of the Human Tasks service. The assigned users can then view and act on the task.

    After the task is complete, the process receives a callback and continues with the subsequent steps. After a process completes, you can view the Human Task step execution details in Application Integration Console.

For more information about the Human Task Inbox, see the Human Tasks help.

INDEX