# Solutions Business Manager
SBM Orchestration Guide

Part number: Product version: 11.7.1

Publication date: 2020-02-27

# Table of Contents

# Part 1: Basic Orchestration Topics

This section contains the following information:

# Chapter 1: Orchestration Concepts

This section describes the concepts related to orchestrations created in SBM Composer.

## About Orchestration Workflows

Orchestrations are created in SBM Composer. They are containers for design elements such as orchestration workflows.

The primary purpose of an orchestration workflow is to enable the use of Web services for coordinating the interaction between an application workflow and one or more external systems. This lets the application workflow present data that can be exchanged and modified in these external systems. Orchestration workflows can also perform modifications on the data that flows within the application workflow.

**Note:** Orchestration workflows are used to automate processes, while application workflows are generally manual processes for users. Typically, users never see a process controlled by an orchestration workflow, but they must interact with an application workflow.

An orchestration workflow is an arrangement of control flow structures, Web services, and data elements.

- Control flow structures enable an orchestration workflow to make calculations, decide between two possible sets of instructions, handle exceptions, and so on. In SBM

Composer, these structures are known as *steps*, and they are listed on the **Step Palette** of the orchestration workflow editor.

- Web services can be called by means of the **Service** step, which is also listed on the **Step Palette.** Web services let orchestration workflows query and update data in application workflow items and in external products.
- Data elements include data sent with the event, data expected by or returned by a Web service, and working data created for temporary use during the execution of an orchestration workflow. Data elements are mapped in the step Property Editor.

## Comparing Synchronous With Asynchronous Orchestration Workflows

Orchestration workflows can run *synchronously* or *asynchronously*. An orchestration workflow that runs synchronously immediately returns the data to an application. However, an orchestration workflow that runs asynchronously can keep going, independently of the application that contains it.

There are situations in which asynchronous (not synchronous) orchestration workflows should be used. For information, see Usage [page 175].

The following table lists the differences between synchronous and asynchronous orchestration workflows.

| **Synchronous Orchestration Workflow** | **Asynchronous Orchestration Workflow** |
|---|---|
| Used when an immediate reply is required. For example, a synchronous orchestration workflow might be used when certain data is needed before a user can transition to the next step in the application workflow, or when a Web service call is expected to return a quick reply such as a stock quote or a weather forecast. | Used when an immediate reply is not required. For example, an asynchronous orchestration workflow might be used when the orchestration contains a long-running program, when the data that is returned by the Web service is required in a later step in the application workflow, or when the orchestration workflow performs some unrelated task such as sending an e-mail. |
| Requires the application workflow to wait for a reply before it can continue. | Does not require the application workflow to wait for a reply. Runs independently of the application workflow. |
| Can be used for transition actions and for state actions. | Can only be used for transition actions. |
| The SBM Application Engine invokes the SBM Orchestration Engine directly. The orchestration workflow can only be called by the SBM Application Engine. | The SBM Application Engine calls the Event Manager, which then invokes the SBM Orchestration Engine based on event mappings in the event definition. |

**Tip:** In SBM Composer, distinct icons let you quickly determine whether an orchestration workflow is synchronous or asynchronous. In addition, the orchestration workflow type is shown on the **General** tab of the orchestration workflow Property Editor.

In the exercises in Chapter 6: Orchestration Tutorial [page 185], you will create a process app that contains both synchronous (event with reply) and asynchronous (event without reply) orchestration workflows. When you run the process app, you will be able to see the differences in behavior between them.

**Note:** The **Web Service Invocation Timeout** setting in the **Database** tab in SBM System Administrator is used to control the timeout for synchronous orchestrations. The timeout value that you specify controls the amount of time that the system will wait for an synchronous orchestration to complete. Prior to SBM 10.1.3, the SBM Orchestration Engine would timeout the response from a synchronous orchestration at sixty seconds.

## About Subroutines

A subroutine is a workflow that you can call from multiple orchestration workflows. Subroutines can contain any orchestration elements, such as the Calculate step, and are useful when your orchestrations contain common actions. Once you have created a subroutine, you can use it by dragging it from the **Step Palette** into an orchestration workflow.

You can define inputs for a subroutine workflow, similar to defining Working Data, and you can define outputs for a subroutine in the **Data Mapping** tab of the **End** step.

Like Web services, you can then map these inputs within the calling orchestration to the appropriate source elements. You can use outputs in mapping and expressions of other steps.

Note the following points about subroutines:

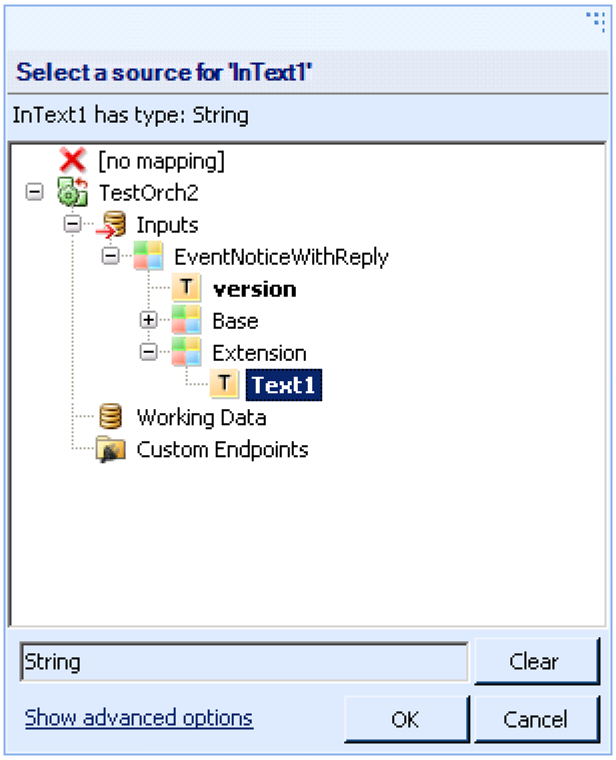
- Subroutines can be used in both asynchronous and synchronous orchestrations.
- You can define as many subroutines as you want and nest them as needed.
- A subroutine cannot call itself either directly or indirectly via another subroutine.

### Example

This simple example demonstrates how to create a subroutine that appends the text "-subroutine" to a field.

1. Create an application process app and add a new text field called `Text1`.
2. In the application workflow, create a quick transition called `Test Subroutine` that starts and ends at the **New** state.
3. Add an action to the **Test Subroutine** transition that does the following:
   - Calls a new synchronous orchestration before the transition occurs
   - Includes **Text1** as a field that is used and returned by the event
4. In app explorer, right-click **Orchestration Workflows** and select **Add New Subroutine**.
5. Change the default name to `AppendTextSubroutine`.
6. From the **Step Palette**, drag a **Calculate** step to the subroutine.
7. Select the subroutine workflow. For the data mapping inputs, add the string `InText1`.
8. Select the **End** step. For the data mapping outputs, add the string `OutText1`.
9. Select the **Calculate** step, and create a new assignment with the following values:
   - **Target:** `OutText1`
   - **Expression:** `CONCAT(InText1, "-subroutine")`
10. Select the orchestration you created in step 2.
11. Drag the new subroutine step **AppendTextSubroutine** to the orchestration.

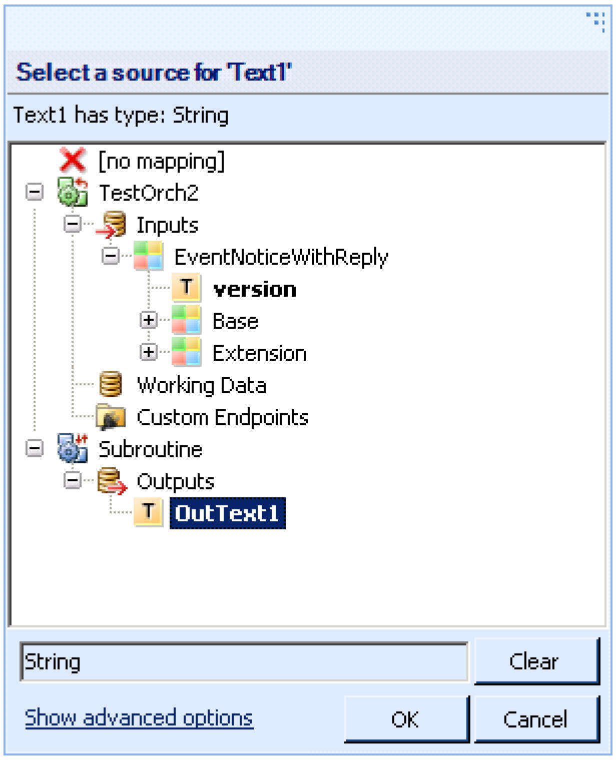
12. Select the subroutine step and map the **InText1** input to the following source:



Select a source for 'InText1'

InText1 has type: String

13. Select the **End** step and map the **Text1** output to the following source:

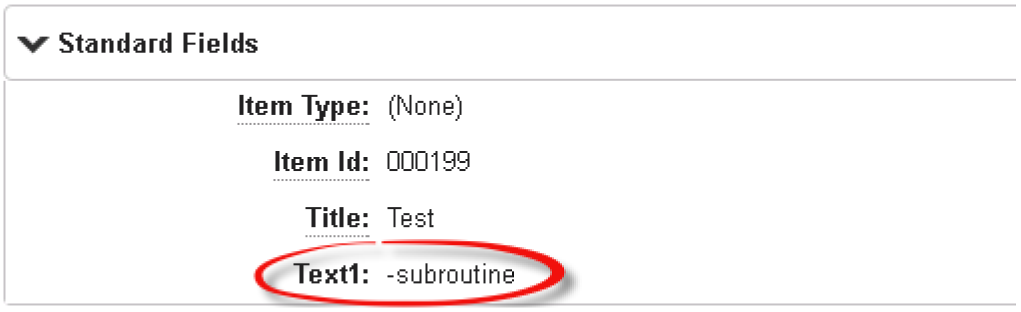


Select a source for 'Text1'

Text1 has type: String

14. Save and deploy the process app.

To test in SBM Work Center, submit a new item and then click the **Test Subroutine** button. The **Text1** field should display the following:



Standard Fields

| | |
|---|---|
| Item Type: | (None) |
| Item Id: | 000199 |
| Title: | Test |
| Text1: | -subroutine |

## About Working Data

You can see the working data hierarchy on the **Data Mapping** tab of the orchestration workflow Property Editor. Because the working data is available throughout the orchestration workflow, it is displayed when no specific step is selected.

Working data is composed of individual data elements that you create and delete. The possible types are:

- Private Simple: A type such as Integer or String.
- Library Type: A system type such as User, Field, or Privilege.
- Private Complex: A structure made of some combination of the other two types. This type has child data elements.

Working data is used in orchestration workflows. For example, you might want to temporarily store the value returned by a Web service or a loop counter variable. As with the inputs to Web services, you can define source elements and default values for working data.

**Important:** Working data elements cannot be named `input` or `output`.This includes any variations in capitalization such as `Input` or `Output`.

**Note:** For string type data elements, you must use escape sequences to add a new line, insert a tab, enter a carriage return, or type a literal backslash. See Using Escape Sequences [page 21] for details.

### Resetting Data

For loops using the For Each or While step, the Web service and subroutine inputs retain the data for all transactions. In some cases, this caching may cause issues for subsequent loops. To reset the data collected in these inputs, first use one of the following methods to define an empty variable:

- From the data mapping inputs of the Web service or subroutine, copy the input structure (or its parts) that you want to reset, and paste it in the working data.
- Likewise, for output data, turn on the properties mode and copy the output structure (or its parts) that you want to reset, and paste it in the working data.

- To reset an array item, copy and paste it to the working data. In the properties mode, change IsUnbounded to False.
- In the Web service, identify the named type of the input that you want to reset. In the working data, create a variable that matches the named type from the type library.

Once you have defined the empty variable, assign it to the corresponding part of the input data, just prior to initiating the loop via the Calculate step for each iteration (inside of the loop).

## About Data Mapping

As you create orchestrations in SBM Composer, you find that you need to pass data to Web services, use the return values from Web services, or store data values in temporary locations.

In SBM Composer, you use data mapping to arrange the inputs and outputs of Web services. **Data Mapping** is one of the tabs that is displayed in the orchestration workflow Property Editor when you select an orchestration workflow in App Explorer or when you select a **Service** step in an orchestration workflow. (As with all Property Editor tabs, the content of the **Data Mapping** tab depends on what is selected.) When you define inputs, you can either use the mappings suggested by SBM Composer or specify other mappings.

You can do any of the following tasks from the **Data Mapping** tab:

- Choose an input suggested by SBM Composer (suggested mappings).
- Open the **Select a Source** popup, in which you select an input from a hierarchical list of choices.

  **Note:** The outputs from a Web service are available as source elements only for subsequent items in the orchestration workflow.

- Display advanced options with which you map incompatible types or multiple items in the **Select a Source** popup.

  **Note:** If incompatible types are mapped, the mapping appears in a warning color (dark blue) preceded by a ! character.

- Use data that accompanies the event that triggered the orchestration workflow.
- Clear mappings for the selected item.
- Set the display of data elements to show a red highlight on the icons of required items.
- Switch between mapping mode and properties mode. In properties mode, you can view and edit the lowest-level details for a working data element or step input, including its namespace and type.
- Initialize values for optional elements and attributes in complex types. For more information, see Element and Attribute Mapping [page 23].

**Note:**

- For string type data elements, you must use escape sequences to add a new line, insert a tab, enter a carriage return, or type a literal backslash. See Using Escape Sequences [page 21] for details.
- Elements on the **Data Mapping** tab that have a lock icon on them are external types that are defined by a Web service; they are not defined in the process app. These external types are shown in the Type Library Editor [page 55]. You cannot change the type of locked elements or change their child elements.

## About Value Assignment

Values are held in variables that exist for the entire execution of an orchestration workflow. These variables include the following:

- Working data variables
- The asynchronous orchestration workflow **EventNotice** or the synchronous orchestration workflow **EventNoticeWithReply** and **EventNoticeWithReplyResponse**
- Request and response messages for each **Service** step in the workflow

A working data value can be assigned on the **Data Mapping** tab in the orchestration workflow Property Editor. Alternatively, a **Calculate** step can be used to assign a value to a working data variable.

The **EventNotice** or **EventNoticeWithReply** variables will contain the values sent when the orchestration workflow is invoked, but can be overwritten by **Calculate** steps. The **EventNoticeWithReplyResponse** can be assigned using the **Data Mapping** tab in the **End** step Property Editor.

A **Service** step input variable can be assigned on the **Data Mapping** tab for the step or by a **Calculate** step that specifies the input to a **Service** step as its target. A **Service** step output variable is normally assigned by the **Service** step on its return. However, a **Calculate** step can be used to set or override a value in the **Service** step output variable.

The **Data Mapping** tab includes a **Source elements** column, where values are mapped from other variables. It also includes a **Default value** column, where a constant value can be specified.

Values are assigned to **Service** steps as follows:

1. When an orchestration workflow is invoked, its first action is to allocate the event variables, the working data, and the **Service** step input and output message variables. It does this by assigning any default values that have been specified. Its next action is to perform mappings to working data, overwriting any default values with the respective mapped values.

2. When the workflow reaches a **Calculate** step, values assigned by that step replace either the initial values as described above or the values set by previous **Calculate** steps.

3. When the workflow reaches a **Service** step with mappings, the mappings replace either the initial values as described above or the values set by previous **Calculate** steps.

**Note:** An error will result if a source for a **Service** step mapping is not initialized. To prevent this, make sure you either provide a default value for mapped working data or use a **Calculate** step to provide the value.

A data element with a complex type has a structure that includes a parent element and child elements. When you map a parent element to a source parent element, its child elements are replaced by a copy of the source child elements. Because the mappings copy the source data structure, you can explicitly assign different values to the child elements in the new structure without changing values in the parent. For details about complex types, see About Complex Types and Namespaces [page 23].

## Using Escape Sequences

Escape sequences are characters that change the way subsequent characters in a string are interpreted by a program. The following table lists escape sequences are available when you specify default values for string type data elements in working data or **Service** step data mapping. (The numeric value of the characters encoded by each escape sequence is included.)

| Name | Escape Sequence | ASCII Value |
|---|---|---|
| Tab | \t | 9 |
| Newline | \n | 10 |

| Name | Escape Sequence | ASCII Value |
|---|---|---|
| Carriage return | \r | 13 |
| Backslash | \\ | 92 |

For example:

- To add a new line, type `\n` before you type the characters in the new line. For example, `First line text\nSecond line text`.
- To type a literal backslash, such as in a directory path, type a double backslash. For example, `C:\\Program Files\\Serena\\SBM\\Composer`.

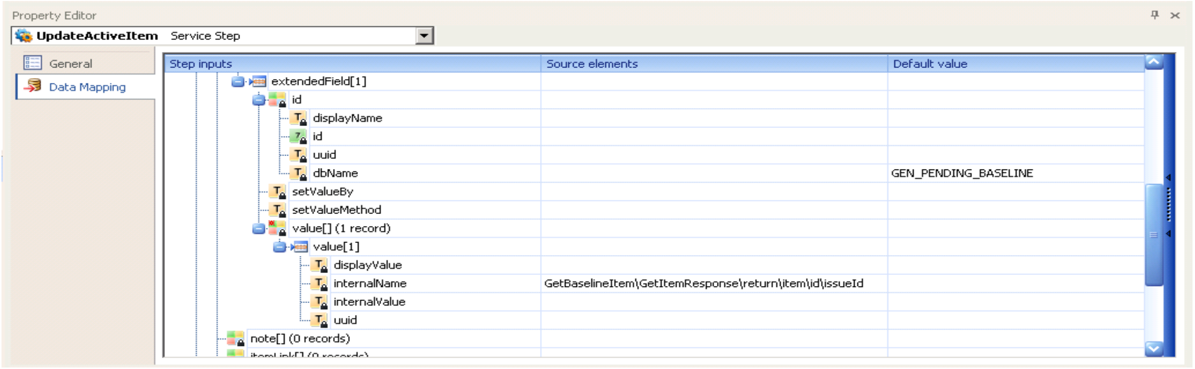
The escape sequence characters will be visible in the **Default value** column on the **Data Mapping** tab of the workflow or step Property Editor, but the text will be represented correctly at runtime.

**Note:** You cannot use a backslash outside the context of an escape sequence; doing so will generate an error message.

## About ExtendedField

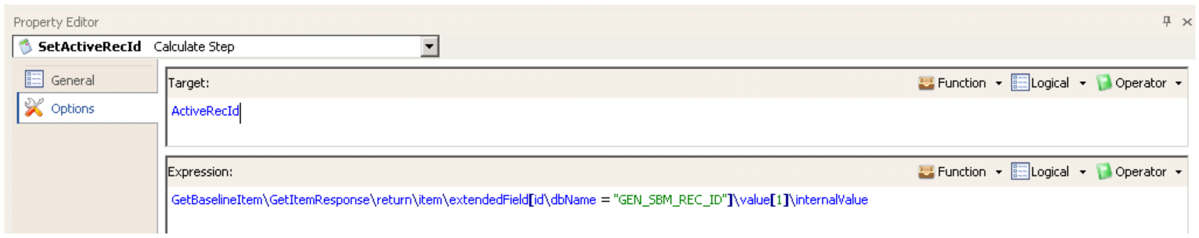
The ExtendedField parameter is available on some operations in SBM Web services. The `extendedField[]` extension provides a way to interact with the various custom fields defined in an application. You can specify something that identifies the field (for example, its name) and a value for the field by mapping to a value or by specifying a default value.

The following illustration shows the ExtendedField input in a **Service** step that uses the "TransitionItem" operation. It creates an update record for the *GEN_PENDING_BASELINE* field and sets its value to the item ID.



| Step inputs | Source elements | Default value |
|---|---|---|
| extendedField[1] | | |
| id | | |
| displayName | | |
| id | | |
| uuid | | |
| dbName | | GEN_PENDING_BASELINE |
| setValueBy | | |
| setValueMethod | | |
| value[] (1 record) | | |
| value[1] | | |
| displayValue | | |
| internalName | GetBaselineItem\GetItemResponse\return\item\id\issueId | |
| internalValue | | |
| uuid | | |
| note[] (0 records) | | |

ExtendedField can also be used in **Calculate** step expressions. The following illustration shows a **Calculate** step that takes the internal value of the *GEN_SBM_REC_ID* field and stores it in the `ActiveRecId` working data element.



SetActiveRecId Calculate Step

Target:

ActiveRecId

Expression:

GetBaselineItem\GetItemResponse\return\item\extendedField[id\dbName = "GEN_SBM_REC_ID"]\value[1]\internalValue

**Note:** It is recommended that you create the array elements directly in the **Service** steps and assign values to them directly, instead of trying to dynamically create them.

## About Complex Types and Namespaces

Complex types differ from simple types in that complex types can have child data elements and attributes, while simple types cannot. Complex types can be named or anonymous. All types except anonymous types are identified by their name and target namespace. Anonymous complex types do not have names.

You can view the name and namespace for a data element by switching from mapping mode to properties mode on the **Data Mapping** tab in the Property Editor for an orchestration workflow or **Service** step. To switch modes, click the vertical bar near the right side of the tab.

You can use the Select Library Type Dialog Box [page 55] to associate a named type with a data element. This dialog box contains the named types defined by the Web services that were imported into the orchestration workflow. In the orchestration workflow Property Editor, the **NamedType**, **NamedType Namespace**, and **Namespace** are read-only. You cannot manually add a child element to a data element with a named type, even if it is a complex type.

Note the following points about anonymous complex types:

- The default target namespace is `http://`*`workflow name`*.
- Unless it is changed explicitly, a child data element continues to have the default namespace, even if the namespace of its parent data element is changed explicitly.
- If you want a child data element to store data from a Web service response, its namespace must be the same as the namespace in the Web service schema. In other words, it must be the same as the `targetNamespace` of the schema element defined in your WSDL file. The schema element must contain the complex type from which you are trying to map.

### Element and Attribute Mapping

In an XML schema, by default, elements are required and attributes are optional. If an element or attribute is optional, no element or attribute is created; therefore, the element or attribute will have no value. If you need to map that element or attribute, you must initialize it with a value by doing one of the following:

- Provide a default value for it in the working data for the orchestration workflow.
- Use a **Calculate** step to assign it a value.
- Select a source element as an input value. (See About Data Mapping [page 19] for details.)

SBM Composer does not distinguish elements from attributes; both are presented as data elements in the **Data Mapping** tab of an orchestration workflow or **Service** step Property Editor. To see elements and attributes, refer to the original schema definition, which is imported into SBM Composer as a service WSDL file. In this file, an attribute is declared as part of a complex type. The `use` attribute is used to specify that it is required. For example:

```
<xs:attribute name="myattr" type="xs.string" use="required"/>
```

If an attribute is contained within a named type, the attributes will be presented as child data elements on the **Data Mapping** tab, and you can initialize them as described above. However, if an attribute is contained within an anonymous type, it will not appear on the **Data Mapping** tab, and you will not be able to create it there as a child of a data element. To work around this situation, do the following:

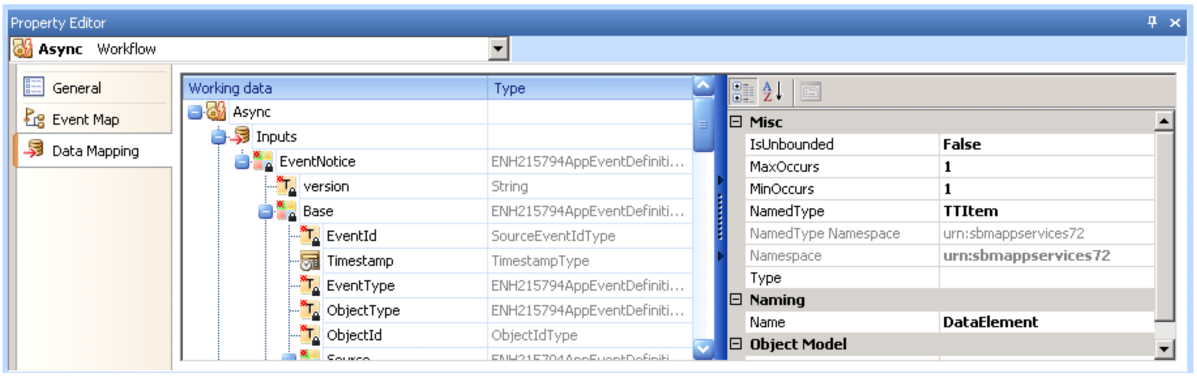
1. Open the service WSDL file to edit it.
2. Create a named type in the appropriate namespace.
3. Create the attribute on the appropriate element.
4. Reimport the WSDL file into SBM Composer.
5. Initialize the attribute using one of the three methods listed above.

## Example: Named Type

A complex data element associated with a named type inherits its namespace and children data elements from the named type. For example, suppose you do the following:

1. Add the **sbmappservices72** Web service to the process app. To do so, right-click the **Web Services** heading in App Explorer, and select **Add New Service**. In the **Web Service Configuration** dialog box that opens, navigate to the `sbmappservices72.wsdl` file, and then click **OK**. This file is in the *`installDir`*`\Application Engine\webservices\bin` directory.
2. In the orchestration workflow Property Editor, select the **Data Mapping** tab.
3. Right-click the **WorkingData** step input, select **Add New**, and then select **Select from Libray Type**.
4. In the Select Library Type Dialog Box [page 55] that opens, find and select **TTItem**) and then click **OK**.

   As shown in the following illustration, the **Namespace** value is inherited from the **NamedType Namespace** value.



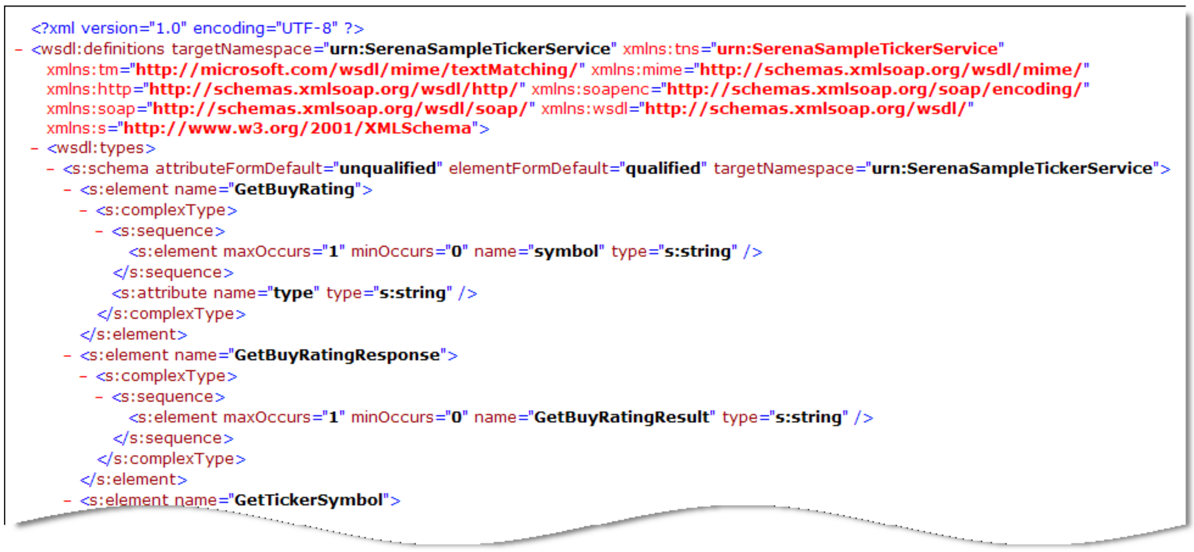
| Working data | Type |
|---|---|
| Async | |
| Inputs | |
| EventNotice | ENH215794AppEventDefiniti… |
| version | String |
| Base | ENH215794AppEventDefiniti… |
| EventId | SourceEventIdType |
| Timestamp | TimestampType |
| EventType | ENH215794AppEventDefiniti… |
| ObjectType | ENH215794AppEventDefiniti… |
| ObjectId | ObjectIdType |

| | |
|---|---|
| **Misc** | |
| IsUnbounded | **False** |
| MaxOccurs | **1** |
| MinOccurs | **1** |
| NamedType | **TTItem** |
| NamedType Namespace | urn:sbmappservices72 |
| Namespace | **urn:sbmappservices72** |
| Type | |
| **Naming** | |
| Name | **DataElement** |
| **Object Model** | |

## Example: Anonymous Type

For a complex data element with an anonymous type, you must create the structure manually and specify the correct namespace. You get the information you need from the WSDL file for the Web service.

The following illustration of shows part of a WSDL file. This file includes the target namespace (`urn:SerenaSampleTickerService`) and some of the elements that can be added to the structure (`GetBuyRating`, `GetBuyRatingResponse`, `GetTickerSymbol`).



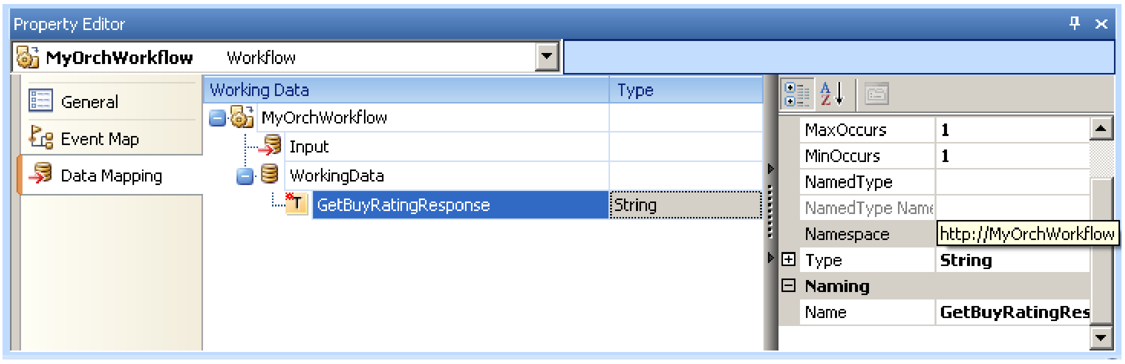
```
  <?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="urn:SerenaSampleTickerService" xmlns:tns="urn:SerenaSampleTickerService"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:s="http://www.w3.org/2001/XMLSchema">
  - <wsdl:types>
    - <s:schema attributeFormDefault="unqualified" elementFormDefault="qualified" targetNamespace="urn:SerenaSampleTickerService">
      - <s:element name="GetBuyRating">
        - <s:complexType>
          - <s:sequence>
              <s:element maxOccurs="1" minOccurs="0" name="symbol" type="s:string" />
            </s:sequence>
            <s:attribute name="type" type="s:string" />
          </s:complexType>
        </s:element>
      - <s:element name="GetBuyRatingResponse">
        - <s:complexType>
          - <s:sequence>
              <s:element maxOccurs="1" minOccurs="0" name="GetBuyRatingResult" type="s:string" />
            </s:sequence>
          </s:complexType>
        </s:element>
      - <s:element name="GetTickerSymbol">
```

Suppose you want to store the `GetBuyRatingResponse` value and use it later in the orchestration workflow. You would perform the following steps:

1. Add the **SerenaSampleTickerService** Web service to the process app. To do so, right-click the **Web Services** heading in App Explorer, and then select **Add New Service**. In the **Web Service Configuration** dialog box that opens, in the **WSDL** box, type **`http://`*`serverName`*`:8085/Ticker/services/SerenaSampleTickerService?wsdl`** and then click **OK**.

2. Add a **Service** step to the orchestration workflow. On the **General** tab in the Property Editor for the step, select **SerenaSampleTickerService** from the **Service** list, and select **GetBuyRating** from the **Operation** list.

3. Add a new data element (complex type) to the **WorkingData** node and name it **`GetBuyRatingResponse`**.



Property Editor

MyOrchWorkflow Workflow

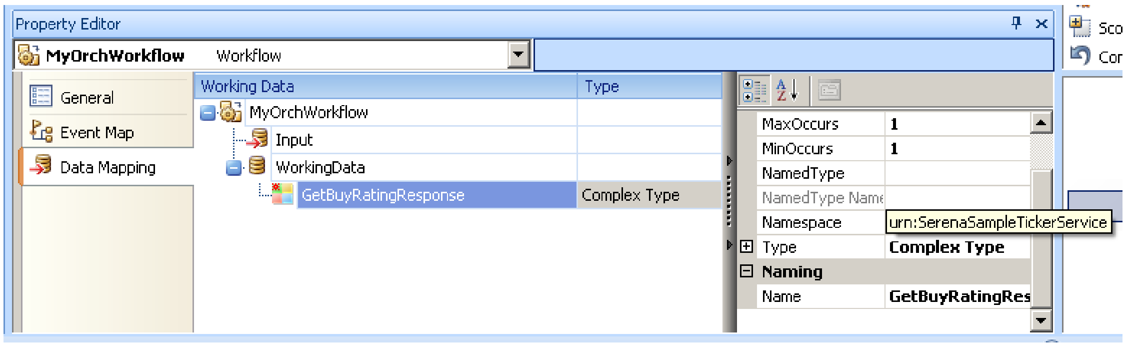
General
Event Map
Data Mapping

| Working Data | Type |
|---|---|
| MyOrchWorkflow | |
| Input | |
| WorkingData | |
| GetBuyRatingResponse | String |

| | |
|---|---|
| MaxOccurs | 1 |
| MinOccurs | 1 |
| NamedType | |
| NamedType Name | |
| Namespace | http://MyOrchWorkflow |
| Type | String |
| Naming | |
| Name | GetBuyRatingRes |

The default namespace is **http://MyOrchWorkflow** (the name of the orchestration workflow).

4. Change the namespace to the `targetNamespace` shown in the WSDL file (`urn:SerenaSampleTickerService`). To do this, simply copy and paste the namespace into the **Namespace** box.



Property Editor

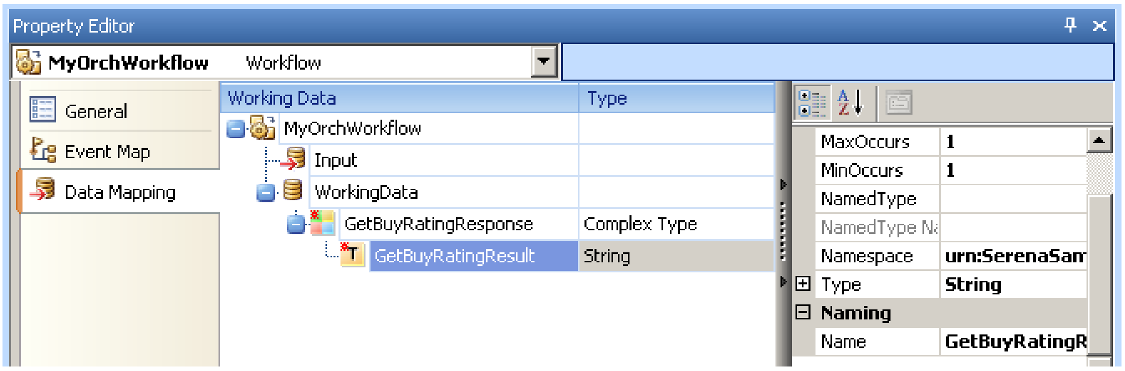
MyOrchWorkflow Workflow

General
Event Map
Data Mapping

| Working Data | Type |
|---|---|
| MyOrchWorkflow | |
| Input | |
| WorkingData | |
| GetBuyRatingResponse | Complex Type |

| | |
|---|---|
| MaxOccurs | 1 |
| MinOccurs | 1 |
| NamedType | |
| NamedType Name | |
| Namespace | urn:SerenaSampleTickerService |
| Type | Complex Type |
| Naming | |
| Name | GetBuyRatingRes |

5. Add a child data element and name it **`GetBuyRatingResult`**.



Property Editor

MyOrchWorkflow Workflow

General
Event Map
Data Mapping

| Working Data | Type |
|---|---|
| MyOrchWorkflow | |
| Input | |
| WorkingData | |
| GetBuyRatingResponse | Complex Type |
| GetBuyRatingResult | String |

| | |
|---|---|
| MaxOccurs | 1 |
| MinOccurs | 1 |
| NamedType | |
| NamedType Na | |
| Namespace | urn:SerenaSam |
| Type | String |
| Naming | |
| Name | GetBuyRatingR |

The namespace was inherited from the parent data element. The namespace matches the `targetNamespace` in the WSDL file.

**Important:** If the namespace does not match the `targetNamespace` in the WSDL file, you must change it manually.

## About Events

An event signals a meaningful change from an SBM application or an external product. For example:

- An issue defect management application in SBM raises an event every time a user submits a new defect.
- A software build product raises an event every time a build completes.
- Salesforce.com raises an event every time a potential customer is initially contacted.

Any SBM application or external product that is capable of calling a Web service can raise events in SBM. After an event is raised, the Event Manager receives it, and then calls the SBM Orchestration Engine to execute the asynchronous orchestration workflow linked to the event. The orchestration workflow could update an SBM item, create a new SBM item, and so on.

The orchestration workflow provides the integration point between an SBM application and an external product, or two SBM applications in the same process app or in different process apps. In the first case, the orchestration workflow can be initiated from either the application or the external product. In the second case, the orchestration workflow can be initiated from either application. Something happens in one that raises an event, and the orchestration workflow runs in response to the event and updates the other.

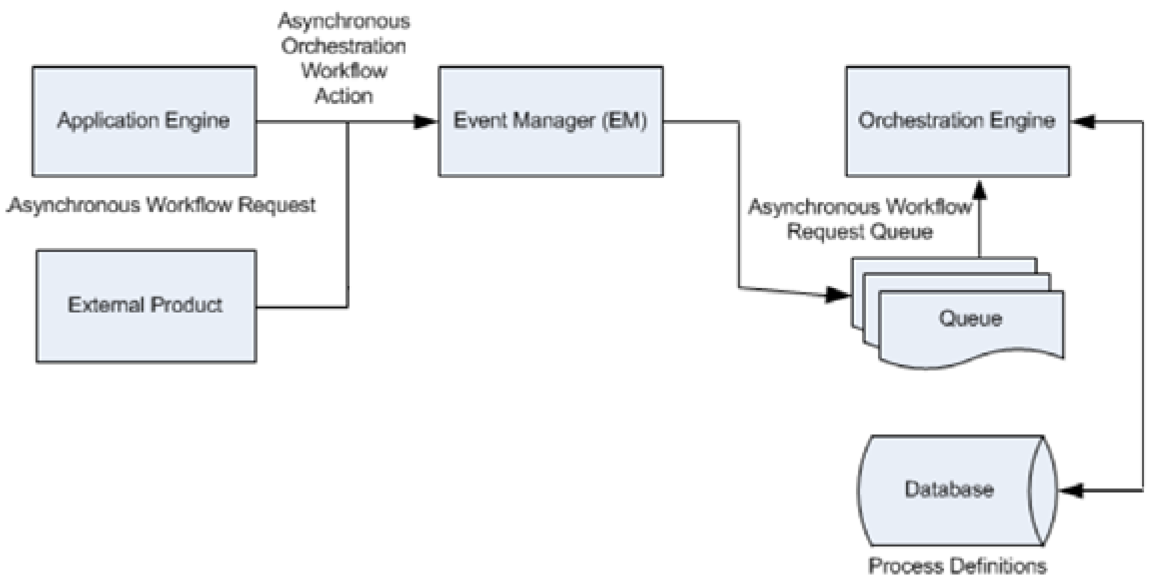
An event can be raised in the following ways:

- From the SBM Application Engine by creating an asynchronous orchestration workflow action on a transition in an application workflow. You use the **Action Wizard** to create this type of event. You can raise the application's standard **Event without Reply** orchestration link ("the local event") to invoke orchestration workflows that were defined for use with the application, or use an imported orchestration link ("external event") to invoke any other orchestration workflow. For more information, see the action wizard information in the *SBM Composer Guide*.

  **Note:** When there are multiple asynchronous orchestration actions on the same transition, only one event of each event type is raised. Events start orchestration workflows that will run simultaneously depending on available resources, so the ordering of event actions on the **Actions** tab has no effect on the execution order of the orchestration workflows. See the "Considerations for Using Actions" topic in the *SBM Composer Guide* for details.

- From an external product using any of the following:
  - Event Manager Web service API
  - E-mail message

The following diagram illustrates the way an event is processed.


Asynchronous Orchestration Workflow Action
Application Engine
Event Manager (EM)
Orchestration Engine
Asynchronous Workflow Request
Asynchronous Workflow Request Queue
External Product
Queue
Database
Process Definitions

**Remember:** Events are relevant for asynchronous orchestration workflows only. For information about the differences between asynchronous and synchronous orchestration workflows, see Comparing Synchronous With Asynchronous Orchestration Workflows [page 14].

In addition to initiating an orchestration workflow, an event sends data to it.

- For an event raised from an SBM application, the mappings are implicit, so you do not need to map application data into the event. You can select which fields from the primary application table to be sent with the event, and can define additional data to be sent with the event. To do this, click **Event without Reply** under the **Orchestration Links** heading in App Explorer. See About Orchestration Links [page 29] for more information.
- For an event that is not raised by the primary application, you must map application data into the event. In this case, you import an existing event definition or create a new custom event definition using SBM Composer. See About Application Links and Event Definitions [page 28] and External Events [page 30] for more information.

In certain external integrations with SBM, errors are handled in a limited way and cannot be automatically retried. You can manually retry these events in SBM Application Repository. For more information, see Retrying Failed Asynchronous Events [page 201].

## About Application Links and Event Definitions

An event definition is a specific format that lets SBM applications and external products declare the events they can raise, and lets receiving applications understand these events so they can respond to them. Event definitions are listed under the **Application Links** heading in App Explorer.

An event definition includes data specific to the application or external product, and the following event values: `EventType`, `ObjectType`, `Product`, `ProductVersion`, and `ProductInstance`. SBM uses the event definition to construct an event map that is deployed to the Event Manager. At runtime, the Event Manager receives the event, and if the event is in a deployed event map, it invokes the associated asynchronous orchestration workflow, passing the event definition data to the workflow.

**Important:** Orchestration links, not event definitions, are used to define events for synchronous orchestration workflows. For more information, see About Orchestration Links [page 29].

Event definitions can be created in two ways:

- **Automatic:** If you use the **Action Wizard** to create a new orchestration, a new event definition is created under the **Application Links** heading. This event definition is designed to be used with the **Event without Reply** that already exists in the application under the **Orchestration Links** heading. The application link lets the orchestration know what the incoming event will look like; the orchestration link lets the application generate an event that looks like what the orchestration is expecting. For more information, see the orchestration workflow information in the "Action Wizard" topics in the *SBM Composer Guide*.

- **Manual:** You can create a new custom event definition or import an existing one. For instructions, see Creating a New Custom Event Definition [page 69] and Importing an Event Definition File for a New Custom Event Definition [page 69]. You typically use custom event definitions to do the following:
  - Export an event definition (`.mtd`) or Web Service Definition Language (`.wsdl`) file from an orchestration and then import the file into any application that needs to raise the event.

    See the "Calling an Orchestration Workflow from Any Application" tutorial in the *SBM Composer Guide* for instructions.

  - Export an `.mtd` or `.wsdl` file from an orchestration and then use it to raise an event from an external product. After you export this file, consult the documentation for the external product to determine how to call an operation on the Web service to raise the event.

    See Raising Events from External Products [page 145] for instructions.

## About Orchestration Links

An orchestration link defines the data that is sent to an orchestration workflow when a transition is executed or a state is reached in an application workflow. The values from the primary table fields that you select and any additional data items you define are passed to the orchestration workflow that you select when you create the orchestration link. For synchronous orchestration workflows, you also select primary table fields whose values should be returned from the orchestration workflow.

**Remember:** Because changes to an orchestration link propagate to the inputs and outputs of an orchestration workflow, the orchestration link is the primary area in which the inputs and outputs are defined.

Orchestration links are displayed directly under the **Orchestration Links** heading in App Explorer.

- **Event without Reply** is added automatically. This defines event data for asynchronous orchestration workflows that are called from an application workflow when a transition is executed. This orchestration link can be exported and then imported into other orchestrations so they can handle the events that the application raises. These orchestrations do not have to be in the same process app.

  **Important:** If you want the "Event without Reply" to be used by an asynchronous orchestration workflow in another process app, lock it before you export it, using the Event Editor [page 40]. The definition includes an event type name (constructed from the application, state, and transition names) and the extension data fields that will be sent with the event. If you lock the definition, it becomes portable, because these names will not change in the event definition, even if the corresponding field names change in the original process app.

- Other orchestration links are for synchronous orchestration workflows that are called from an application workflow when a transition is executed or a state is reached. These orchestration links are typically added when you create a new synchronous workflow through the **Action Wizard**. For more information, see the "Action Wizard" information in the *SBM Composer Guide*.

### External Events

The orchestration links described above define inputs based on primary table fields in the calling application. Another use case is when an orchestration workflow needs to be called by one or more applications with different primary tables. This orchestration workflow could be defined as an interface to a particular application or as an interface to an external product.

To handle this use case, first export the "Event without Reply" from the application or export the event definition from the orchestration. Then import what you exported into the calling application. These orchestration links are displayed under the **External Events** subheading in App Explorer. In the **Action Wizard**, you map the orchestration inputs to the primary table fields of the calling application.

For instructions, see the "Calling an Orchestration Workflow from Any Application" tutorial in the *SBM Composer Guide*.

## About Web Service Calls and Orchestrations

SBM automatically passes security tokens for automated processes such as SBM Web service calls and orchestration workflows. The credentials of the user that invokes the orchestration workflow are automatically supplied to all of the SBM Application Engine Web service calls that are made throughout the orchestration workflow at runtime. This means that the orchestration workflow is invoked under the control of the user's privileges, and the user's name appears in the change history for the affected item.

**Restriction:** An external event could have a security token or a user credential that could be used to obtain a security token, or be anonymous. When the external event is anonymous, there is no security token for the orchestration to pass to the SBM Web services, so authentication credentials must be hard coded in the **auth** element in the orchestration workflow.

The dynamic relationship between the orchestration workflow and the user performing the change not only grants tighter privilege control, but also provides a more detailed audit trail in the affected item's change history. For example, when Bill executes a transition that invokes an orchestration workflow containing the **TransitionItem** Web service operation, the update is performed by Bill's user account under the control of his item privileges. His user credentials are automatically supplied by his security token to the **auth** element for this operation; therefore, the administrator does not need to hard code user credentials in the orchestration workflow ahead of time. If Bill does not have privileges to update the associated item, the **TransitionItem** operation will fail. If Bill does have these privileges, after the transition completes, Bill's user name appears in the change history of the updated item.

An asynchronous orchestration workflow is only executed after the transition that invoked it finishes. For example, suppose a user transitions an item from the **New** state to the **Assigned** state. The asynchronous orchestration workflow is executed after the item is in the **Assigned** state. If the user who initiated the transition no longer owns the item, or does not have the privilege to update items in the **Assigned** state, the orchestration workflow will fail.

There are two ways to handle or prevent these failures:

- In the application workflow, include an intermediate state that waits for the asynchronous orchestration workflow to finish. When it finishes, the item moves to the intermediate state, which can be used to handle failures. For example, this state could have a "return" transition that an administrator can use to move the item back to the original state or to an error state for reprocessing. The original user needs privileges to transition the item to any successful state but should not have privileges to open items that are in the intermediate state or execute the "return" transition.
- Make sure the user performing the transition has privileges to update items in the affected states.

**CAUTION:**

Credentials you hard code in the **auth** element will override security token credentials, so make sure the user has the appropriate privileges for the action that the Web service operation will take. Conversely, if you remove the credentials from the **auth** element, make sure the user who runs the orchestration workflow has sufficient item privileges in SBM.

## About the Step Palette

One of the things that distinguishes the use of Web services in orchestrations, rather than in applications, is the availability of control flow structures. These control flow structures make it possible to design an orchestration workflow that can branch based on a value, make calculations based on returned results, iterate through a set of data, and perform many other tasks.

The control flow structures are available on the area known as the **Step Palette**. Each item in the **Step Palette** is known as a *step*.

The following table describes the steps on the **Step Palette**.

| **Step** | **Purpose** |
|---|---|
| Calculate | Performs a calculation or sets values on data. |
| Decision | Inserts branches into the workflow. |

| Step | Purpose |
|---|---|
| ForEach | Cycles through all members of a complex data element. |
| While | Repeats during the time a certain condition is true. |
| Service | Inserts a Web service into the workflow. |
| Group | Creates a structure to organize steps logically. |
| Scope | Creates a structure to handle faults that occur during the execution of a Web service. |
| Compensate | Defines steps that can compensate for a problem. |
| Throw | Defines steps to throw an exception. |

### About Scope, Compensate, and Throw

The **Scope**, **Compensate**, and **Throw** steps in the **Step Palette** deserve special mention. These steps are related and are used to create a structure for handling faults that occur during Web-service execution.

- The **Scope** step makes it possible to have a FaultHandler and a CompensationHandler.
- The **Compensate** step is designed to work with a CompensationHandler or FaultHandler to roll back actions from an entire scope. You can use the **Compensate** step in one of two ways:
  - If you name a scope explicitly in the step Property Editor, then only the named scope's CompensationHandler is invoked.
  - If you do not name a scope in the step Property Editor, then every CompensationHandler for scopes enclosed immediately within the scope of the **Compensate** step is invoked, in reverse order of execution.
- The **Throw** step is used inside a nested scope to pass the exception or fault to the outer scope.

**Note:** The values of working data are not affected by scopes. Creating a scope does not create an area where working data is only visible within that scope. Working data is global to an orchestration and is visible across all scopes.

**Note:** For detailed information about these steps, see Using the Scope, Throw, and Compensate Steps to Handle Faults From Web Services [page 96].

## About the Expression Editor

Many of the steps on the **Step Palette** include areas that let you define an arithmetic or logical expression. SBM Composer contains a feature to help you write these expressions. This feature is called the *expression editor*.

Expressions might require a long string such as "EventNotice.Extension.ActiveInactive." To reduce typing errors, the expression editor helps you complete these long strings.

The expression editor behaves similarly to the feature in word processors that suggests potential word choices when you begin typing. The expression editor tries to fill in the expression based on what it knows about the hierarchies of working data, Web services, and process app events.

You can see the expression editor by creating a **Calculate** step in an orchestration workflow and then typing a letter into either the **Target** or **Expression** section on the **Options** tab of the step Property Editor. The expression editor tries to match that letter, showing you all available choices.

When you get to the end of a structure element, such as EventNotice, press the period key. The expression editor shows you the choices available in the next level of the hierarchy.

The expression editor also recognizes the XPath functions that are available.

For more information, see the following topics:

Using Data Mapping [page 57]

Using the Step Palette [page 71]

Supported XPath Functions [page 33]

## About Advanced Mapping

By default, the expression editor for the **Calculate**, **ForEach**, **While**, and **Decision/ Branch** steps provides a list of standard choices that make sense for typical data flow. In more advanced use cases, you might want the expression editor to display a list of all available choices.

To display a complete list, select the **Advanced mapping** checkbox on the **Options** tab of any of these steps. For each expression, the list will display additional choices as shown below:

- **Target** expression in the **Calculate** step: Workflow inputs and step outputs
- **Source** expression in the **Calculate** and **ForEach** steps: Step inputs and workflow outputs
- **Rule** expression in the **Decision/Branch** and **While** steps: Step inputs and workflow outputs

Use these choices with caution, as they may lead to unpredictable results.

## Supported XPath Functions

Some of the steps in the **Step Palette** let you define expressions for a calculation or value assignment. In these expressions, you can use any of the functions or operators that are available in the step Property Editor.

The functions in the **Functions** list in the step Property Editor are XPath functions. Only XPath 1.0 functions are currently supported. XPath 2.0 functions are not supported.

**Note:** Some function names in SBM Composer differ from the names in the XPath standard. For example, `NORMALIZESPACE()` is the SBM Composer name for the `normalize-space()` XPath function.

The following table describes the supported XPath 1.0 functions.

| Function | Description |
|---|---|
| BOOLEAN() | Returns a Boolean value for a number, string, or array. Numbers that are not equal to 0 and strings that are not null or empty always return `true`.<br><br>For example, **`BOOLEAN(0)`** returns `false`, and **`BOOLEAN("false")`** returns `true`. |
| CEILING() | Returns the smallest integer that is greater than or equal to the number argument.<br><br>For example, **`CEILING(2.35)`** returns `3`. |
| CONCAT() | Returns the concatenation of two or more strings.<br><br>For example, **`CONCAT("day","light")`** returns "`daylight`". |
| CONTAINS() | Returns `true` if the first string contains the second string. Otherwise, returns `false`.<br><br>For example, **`CONTAINS("XPath","Path")`** returns `true`. |
| COUNT() | Returns the number of elements in an array.<br><br>For example, suppose you have a `Files[]` array under an `Issues` data element. Each array element in the array stores a file name. **`COUNT(Issues.Files)`** returns `5`, because there are five array elements in the array. |
| FLOOR() | Returns the largest integer that is less than or equal to the number argument.<br><br>For example, **`FLOOR(2.35)`** returns `2`. |
| LAST() | Returns the last element in an array.<br><br>For example, suppose you have a `Testers[]` array under a `Users` data element. Each array element in the array stores the name of a software tester. **`Users.Testers[LAST()]`** returns `Jim Wilson`, because his name was stored in the last array element. |
| NORMALIZESPACE() | Returns an argument string after removing leading and trailing spaces and replacing each sequence of spaces with a single space.<br><br>For example, **`NORMALIZESPACE(" 555-1212 ")`** returns "`555-1212`". |

| Function | Description |
|---|---|
| NOT() | Reduces an argument to a Boolean expression, and then returns the opposite value.<br>For example, `NOT(false())` returns `true`. |
| NUMBER() | Returns the numeric value of an argument. The argument can be a Boolean, string, or array.<br>For example, `NUMBER("500")` returns `500`. |
| POSITION() | Returns the index position of an array that is being processed.<br>For example, `testcase[POSITION()<=2]` selects the first two test cases. |
| ROUND() | Returns a number that is the nearest integer to the specified value. If there are two such numbers, the greater one is returned.<br>For example, `ROUND(5.24)` returns `5`, and `ROUND(6.5)` returns `7`. |
| STARTSWITH() | Returns `true` if the first string starts with the second string. Otherwise, returns `false`.<br>For example, `STARTSWITH("XPath","XP")` returns `true`. |
| STRING() | Returns the string value for a Boolean or number.<br>For example, `STRING(1117)` returns `"1117"`. |
| STRINGLENGTH() | Returns the number of characters in a string.<br>For example, `STRINGLENGTH("HOLIDAY")` returns `7`. |
| SUBSTRING() | Returns a substring from the starting position to the provided length. The index of the first character of the string to be operated on is 1. The three arguments are the string to be operated on, the starting position, and the length. If the length (the third argument) is not provided, returns the substring from the starting position to the ending position.<br>For example, `SUBSTRING("Salesperson",1,3)` returns `"Sal"`, and `SUBSTRING("Salesperson",4)` returns `"esperson"`. |

| Function | Description |
|---|---|
| SUBSTRINGAFTER() | Returns the remaining characters in the first string after the second string occurs in it.<br>If the first string does not contain the second string, an empty string is returned.<br>For example, **`SUBSTRINGAFTER("555-1212","-")`** returns `"1212"`. |
| SUBSTRINGBEFORE() | Returns the characters in the first string before the second string occurs in it.<br>If the first string does not contain the second string, an empty string is returned.<br>For example, **`SUBSTRINGBEFORE("555-1212","-")`** returns `"555"`. |
| SUM() | Returns the sum of the numeric values of each element in the array.<br>For example, suppose you have a `SalesTax[]` array under a `Taxes` data element. Each array element in the array stores the sales tax amount from a transaction. **`SUM(Taxes.SalesTax)`** returns `100.00`, because there are four array elements in the array, with values of `20`, `40`, `25`, and `15` respectively. |
| TRANSLATE() | Returns the first string with occurrences of characters in the second string replaced by the character at the corresponding position in the third string.<br>For example, **`TRANSLATE("bat","abc","ABC")`** returns `"BAt"`. |

## About SOAP Messages

Orchestrations communicate with Web services by sending and receiving *SOAP messages*. SOAP messages are XML (eXtensible Markup Language) documents that are formatted according to the rules of the SOAP specification. (See the World Wide Web Consortium Web site at http://www.w3.org for more information about the SOAP specification.)

### SOAP Envelope

A SOAP message consists of a SOAP envelope. The SOAP envelope contains an optional SOAP Header and a required SOAP Body.

### SOAP Header

The optional SOAP Header includes application-specific information about how the SOAP message is to be processed. Each Header contains one or more *header blocks*, which can include message routing and delivery instructions, payment information, authentication credentials, or any other information that relates to processing the data in the SOAP Body.

The SOAP Header can also contain a *headerfault message* element that usually relates to Header-processing errors.

### SOAP Body

The required SOAP Body contains the actual message to be processed by the ultimate endpoint. The Body may contain an XML element such as `employeeNumber`, or an element that maps to the arguments or parameters in a programming method or function.

### SOAP Fault

The SOAP Body can also contain an optional *SOAP fault*, which is used to carry error and status information about a SOAP message. If an error occurs during the processing of a request, a response SOAP message is returned to the sender that contains the SOAP fault in the Body of the message.

# Chapter 2: Orchestration User Interface

This section describes the following orchestration-related dialog boxes, editors, and Property Editors.

- Orchestration Link Editor [page 39]
- New Orchestration Dialog Box [page 42]
- Event Definitions List [page 43]
- Event Definition Configuration Dialog Box [page 44]
- Event Definition Editor [page 45]
- External Event Configuration Dialog Box [page 49]
- Orchestration Workflow Editor [page 50]
- Type Library Editor [page 55]

## Orchestration Link Editor

This editor is displayed when you select an existing orchestration link (not the **Event without Reply** item) under the **Orchestration Links** heading in App Explorer. For information about orchestration links, see About Orchestration Links [page 29].

**Important:** The **Event** item under the **Orchestration Links** heading is used for asynchronous orchestration workflows, and the other items are used for synchronous orchestration workflows. For information about the **Event without Reply** item, see About Events [page 27] and Event Editor [page 40].

| Element | Description |
|---|---|
| Name | The name of the orchestration link. This name is displayed under the **Orchestration Links** heading in App Explorer. |
| Description | An optional description of the orchestration link. |
| Orchestration | The orchestration that contains the orchestration workflow that is invoked when the action corresponding to this orchestration link is executed. For convenience, you can click the name of the orchestration to view its editor. Click in the Quick Access Toolbar to return to the current view of the orchestration link editor. |

| Element | Description |
|---|---|
| Workflow | The orchestration workflow that is invoked when the action corresponding to this orchestration link is executed. For convenience, you can click the name of the orchestration workflow to view it. Click ⬅ in the Quick Access Toolbar to return to the current view of the orchestration link editor. |
| Fields used by event | The fields from the primary table that are passed as inputs to the linked orchestration workflow. If you need data that is not listed, define it as **Additional data used by event**. |
| Fields returned by event | The fields in the primary table that are set by the results of the synchronous orchestration workflow. |
| Additional data used by event | Additional data items that are passed as inputs to the linked orchestration workflow. |
| New | Adds an item to the list of additional data items. |
| Delete | Removes the selected data item from the list. |
| Move up, Move down | Moves the selected data item higher or lower in the list. |

## Event Editor

The event editor is displayed when you select **Event without Reply** under the **Orchestration Links** heading in App Explorer. This editor describes field values and additional data that is sent with events raised during application workflow transitions. These events are raised when a transition action is created that invokes an asynchronous orchestration workflow (one created in the **Action Wizard** with the **continue executing (asynchronous)** option). The event corresponds to an event definition in the called orchestration. For information about events and event definitions, see About Events [page 27] and About Application Links and Event Definitions [page 28]. For information about the Action Wizard, see the *SBM Composer Guide*.

**Note:** The event is shown as the **Event definition source** on the General Tab of the Event Definition Property Editor [page 48] for the event definition that is automatically created when you use the **Action Wizard** to create an asynchronous orchestration workflow.

**Note:** The fields and additional data that you specify in this editor appear as **Custom data** elements in the Event Definition Editor [page 45].

| Element | Description |
|---|---|
| Description | An optional description of the event. |

| Element | Description |
|---|---|
| Export to file | Lets you save the event to an `.mtd` file that can be imported by another process app that responds to this event. The file is used to create an event definition in the other process app.<br><br>You can only export an event if an asynchronous orchestration action is created for some transition.<br><br>After you export an event, a new event definition appears under the **Application Links** heading in App Explorer. |
| Lock definition | Lets an asynchronous orchestration workflow in another process app use the event definition. For details, see About Events [page 27]. |
| Fields to send with event | Lets you specify which fields from the primary table are sent with the event. If you want to use a field that is not on the list, add the field to the primary table. |
| Additional data to send with event | Lets you add data elements to send with the event. These data elements are then accessible to the receiving orchestration workflow. |
| New | Adds an entry to the list of additional data elements. |
| Delete | Removes the selected data element from the list. |
| Move up, Move down | Moves the selected data element up or down in the list. |

## Event with Reply Dialog Box

This dialog box opens when you select the following options to create an orchestration link and a new synchronous orchestration workflow from the **Action Wizard**. (See the *SBM Composer Guide* for information about the **Action Wizard**.)

- **Orchestration Workflow** as the action type
- **and wait for reply (synchronous)** in the rule description
- **(Add new workflow...)** from the **Select workflow service** list

The information you specify in this dialog box is reflected in the orchestration link editor. For more information, see About Orchestration Links [page 29] and Orchestration Link Editor [page 39].

**Note:** You can also open this dialog box by right-clicking the **Orchestration Links** heading in App Explorer and selecting **Add New Event With Reply**. In this scenario, you create the orchestration link first and then use it later in the **Action Wizard**. However, this method is not recommended.

The following table describes the configuration settings for a new orchestration link.

| Element | Description |
|---|---|
| Name | The name of the new orchestration link. This name appears under the **Orchestration Links** heading in App Explorer, where you can select it to make changes. |
| Description | An optional description for the orchestration link. |
| Orchestration | The orchestration that contains the orchestration workflow. You can select **(New orchestration...)** to associate the orchestration link with an orchestration workflow in a new orchestration that is to be added to the open process app. |
| Workflow | The name of the new orchestration workflow. |
| Fields used by event | The fields from the primary table whose values are passed as inputs to the orchestration workflow. If you need data that is not listed, define it as **Additional data used by event**, (described below). |
| Fields returned by event | The fields in the primary table whose values are returned as outputs from the orchestration workflow. **Important:** The *Item ID* field cannot be returned by the workflow. |
| Additional data used by event | Additional data items that are passed as inputs to the orchestration workflow. |
| New | Adds an item to the list of additional data items. |
| Delete | Removes the selected data item from the list. |
| Move up, Move down | Moves the selected data item higher or lower in the list. |

## New Orchestration Dialog Box

Use the **New Orchestration** dialog box to add an orchestration to a process app. This dialog box opens when you select the following options to define an orchestration action in the **Action Wizard**. (See the *SBM Composer Guide* for information about the **Action Wizard**.)

- **Orchestration Workflow** as the action type
- **and continue executing (asynchronous)** or **and wait for reply (synchronous)** in the rule description

- **(New orchestration...)** from the **Select an orchestration and an orchestration workflow** list

This dialog box also opens when you do one of the following:

- Click the down arrow under the **Component** icon on the Ribbon, and select **Orchestration**.
- Right-click the name of an existing application in App Explorer, point to **Add New**, and then select **Orchestration**.
- Right-click the process app name in App Explorer, point to **Add New**, and then select **Orchestration**.
- Right-click the name of an existing orchestration in App Explorer, point to **Add New**, and then select **Orchestration**.

| **Element** | **Description** |
|---|---|
| Name | The name of the orchestration, which is limited to 128 characters, including spaces. Orchestrations in the repository must be uniquely named, even if they are used in different process apps. |
| Category | An optional category for the orchestration. |

## Event Definitions List

When you select the **Application Links** heading under the name of an orchestration in App Explorer, the list of event defintions associated with the orchestration is displayed on the **Event Definition** tab in the editor pane.

**Tip:** Double-click an event definition in the list to view it in the Property Editor and on the event definition tab.

The following table describes the elements in the event definition editor.

| **Element** | **Description** |
|---|---|
| Name | The name of the event definition. |
| Product name | The name of the external product or application raising the event associated with the event definition. |
| Tool version | The version number of the event definition. This information is used to distinguish between event definitions used by the same product. |
| Product instance | The instance of the event definition. This information is used to distinguish between event definitions used by the same product. |
| Type | For an event defintion, the type is always Event Definition. |

| Element | Description |
|---|---|
| Updated by | The user name of the person who last reimported the event definition. |
| Updated on | The date and time that the event definition was last reimported. |

## Event Definition Configuration Dialog Box

This dialog box opens when you right-click **Application Links** under an orchestration name in App Explorer and then select **Add New Event Definition**.

**Note:** For more information about event definitions, see About Application Links and Event Definitions [page 28].

### Creating a New Custom Event Definition

The following table describes the configuration settings for a new custom event definition. These settings are present when **Create new custom event definition** is selected in the dialog box (the default selection). The Event Manager uses these settings (and the event notice, object type, and event type that you define later) to identify the orchestration workflow that will be called in response to an event raised from an external product or another process app.

| Element | Description |
|---|---|
| Event definition name | A unique name for the new event definition. This name does not affect the mapping of an event to a workflow. |
| Event definition version | The event definition version. This information is used to distinguish between event definitions used by the same product. |
| Product name | The name of the external product or application raising the event associated with the event definition. |
| Product instance | The instance of the event definition. This information is used to distinguish between event definitions used by the same product. |

### Importing an Event Definition File to Create a New Custom Event Definition

The following table describes the configuration settings for a new event definition that uses an existing event definition (`.mtd`) file or `.wsdl` file that was exported from another event definition. These configuration settings are present when you select **Create from event definition file** in the dialog box.

| Element | Description |
|---|---|
| MTD | The name of the event definition file that defines the event definition. This file was exported from another event definition, and contains the available messages, object types, and event types.<br><br>Browse for a file with an `.mtd` (or `.wsdl`) extension that defines the event you want to process. Alternatively, you could enter a URL to a Web page containing information that defines the event definition.<br><br>When you are satisfied with your selection, click **OK**.<br><br>**Note:** Be sure to specify a file that contains valid ALF event definitions. Otherwise, when you click **OK**, SBM Composer reminds you to select a valid file. |
| Documentation | Information about the event definition, as defined in the event definition file. |
| Operations | The operations defined in the process event definition file. |

## Event Definition Editor

The event definition Property Editor is displayed when you select an event definition in App Explorer. The tab displays the details of an event definition that SBM automatically creates when an application workflow is linked to an asynchronous orchestration workflow using the **Action Wizard**, or an event definition that you create or import using the **Event Definition Configuration** dialog box. (See the *SBM Composer Guide* for information about the **Action Wizard**.)

Each event is defined by an object type and an event type. When an event is fired during the execution of a transition, information is passed to the asynchronous orchestration workflow. The event definition editor shows what the information could be.

| Element | Description |
|---|---|
| Object types | The object types defined in the event definition. |
| Event types | The event types defined in the event definition. |
| Custom data | The data sent with the event, shown in the **Extension** node. You can define data elements when you create a new event definition. If you import the event definition, the **Extension** node includes the field data and additional data you specified when you created the event definition (see Event Editor [page 40]). |

### Applying Event Priorities

You can use the event types below to prioritize events. For example, to prioritize synchronous events over asynchronous events, ensure that all asynchronous events are assigned one of the INCONTROL event types. Alternatively, to lower the priority of synchronous events below Application Engine asynchronous events, you could assign them one of the INCONTROL event types.

| Priority Rank | Event Type | Comments |
|---|---|---|
| 10 | INCONTROL LOW Event | Lowest event priority. |
| 20 | INCONTROL Event | Recommended event type for external asynchronous events. |
| 30 | INCONTROL HIGH Event | |
| 40 | LOW Priority Event | Assigned to all asynchronous events by default, including those generated by Application Engine. |
| 50 | MEDIUM Priority Event | |
| 60 | HIGH Priority Event | Assigned to all synchronous events by default. It is not recommended to assign this event type to asynchronous events to prioritize them over synchronous events. |
| 70 | CRITICAL Priority Event | Highest event priority. Use only if you are certain that an asynchronous event should be executed prior to all other asynchronous events and synchronous events. |

By default, synchronous events from Application Engine are assigned **HIGH Priority Event** status and asynchronous events from Application Engine are assigned **LOW Priority Event** status. This gives priority to orchestrations that are executed when a user performs a transition in the browser interface so that users are not waiting unnecessarily.

For external events, this type can be set directly in the "**EventType**" parameter of the ALF Event SOAP message.

## Map Event Definition to Workflow Dialog Box

This dialog box opens when you add a new event map from the **Event Map** tab of the event definition Property Editor. It enables you to associate an event definition event with an asynchronous orchestration workflow. In this dialog box, you select from the events that the event definition can raise, and select an orchestration workflow that is compatible with the selected event. You can also use this dialog box to create a new orchestration workflow.

When the event definition defines specific events, the dialog box contains only compatible events. Event compatibility is determined by the content of the **Extension** element in the **Custom data** section of the event definition editor. If two different elements have the same information in the **Extensions** element, then the same orchestration workflow can handle both events.

**Important:** An event definition must contain at least one object type and one event type.

The following table describes the configuration settings for a new event map.

| Element | Description |
|---|---|
| Event definition | The name of the event definition. This field is read-only. |
| Tool version | The version number of the event definition. This information is used to distinguish between event definitions used by the same product. |
| Product name | The name of the external product or application raising the event associated with the event definition. |
| Product instance | The instance of the event definition. This information is used to distinguish between event definitions used by the same product. |
| Object type | The object type that together with the event type, defines this mapping to the Event Manager. The object type identifies the type of object or record that triggers the event type.<br><br>Select the object type to associate with the orchestration workflow. |
| Event type | The event type that together with the object type, defines this mapping to the Event Manager. The event type identifies what occurs to trigger the event.<br><br>Select the event type to associate with the orchestration workflow. |
| Workflow | Select an existing orchestration workflow or select **[New Workflow]** to create a new one. This should be the orchestration workflow that you want to run when an event with the selected values is received by the Event Manager. |

## Event Definition Property Editor

The event definition Property Editor is displayed when you select an event definition in App Explorer. Use it to control how event definitions are mapped to asynchronous orchestration workflows in your process app. You can also use it to perform operations such as exporting the `.mtd` or `.wsdl` file for the event definition so it can be imported by another process app or an external product, making an imported event definition editable, and refreshing the definition of the event definition.

## General Tab of the Event Definition Property Editor

This tab is displayed when you select an event definition in App Explorer. It is also displayed when you double-click an event definition in the event definition editor. In App Explorer, event definitions are displayed under the **Application Links** heading in an orchestration, and can be displayed under the **External Events** subheading under **Orchestration Links** in an application.

| Element | Description |
|---|---|
| Name | The name of the event definition. |
| Product name | The name of the external product or application raising the event associated with the event definition. |
| Event definition source | The origin of the event definition:<br>• Imported from the event for the application. (The event is automatically created when you create an asynchronous orchestration workflow using the **Action Wizard**.)<br>• An `.mtd` file or a `.wsdl` file that was exported from another event definition, or a `.wsdl` file that was used to manually create an event definition. (Select **Create from event definition file** on the **Event Definition Configuration** dialog box to create this event definition).<br>• Defined manually by the user. (Select **Create new custom event definition** on the **Event Definition Configuration** dialog box to create this event definition.) |
| Description | An optional description. |
| Event definition version | The version number from `.mtd` or `.wsdl` file. |
| Product instance | The instance of the event definition. This information is used to distinguish between event definitions used by the same product. |
| Export event definition | Saves the `.mtd` or `.wsdl` file for the event definition to the file system of your computer, so it can be imported into any application or orchestration. For more information, see About Application Links and Event Definitions [page 28]. |
| Reimport | Reimports the `.mtd` or `.wsdl` file. This button is only available for existing event definitions. |

| Element | Description |
|---|---|
| Regenerate | Regenerates the event definition. This button is only available for event definitions that were automatically generated from an event in an application. Use this button to regenerate the event definition if the changes you made to the event do not appear in the event definition editor. |
| Export external event WSDL | Saves the `.wsdl` file for the event definition to the file system of your computer, so it can be used to raise an event from any external product that can call Web services. **Note:** This button is only available for new event definitions. For more information, see About Application Links and Event Definitions [page 28]. |
| Convert to custom event definition | Converts an event definition that was created in another process app or created manually into a custom event definition that you can edit. **Note:** This button is available for any event definition that was created in and exported from SBM Composer. It is also available for a manually-created event definition, if its structure and naming conventions are consistent with those of custom event definitions created by SBM Composer. |

## External Event Configuration Dialog Box

This dialog box opens when you create a new external event from App Explorer or when you use the **Action Wizard** to import an event definition that was exported from an orchestration or application. Use it to provide the required configuration information.

The following table describes the configuration settings for a new event definition.

| Element | Description |
|---|---|
| File | The name of the event definition that defines the external event. Browse for an `.mtd` or `.wsdl` file, or enter a URL, and then press the Tab key or click in another field to read the file. |
| Service | Lists the services defined in the file. If only one service is defined, this field is read-only. |
| Port | Lists the unique names of ports for the selected service, as defined in the file. If only one port is defined, this field is read-only. |
| Documentation | Optional information about the services, as provided by the creator of the file. This field is read-only. |

| Element | Description |
|---|---|
| Operations | Lists the individual operations available for the selected service, as defined in the file. |

**Note:** For information about event definitions, see About Application Links and Event Definitions [page 28]. For information about the **Action Wizard**, see the *SBM Composer Guide*.

## Orchestration Workflow Editor

The orchestration workflow editor is displayed when you select an orchestration workflow or subroutine in App Explorer. This is where you create and edit an orchestration workflow or subroutine. You use the orchestration workflow editor (with the **Step Palette** and the orchestration workflow Property Editor) to visually lay out and sequence the components of your orchestration workflow or subroutine.

You create an orchestration workflow or subroutine by dragging and dropping steps from the **Step Palette** and arranging them in sequence. You also configure details such as which Web service is to be called or what calculations are to be performed; and map data among inputs, outputs, and working data.

### Step Palette

The **Step Palette** is located to the right of the orchestration workflow editor. It contains the icons that represent the various functions that an orchestration workflow or subroutine can perform.

The **Step Palette** has three main sections:

- **New Items**, from which you can drag-and-drop orchestration workflow steps.
- **Configured Items**, where items are shown if they have been configured. For example, after you specify the WSDL file for a Web service, it appears in this section.
- Zoom preview section, where you can manipulate the portion of a large orchestration workflow that is visible in the orchestration workflow editor.

### Orchestration Workflow Property Editor

This section describes the tabs of the orchestration workflow Property Editor. This Property Editor is displayed when you select an orchestration workflow or subroutine in App Explorer.

- General Tab of the Orchestration Workflow Property Editor [page 50]
- Event Map Tab of the Orchestration Workflow Property Editor [page 51]
- Data Mapping Tab of the Orchestration Workflow Property Editor [page 52]

#### General Tab of the Orchestration Workflow Property Editor

The **General** tab of the orchestration workflow Property Editor is displayed when you select an orchestration workflow or subroutine in App Explorer.

| Element | Description |
|---|---|
| Name | The name of an orchestration workflow or subroutine.<br>The name must start with a letter (A-Z, a-z), and can contain any combination of letters, digits (0-9), and underscores. |
| Type | The orchestration workflow type, either **Synchronous** or **Asynchronous**. See Comparing Synchronous With Asynchronous Orchestration Workflows [page 14] for details.<br>For subroutines, the type is **Synchronous**. |
| Description | An optional description of the orchestration workflow or subroutine. |
| Event definition *(orchestration workflow only)* | The name of the event definition that is linked to the orchestration workflow. |
| WSDL message *(orchestration workflow only)* | The message from the WSDL file for the event definition. Often this is the EventNotice containing the data that accompanies the event. |

### Event Map Tab of the Orchestration Workflow Property Editor

The **Event Map** tab of the orchestration workflow Property Editor is displayed when you select an aynchronous orchestration workflow in App Explorer. The linking between an event and the orchestration workflow is called an *event map*, and is displayed on this tab.

| Element | Description |
|---|---|
| Event definition | The name of the event definition that is mapped to the orchestration workflow. |
| Object type | The object type that together with the event type, defines this mapping to the Event Manager. The object type identifies the type of object or record that triggers the event type. |
| Event type | The event type that together with the object type, defines this mapping to the Event Manager. The event type identifies what occurs to trigger the event. |
| Add | Opens the **Map Workflow to Event Definition** dialog box, in which you map the orchestration workflow to an event definition. See Map Workflow to Event Definition Dialog Box [page 54] for details. |
| Remove | Removes the selected event map entry. |

| Element | Description |
|---|---|
| View/ Edit | Opens the **Event Definition Event Mapping** dialog box, which lets you view the mapping of event information to an orchestration workflow. See Event Definition Event Mapping Dialog Box [page 53] for details.<br>**Note:** If the orchestration workflow is checked in, the button is **View**, and the dialog box is read-only. If it is checked out, the button is **Edit**, and you can modify some fields in the dialog box. |

## Data Mapping Tab of the Orchestration Workflow Property Editor

The **Data Mapping** tab of the orchestration workflow Property Editor is displayed when you select an orchestration workflow or subroutine in App Explorer. See About Data Mapping [page 19] for related information.

| Element | Description |
|---|---|
| Working data | Displays (in a hierarchical manner) the working data for the orchestration workflow or subroutine.<br>Right-click a working data element for a menu of applicable commands. For example, the **Properties Mode** and **Mapping Mode** commands toggle between the two modes, and the **Type [*type*]** command gives you the option of changing the type of the selected data element.<br>**Note:** Properties mode shows additional information for the selected element (such as its type and namespace). |
| Inputs | Displays the inputs to the orchestration workflow or subroutine. For subroutines, you can add, edit, and delete inputs, while for orchestration workflows, you can only view inputs.<br>Right-click an input to display a menu of applicable commands. |
| Source elements | Shows the source of each working data element, if any.<br>Right-click a data element and select **Suggested Mappings** to see the mappings that SBM Composer suggests.<br>Select a cell in this column and click the down arrow to open the **Select a Source** popup, which offers options beyond the suggested mappings.<br>Right-click a source element to open a menu of applicable commands. |
| Default value | Lets you view or edit the default values, if any, for data elements.<br>**Note:** Any value specified in the **Source elements** column will override the value in the corresponding **Default value** column. |

| Element | Description |
|---|---|
| *Vertical divider* | Clicking the vertical divider to the right of the Property Editor switches the Property Editor between mapping mode and properties mode. You can drag the divider to the left to expand the properties mode panel. |

## Event Definition Event Mapping Dialog Box

The **Event Definition Event Mapping** dialog box lets you view or edit the mapping between an event definition and an asynchronous orchestration workflow. The map defines the specific values for the event that causes the workflow to be invoked.

| Element | Description |
|---|---|
| Event Definition | Identifies the event definition. |
| Version | The version number of the event definition. This information is used to distinguish between event definitions used by the same product. |
| Product name | The name of the external product or application raising the event associated with the event definition. |
| Product instance | The instance of the event definition. This information is used to distinguish between event definitions used by the same product.<br>If the mapped orchestration workflow is checked out, you can modify the value of this element. |
| Object type | The object type that together with the event type, defines this mapping to the Event Manager. The object type identifies the type of object or record that triggers the event type.<br>If the mapped orchestration workflow is checked out, you can modify the value of this element. |
| Event type | The event type that together with the object type, defines this mapping to the Event Manager. The event type identifies what occurs to trigger the event.<br>If the mapped orchestration workflow is checked out, you can modify the value of this element. |
| Workflow | The orchestration workflow that you want to run when an event with the selected values is received by the Event Manager. |

## Map Workflow to Event Definition Dialog Box

This dialog box opens when you add a new event map from the **Event Map** tab on an asynchronous orchestration workflow Property Editor. It lets you associate an orchestration workflow with an event definition. In this dialog box, you select from the events that the event definition can raise, and select an orchestration workflow that is compatible with the selected event. You can also use this dialog box to create a new orchestration workflow.

You can associate an orchestration workflow with more than one event but the events must be compatible. Event compatibility is determined by the content of the **Extension** element in the **Custom data** section of the event definition editor. If two different elements have the same information in the **Extensions** element, then the same orchestration workflow can handle both events.

After an orchestration workflow is associated with an event definition event, the dialog box shows only compatible event definition events.

**Note:** Event definition events can come from SBM applications or from external products.

The following table describes the configuration settings for a new event map.

| Element | Description |
|---|---|
| Event definition | The name of the event definition. This field is read-only. |
| Tool version | The version number of the event definition. This information is used to distinguish between event definitions used by the same product. |
| Product name | The name of the external product or application raising the event associated with the event definition. |
| Product instance | The instance of the event definition. This information is used to distinguish between event definitions used by the same product. |
| Object type | The object type that together with the event type, defines this mapping to the Event Manager. The object type identifies the type of object or record that triggers the event type.<br>Select the object type to associate with the orchestration workflow. |
| Event type | The event type that together with the object type, defines this mapping to the Event Manager. The event type identifies what occurs to trigger the event.<br>Select the event type to associate with the orchestration workflow. |

| Element | Description |
|---|---|
| Workflow | Select an existing orchestration workflow or select **[New Workflow]** to create a new one. This should be the orchestration workflow that you want to run when an event with the selected values is received by the Event Manager. |

## Select Library Type Dialog Box

You use the **Select Library Type** dialog box when you create a new data element and need to assign a type to it, or when you need to change the type of an existing data element.

This dialog box makes it easy to select a type when there are many types in the Type Library. The types that are listed in the dialog box are all named types defined by the Web services that were imported into the orchestration, and that are listed in the Type Library Editor [page 55].

This dialog box opens when you perform the following steps:

1. Click the **Data Mapping** tab on the orchestration workflow Property Editor.
2. Right-click in the **Working Data** area, and select **Properties Mode**, if not already selected.
3. To add a new data element and select the library type, right-click the **Working Data** node, select **Add New**, and then select **Select from Type Library**.
4. To change the library type of an existing data element, right-click the data element, select **Type [*type*]**, and then select **Select from Type Library**.

| Element | Description |
|---|---|
| Look for | Type a partial or complete data element name you want to find. |
| Find | Starts the search. |
| Clear | Clears the information in the **Look for** box. |
| Options | Provides other search options. |
| Show type details | Opens the **Type details** section, which shows detailed information about the selected element. |

## Type Library Editor

When you select the **Type Library** heading under the name of an orchestration in App Explorer, a list of named types associated with the process app is displayed in the editor pane. This read-only list contains all named types defined by the Web services that were imported into the orchestration.

The named types are grouped by namespace at the top part of the editor. When you select a named type, its data elements are displayed at the bottom part of the editor.

You can search for a named type by typing the full name or a few letters of the name in the **Look for** box, and then clicking **Find**. Search options are available in the **Options** menu.

# Chapter 3: Orchestration Procedures

This section includes the following orchestration procedures:

- Using Data Mapping [page 57]
- Creating a New Custom Event Definition [page 69]
- Importing an Event Definition File for a New Custom Event Definition [page 69]
- Mapping an Orchestration Workflow to an Event Definition [page 70]
- Using the Step Palette [page 71]
- Using the Scope, Throw, and Compensate Steps to Handle Faults From Web Services [page 96]
- Raising External Events [page 137]

## Using Data Mapping

You use the **Data Mapping** tab of the orchestration workflow Property Editor to create working data and define its default values, and to specify the source of data needed by a Web service. This section includes data mapping procedures.

- Creating a Practice Process App for Data Mapping [page 58]
- Creating Private Simple or Library Type Working Data [page 59]
- Creating Private Complex Working Data [page 60]
- Creating Arrays of Working Data [page 61]
- Setting Default Values [page 63]
- Setting Source Values Using Suggested Mappings [page 64]
- Setting Source Element Mappings Manually [page 64]
- Mapping Identical Structures [page 65]
- Viewing and Editing Data Element Properties [page 66]
- Showing the Required Flag [page 67]
- Clearing Data Mapping [page 68]

## Creating a Practice Process App for Data Mapping

In this section, you will create a new process app (DataMappingProcessApp) that contains an application workflow. Then you will create an orchestration workflow (DataMappingOrchWF), which you will use to practice mapping data.

**Important:** This process app is for demonstration purposes only and is not valid. Do not try to publish or deploy it.

**To create the practice process app:**

1. Start SBM Composer.
2. Click the Composer button, and then select **New**.
3. In the **Create New Process App** dialog box that opens, in the **Available Templates** pane, click **Application Process App**, and then click **Create**.
4. In the **Configure Process App** dialog box that opens, in the **Process app name** box, type `DataMappingProcessApp`.
5. In the **Category** box, type `Examples`.
6. In the **Application name** box, type `DataMappingApp`, and then click **OK**.
7. In App Explorer, under the **Application Workflows** heading, right-click **DataMappingApp** and then select **Rename**.
8. Change the name to `DataMappingAppWF` and then press the Tab key.
9. In App Explorer, right-click **DataMappingProcessApp**, point to **Add New**, and then select **Orchestration**.
10. In the **New Orchestration** dialog box, type `DataMappingOrch` in the **Name** box and then click **OK**.
11. In App Explorer, under **Application Workflows**, select **DataMappingAppWF**.
12. In the application workflow editor, select the **New** state. On the **General** tab of the application workflow Property Editor, change the value of the **Name** field to `State`, and then press the Tab key.
13. In the application workflow editor, select the **Submit** transition.
14. On the **General** tab of the transition Property Editor, change the value of the **Name** field to `Transition`, and then press the Tab key.
15. Right-click the **Transition** transition, and select **Show Actions**.
16. On the **Actions** tab of the transition Property Editor, click **New.**

    The **Action Wizard** asks, "Which type of action do you want to execute?"
17. Without changing anything, click **Next.**

    The **Action Wizard** asks, "What do you want to affect?"
18. Without changing anything, click **Next**.

The **Action Wizard** asks, "Which condition do you want to check?"

19. Without changing anything, click **Next**.

    The **Action Wizard** asks, "Which orchestration workflow do you want to invoke?"

20. In the list under **Step 1**, select **(Add new workflow...)**.

    **SubmitTransitionWorkflow** is added to list and selected.

21. Without changing anything in **Step 2**, click **Finish**.

22. In App Explorer, under **DataMappingOrch**, under **Orchestration Workflows**, select **SubmitTransitionWorkflow**.

23. On the **General** tab of the orchestration workflow Property Editor, change the value of the **Name** field to `DataMappingOrchWF`, and then press the Tab key.

24. Save the process app by performing the following steps:

    a. On the Quick Access Toolbar, click the **Save locally** icon. A message reminds you that the design elements were saved to the Local Cache only.

    b. Click **OK**.

25. If you closed SBM Composer after saving the process app, perform the following steps:

    a. Start SBM Composer.

    b. Click the Composer button, and then select **Open**.

    c. In the **Open Process App** dialog box, select **DataMappingProcessApp**, and then click **Open.**

    d. In App Explorer, click the **All Items** filter, if it is not already selected.

## Creating Private Simple or Library Type Working Data

On the **Data Mapping** tab of the orchestration workflow Property Editor, you can create *private simple* or *library type* working data elements to use in an orchestration. The steps for creating them are the same.

Private simple working data is classified by data types. In computer programming, a data type restricts a data element to a particular type of information. For example, a *string* can contain only characters and spaces. "ProcessApp101" and "ab cde!f" are both strings. An *integer* can contain only whole numbers, that is, numbers that do not contain fractional parts. The numbers 1, 88, and 1099 are integers; however, 1.5, 88.72, and 1099.579 are not (they are called *floating-point numbers*). Private simple working data is shared by all of the steps in the orchestration workflow.

Library type working data is provided by the WSDL files that are imported into an orchestration workflow. For example, the "auth" library type is a private complex type that contains the userId, password, hostname, and loginAsUserId parameters supplied by the SBM Web service. For more information about the "auth" data element (argument), refer to the *SBM Web Services Developer's Guide*.

**Procedure Summary**

1. Select an orchestration workflow to display it in the orchestration workflow editor.
2. Click the **Data Mapping** tab of the orchestration workflow Property Editor.
3. To add a new data element, right-click the **WorkingData** step input, select **Add New**, and then select **Select from Type Library**.
4. Select a data type. (The icon to the left of the name changes accordingly.)
5. Change the name of the new data element, if you want.

In the following exercise, you will create a private simple working data element (PrivateSimple) and a library type data element (LibraryType).

**To create a private simple working data element and a library type working data element:**

1. In App Explorer, select **DataMappingOrchWF** under the **Orchestration Workflows** heading.
2. In the orchestration workflow Property Editor, select the **Data Mapping** tab.

   The **WorkingData** step input should be visible under the name of the orchestration workflow (**DataMappingOrchWF**) in the **Working data** column. (If it is not, click a blank area in the orchestration workflow editor.)
3. To create a private simple working data element:
   a. Right-click the **WorkingData** step input, select **Add New**, and then select a data type such as **Boolean** or **Integer**.

      A new data element (**Boolean** or **Integer**) is added under the **WorkingData** step input.

   b. Change the name of the new data element to `PrivateSimple`.
4. To create a library type working data element:
   a. Right-click the **WorkingData** step input, select **Add New**, and then select **Select from Libray Type**.

   b. Select a data type such as **Auth** or **CredentialsType**. If you selected **Auth**, four child elements are added under the **Auth** data element.

      A new data element (**Auth** or **Credentials**) is added under **PrivateSimple**.

   c. Change the name of the new data element to `LibraryType`.

## Creating Private Complex Working Data

On the **Data Mapping** tab of the orchestration workflow Property Editor, you can create private complex data to use in orchestrations.

**Procedure Summary**

1. Select an orchestration workflow to display it in the orchestration workflow editor.
2. Click the **Data Mapping** tab of the orchestration workflow Property Editor.

3. To add a new data element, right-click the **WorkingData** step input, select **Add New**, and then select **Complex Type**.

4. Rename the new data element, if you want.

5. To add a child data element, right-click the new data element again, select **Add Child**, and then select a type.

6. Rename the new child data element, if you want.

7. Repeat the previous two steps to add more child data elements.

In the following exercise, you will create a private complex working data element named "PrivateComplex" that contains three child data elements (Child1, Child2, and Child3).

**To create a private complex working data element:**

1. Under **Orchestration Workflows**, select **DataMappingOrchWF.**

2. In the orchestration workflow Property Editor, click the **Data Mapping** tab.

   The **WorkingData** step input should be visible under the name of the orchestration workflow (**DataMapping**) in the **Working data** column. (If it is not, click a blank area in the orchestration workflow editor.)

3. Right-click the **WorkingData** step input, select **Add New**, and then select **Complex Type**.

   **ComplexType** appears under the **WorkingData** step input.

4. Change the name of the new data element (**ComplexType**) to `PrivateComplex`.

5. Right-click **PrivateComplex**, select **Add Child**, and then select a type. A new child data element is added under **PrivateComplex.**

6. Change the name of the new child data element to `Child1`.

7. Repeat steps 5 [page 61] and 6 [page 61] two more times, naming the new child data elements `Child2` and `Child3`.

## Creating Arrays of Working Data

On the **Data Mapping** tab of the orchestration workflow Property Editor, you can create an array of working data to use in orchestrations. To do this, you first create an array container data element, then the array, and then the array elements.

**Procedure Summary**

1. Select an orchestration workflow to display it in the orchestration workflow editor.

2. Click the **Data Mapping** tab of the orchestration workflow Property Editor.

3. Create the array container data element by right-clicking the **Working Data** step input, selecting **Add New**, and then selecting **Complex Type**.

4. Rename the new array container data element, if you want.

5. Create the array by right-clicking the array container data element again, selecting **Add Child**, and then selecting a type.
6. Right-click the array, and then select **Properties Mode**, if it is not already selected.
7. In the properties area under **Misc**, select the **IsUnbounded** row.
8. Click the down arrow next to **False**, and then select **True**.
9. Close the properties area by right-clicking the array and then selecting **Mapping Mode**.
10. Rename the array, if you want.
11. Create an array data element by right-clicking the array and then selecting **Add Array Element**. (The new array data element inherits the name and data type of the array. You cannot change this.)
12. Repeat the previous step to add more array data elements.

In the following exercise, you will create an array container data element (ArrayContainer) that contains one array (Array) with three array elements (Array[1], Array[2], and Array[3]).

**To create an array of working data elements:**

1. In App Explorer, select **DataMappingOrchWF** under the **Orchestration Workflows** heading.
2. In the orchestration workflow Property Editor, click the **Data Mapping** tab.

   The **WorkingData** step input should be visible under the name of the orchestration workflow (**DataMappingOrchWF**) in the **Working data** column. (If it is not, click a blank area in the orchestration workflow editor.)
3. Right-click the **WorkingData** step input, select **Add New**, and then select **Complex Type**.

   A new data element (**ComplexType**), which is the array container, is added under the **WorkingData** step input.
4. Change the name of the new data element (**ComplexType**) to `ArrayContainer`.
5. Right-click **ArrayContainer**, select **Add Child**, and then select a type.

   A new child element is added under **ArrayContainer**.
6. Change the name of the new child element to `Array`.
7. Right-click **Array**, and then select **Properties Mode**.

   The properties area opens on the right side of the orchestration workflow Property Editor.
8. In the properties area under **Misc**, select the **IsUnbounded** row.
9. Click the down arrow next to **False**, and then select **True**.

A left and right square bracket are added to the array name (**Array[]**) to indicate that the data element is an array.

10. To close the properties area, right-click **Array[]**, and then select **Mapping Mode**.

    The number of array elements is displayed in parentheses to the right of **Array[]**. The number begins at 0 (zero) and increases by one each time you add an array element.

11. Right-click **Array[]**, and then select **Add Array Element**.

    A new array element (**Array[1]**) is added under **Array[]**.

12. Repeat step 11 [page 63] two more times.

    Array elements **Array[2]** and **Array[3]** are added to **Array[]** under **Array[1]**, and **(3 elements)** is added to the right of **Array[]**.

## Setting Default Values

On the **Data Mapping** tab of the orchestration workflow Property Editor, you can set the default value for a data element.

**Note:** For string type data elements, you must use escape sequences to add a new line, insert a tab, enter a carriage return, or type a literal backslash. See Using Escape Sequences [page 21] for details.

**Procedure Summary**

1. Select an orchestration workflow to display it in the orchestration workflow editor.

2. Click the **Data Mapping** tab of the orchestration workflow Property Editor.

3. Click in the **Default value** column of a data element. Depending on the data type, enter a value, click the down arrow and select an option on a menu, click the down arrow and select a date from a calendar, and so on.

In this procedure, you will set the default value for PrivateSimple, which is a String data type, to `Text`.

**To set the default value for a data element:**

1. In App Explorer, select **DataMappingOrchWF** under the **Orchestration Workflows** heading.

2. In the orchestration workflow Property Editor, click the **Data Mapping** tab.

   The **WorkingData** step input should be visible under the name of the orchestration workflow (**DataMappingOrchWF**) in the **Working data** column. (If it is not, click a blank area in the orchestration workflow editor.)

3. If the data type for **PrivateSimple** is something other than **String**, right-click the **PrivateSimple** data element, point to **Type [*type*]**, point to **Private Simple**, and then select the **String** data type.

4. Locate the **PrivateSimple** data element, click in the cell corresponding to the **Default value** column, enter `Text`, and then press the Tab key.

## Setting Source Values Using Suggested Mappings

On the **Data Mapping** tab of the orchestration workflow Property Editor, you can set the source value for a data element in its **Source elements** column to use in an orchestration workflow. You can perform this procedure by selecting suggested mappings or by selecting the values manually. Suggested mappings are for compatible data types only. You can also map compatible and incompatible data types using the **Select a source** popup, as explained in Setting Source Element Mappings Manually [page 64].

**Procedure Summary**

1. Select an orchestration workflow to display it in the orchestration workflow editor.

2. Click the **Data Mapping** tab.

3. Right-click the **Source elements** column of the data element for which you want to set a source value, point to **Suggested Mappings**, and then select a source value. (You might have to expand one or more data elements to locate the data element you want to map.)

In the following exercise, you will map the source data for the **LibraryType** data element to a suggested data source.

**To use suggested mappings for setting the source value for a data element:**

1. In App Explorer, select **DataMappingOrchWF** under the **Orchestration Workflows** heading.

2. In the orchestration workflow Property Editor, click the **Data Mapping** tab.

   The **Working Data** step input should be visible under the name of the orchestration workflow (**DataMappingOrchWF**) in the **Working data** column. (If it is not, click a blank area in the orchestration workflow editor.)

3. Right-click the **Working Data** step input, select **Add New**, and then select **String**.

4. Change the name of the new data element (**String**) to `SuggestedMapping`.

5. Locate the **SuggestedMapping** data element, right-click the corresponding cell in the **Source elements** column, point to **Suggested Mappings**, point to **Compatible Items**, and then select **Array[1](Working Data - \ArrayContainer)**.

   **DataMappingOrchWorkflow\ArrayContainer\Array[1]** is added to the **Source elements** column.

   **Note:** The **Suggested Mappings** menu is limited to 40 items: a maximum of 20 exact matches, 10 similar matches, and 10 other matches.

## Setting Source Element Mappings Manually

On the **Data Mapping** tab of the orchestration workflow Property Editor, you can set the source value for a data element in its **Source elements** column to use in an orchestration workflow. You can perform this procedure by selecting suggested mappings or by selecting the values manually. You select compatible and incompatible data types manually using the **Select a Source** popup. Selecting suggested mappings is explained in Setting Source Values Using Suggested Mappings [page 64].

**Procedure Summary**

1. Select an orchestration workflow to display it in the orchestration workflow editor.
2. Click the **Data Mapping** tab of the orchestration workflow Property Editor.
3. Find a data element for which you want to set a source value, select its **Source Elements** column, and then click the down arrow.
4. In the **Select a source** popup, select the source value or values that you want to map, and then click **OK**.

In the following exercise, you will manually map the data for **Child1** under the **PrivateComplex** data element to a source value of a compatible data type.

**To manually set the source value for a data element:**

1. In App Explorer, select **DataMappingOrchWF** under the **Orchestration Workflows** heading.
2. In the orchestration workflow Property Editor, click the **Data Mapping** tab.

   The **Working Data** step input should be visible under the name of the orchestration workflow (**DataMappingOrchWF**) in the **Working data** column. (If it is not, click a blank area in the orchestration workflow editor.)
3. Right-click the **Working Data** step input, select **Add New**, and then select **String**.
4. Change the name of the new data element (**String**) to `ManualMapping`.
5. Locate the **ManualMapping** data element, select the corresponding cell in the **Source elements** column, and then click the down arrow.
6. In the **Select a source** popup, under **Working Data**, expand **ArrayContainer** and **Array[] (3 records)**, select **Array[2]**, and then click **OK.**

   **DataMappingOrchWorkflow\ArrayContainer\Array[2]** is added to the **Source elements** column.

   This data source is of a compatible data type, as indicated by the message "Compatible type selected" that briefly appears in the box to the left of the **Clear** button in the **Select a Source** popup. The **Select a Source** popup also lets you map incompatible data types or to select multiple data types to map to a single data element, or both. To do this, click the **Show advanced options** link, and then select the appropriate check box or check boxes.

## Mapping Identical Structures

To reuse complex data structures or arrays within an orchestration workflow, copy them to the desired locations in working data, inputs, or outputs. To pass on the mappings and default values of a structure or array, use the **Select a Source** popup. If the structures are identical, the compatible mapping appears in bold in the tool.

**Procedure Summary**

1. Select an orchestration workflow to display it in the orchestration workflow editor.
2. Click the **Data Mapping** tab of the orchestration workflow Property Editor.

3. Copy a private complex working data structure to another working data location.
4. Use the **Select a Source** popup to map the copy to the original structure.

In the following exercise, you will make a copy of a complex working data structure and map the copy to the original structure.

1. In App Explorer, select **DataMappingOrchWF** under the **Orchestration Workflows** heading.
2. In the orchestration workflow Property Editor, click the **Data Mapping** tab.

   The **Working Data** step input should be visible under the name of the orchestration workflow (**DataMappingOrchWF**) in the **Working data** column. (If it is not, click a blank area in the orchestration workflow editor.)
3. Right-click the **PrivateComplex** data structure, and select **Copy**.
4. Paste the **PrivateComplex** data structure in another working data location.
5. Select the copy's **Source Elements** column, and then click the down arrow.
6. In the **Select a source** popup, select the original **PrivateComplex** data structure, and then click **OK**.

**Note:**

- If incompatible structures are mapped, the mapping appears in a warning color (dark yellow) preceded by a # character, and a validation warning will occur.
- If a mapping that was originally compatible has become incompatible, it also appears in dark yellow preceded by a # with a validation warning. For example, this can occur if a previously allowed mapping in earlier SBM versions has become invalid, or if something has changed in the source or target structure that has caused the mapping to be invalid.
- Additionally, if incompatible structures are mapped with the **Permit incompatible types to be mapped** option enabled, the mapping appears in a warning color (dark blue) preceded by a ! character. No validation warning occurs in this case.

## Viewing and Editing Data Element Properties

On the **Data Mapping** tab of the orchestration workflow Property Editor, you can view and edit property information for a data element. This is useful if you need to view the data types of data elements, or if you need to view or edit other properties, such as the namespace, for a particular data element.

### Procedure Summary

1. Select an orchestration workflow to display it in the orchestration workflow editor.
2. Click the **Data Mapping** tab of the orchestration workflow Property Editor.
3. Right-click the **Working data** or **Source elements** column for working data, or the **Step inputs** or **Source elements** column for source data, and then select **Properties Mode**.

4. View or edit the properties for the selected working data, or view the data elements for a **Service** step in the properties area that opens on the right side of the orchestration workflow Property Editor.

5. To exit properties mode, right-click the **Working data**, **Step inputs**, or **Type** column, and then select **Mapping Mode**.

   **Tip:** You can also switch between mapping mode and properties mode by clicking the vertical divider on the right side of the orchestration workflow Property Editor.

In the following exercise, you will view the properties of the **PrivateSimple** working data element.

**To view property information for a data element:**

1. In App Explorer, select **DataMappingOrchWF** under the **Orchestration Workflows** heading.

2. In the orchestration workflow Property Editor, click the **Data Mapping** tab.

   The **Working Data** step input should be visible under the name of the orchestration workflow (**DataMappingOrchWF**) in the **Working data** column. (If it is not, click a blank area in the orchestration workflow editor.)

3. Locate the **SuggestedMapping** data element, right-click the corresponding cell in the **Source elements** column, and then select **Properties Mode**.
   You can view the properties of working data elements and change any editable values in the property area that opens on the right side of the orchestration workflow Property Editor.

   **Note:** You can only view the properties of a **Service** step.

4. To exit properties mode, right-click the **SuggestedMapping** data element, and then select **Mapping Mode**.

   **Tip:** You can also switch between mapping mode and properties mode by clicking the vertical divider on the right side of the orchestration workflow Property Editor.

## Showing the Required Flag

In the hierarchy of step inputs and data elements on the **Data Mapping** tab of the orchestration workflow Property Editor, each element has an associated icon. You can identify required step inputs and data elements by a red symbol that appears in the top left corner of the icon. This symbol is called the required flag.

Flagged input and output data elements for a Web service require specific data. Sometimes a step input or a data element of the complex type displays the required flag, but its child data elements are not flagged. To determine which data elements and the type of data that are required for a Web service, refer to its WSDL file or to the documentation for the Web service.

Flagged working data must be assigned a value before it can be used in an orchestration workflow.

The following table gives examples of icons that identify required data elements.

| **Required Data Type** | **Icon** |
|---|---|
| String | T |
| Integer | 7 |
| Date | 📅 |

**Procedure Summary**

1. Select an orchestration workflow to display it in the orchestration workflow editor.
2. Click the **Data Mapping** tab in the orchestration workflow Property Editor.
3. To show the required flag for all required step inputs and data elements, right-click anywhere in the data mapping area, and then select **Show Required Flag**.
4. To hide all required flags, right-click anywhere in the data mapping area, and then clear **Show Required Flag**.

**To show the required flag for all required step inputs and data elements:**

1. In App Explorer, select **DataMappingOrchWF** under the **Orchestration Workflows** heading.
2. In the orchestration workflow Property Editor, click the **Data Mapping** tab.

   The **Working Data** step input should be visible under the name of the orchestration workflow (**DataMappingOrchWF**) in the **Working data** column. (If it is not, click a blank area in the orchestration workflow editor.)
3. Right-click anywhere in the data mapping area, and then select **Show Required Flag**.

   The required flag appears on all required step inputs and data elements.
4. To hide all required flags, right-click anywhere in the data mapping area, and then clear **Show Required Flag**.

## Clearing Data Mapping

On the **Data Mapping** tab of the orchestration workflow Property Editor, you can clear the mapping for a data element. (You cannot use this procedure to clear the default value.)

**Procedure Summary**

1. Select an orchestration workflow to display it in the orchestration workflow editor.
2. Click the **Data Mapping** tab of the orchestration workflow Property Editor.
3. Find a data element for which you want to clear the data mapping, right-click the corresponding cell in the **Source elements** column, and then select **Clear Mapping**.

4. You can also clear the mapping by selecting the appropriate cell in the **Source elements** column, clicking the down arrow, selecting **[no mapping]** in the **Select a source** popup, and then clicking **OK**.

In this exercise, you will clear the mapping for the **SuggestedMapping** data element.

**To clear the data mapping for a data element:**

1. In App Explorer, select **DataMappingOrchWF** under the **Orchestration Workflows** heading.
2. In the orchestration workflow Property Editor, click the **Data Mapping** tab.

   The **Working Data** step input should be visible under the name of the orchestration workflow (**DataMappingOrchWF**) in the **Working data** column. (If it is not, click a blank area of the orchestration workflow editor.)
3. Locate the **SuggestedMapping** data element, right-click the corresponding cell in the **Source elements** column, and then select **Clear Mapping**.

   **Tip:** You can also clear the mapping by selecting the appropriate cell in the **Source elements** column, right-clicking the down arrow, selecting **[no mapping]** in the **Select a source** popup, and then clicking **OK**.

## Creating a New Custom Event Definition

This topic describes how to create a new custom event definition in SBM Composer.

**Note:** For information about event definitions, see About Application Links and Event Definitions [page 28].

**To create a new custom event definition:**

1. Right-click **Application Links** under the orchestration name in App Explorer, and then select **Add New Event Definition**. The **Event Definition Configuration** dialog box opens.
2. Click **Create new custom event definition**, if this option is not already selected.
3. Complete the dialog box as described in Event Definition Configuration Dialog Box [page 44].

## Importing an Event Definition File for a New Custom Event Definition

This topic describes how to import an existing event definition (`.mtd`) file or `.wsdl` file to use in a new custom event definition.

**Note:** For information about event definitions, see About Application Links and Event Definitions [page 28].

**To import a file for a new custom event definition:**

1. Right-click **Application Links** under the orchestration name in App Explorer, and then select **Add New Event Definition**. The **Event Definition Configuration** dialog box opens.

2. Click **Create from event definition file**.
3. Complete the dialog box as described in Event Definition Configuration Dialog Box [page 44].

## Mapping an Orchestration Workflow to an Event Definition

To connect events with asynchronous orchestration workflows, you set up a mapping on the **Event Map** tab of the event definition Property Editor or orchestration workflow Property Editor. You can either generate an orchestration workflow from an event, or you can create an orchestration workflow and choose what event it is going to handle. In either case, the event `Extension` data becomes visible as part of the input data for the orchestration workflow and is available for processing by it.

You can add additional events to an orchestration workflow, but only if they are compatible with the existing event definition. To be compatible, the events must have the same `Extension` data. Generally, this means that the events are defined by the same event definition.

**To map a new orchestration workflow to an event definition:**

1. In App Explorer, under the **Application Links** heading, click the name of the event definition.
2. On the **Event Map** tab of the event definition Property Editor, click **Add**.
3. In the **Map Event Definition to Workflow** dialog box that opens, perform the following steps:
   a. Select the event, defined by the **Object type** and **Event type** values, that you want the new orchestration workflow to handle.
   b. Select **[New Workflow]** from the **Workflow** list.
   c. Click **OK**.

      An orchestration workflow named **Workflow** appears under the **Orchestration Workflows** heading, and the event definition is automatically mapped to it.

      **Note:** To see the mapping, view the information on the **Event Map** and **Data Mapping** tabs of the orchestration workflow Property Editor.

**To map an existing orchestration workflow to an event definition (first method):**

1. In App Explorer, under the **Orchestration Workflows** heading, click the name of a new orchestration workflow.
2. On the **Event Map** tab of the orchestration workflow Property Editor, click **Add**.

   **Note:** This tab is not available for synchronous orchestration workflows.

3. In the **Map Workflow to Event Definition** dialog box that opens, perform the following steps:

a. Select an event definition from the **Event definition** list. This list is read-only if there is only one event definition in the orchestration.

b. If the product supports more than one set of custom data, select the message to determine what will be received as inputs to the orchestration workflow.

c. Select the event, defined by the **Object type** and **Event type** values, that you want the new orchestration workflow to handle.

d. Click **OK**.

**To map an existing orchestration workflow to an event definition (second method):**

1. In App Explorer, under the name of an orchestration, under the **Application Links** heading, click the name of the event definition.
2. On the **Event Map** tab of the event definition Property Editor, click **Add**.
3. In the **Map Event Definition to Workflow** dialog box that opens, perform the following steps:

   a. Select the event, defined by the **Object type** and **Event type** values, that you want the new orchestration workflow to handle.

   b. Select the existing orchestration workflow from the **Workflow** list.

   c. Click **OK**.

## Using the Step Palette

Steps are the building blocks of an orchestration workflow. You select steps from the **Step Palette** to the right of the orchestration workflow editor, and then use the drag-and-drop operation to move them onto the line between the **Start** and **End** steps in the orchestration workflow. This creates the control flow structure that determines what the orchestration workflow does.

The following topics describe how to use some steps available on the **Step Palette**:

- Creating a Practice Process App for Using the Step Palette [page 72]
- Using the Calculate Step [page 73]
- Using the Decision Step [page 75]
- Using the ForEach Step [page 78]
- Using the While Step [page 82]
- Using the Service Step [page 85]
- Using the Group Step [page 96]

See Using the Scope, Throw, and Compensate Steps to Handle Faults From Web Services [page 96] for information about those steps.

## Creating a Practice Process App for Using the Step Palette

In this section, you create a new process app with one application workflow. Then you create the following orchestration workflows in the corresponding sections for each step: CalculateOrchWF, DecisionOrchWF, ForEachOrchWF, WhileOrchWF, and ServiceOrchWF. You use these orchestration workflows to practice using the following steps in the **Step Palette**: **Calculate**, **Decision**, **ForEach**, **While**, and **Service**.

**To create the practice process app:**

1. Start SBM Composer.
2. Click the Composer button, and then click **New**.
3. In the **Create New Process App** dialog box that opens, in the **Available Templates** pane, click **Application Process App**, and then click **Create**.
4. In the **Configure Process App** dialog box that opens, in the **Process app name** and **Application name** boxes, type `StepPaletteProcessApp`, and then click **OK.**

   The new process app appears in App Explorer.
5. In App Explorer, click the **All Items** filter.
6. Click **StepPaletteProcessApp**.
7. On the **StepPaletteProcessApp** tab, change the **Logical Name** to `StepPaletteApp`.
8. Under **Application Workflows**, click **StepPaletteProcessApp**.
9. On the **General** tab of the **Property Editor**, change the **Name** to `StepPaletteAppWF`, and then press the Tab key.
10. In the application workflow editor, double-click the name of the **New** state, type `State1`, and then press the Tab key.
11. Double-click the name of the **Submit** transition, type `Calculate`, and then press the Tab key.
12. In the **Common Items** section of the **Workflow Palette**, drag an **State** onto the application workflow editor and drop it to the right of the **State1** state.
13. Change the **Name** to `State2`, and then press the Tab key.
14. In the **Common Items** section of the **Workflow Palette**, drag a **Transition** onto the **State1** state, release the mouse button, and then click **State2**.
15. Change the **Name** of the **Transition** to `Decision`, and then press the Tab key.
16. Add three more states and three more transitions. Drop each new state somewhere after the previous state, and name the new states `State3`, `State4`, and `State5`. Name the transitions as follows:
    - Between **State2** and **State3**: `ForEach`
    - Between **State3** and **State4**: `While`

- Between **State4** and **State5**: `Service`

**Tip:** You can drop a state anywhere on the application workflow editor as long as you connect it, using a transition, to the previous state in the workflow.

17. In App Explorer, click **StepPaletteProcessApp**.
18. On the **Home** tab of the Ribbon, in the **New** group, click **Component**, and then select **Orchestration**.
19. In the **New Orchestration** dialog box, type `StepPaletteOrch` in the **Name** box, and then click **OK**.
20. Save the process app:
    a. On the Quick Access Toolbar, click the **Save locally** icon.

       A message reminds you that the design elements have been saved to the Local Cache only.

    b. Click **OK**.
21. If you closed SBM Composer after saving the process app:
    a. Start SBM Composer.

    b. Click the Composer button, and then click **Open**.

    c. In the **Open Process App** dialog box, select **StepPaletteProcessApp**, and then click **Open.**

    d. In App Explorer, select the **All Items** filter.

## Using the Calculate Step

You use the **Calculate** step to perform calculations on the data elements in an orchestration workflow. When you select a **Calculate** step, the **General** and **Options** tabs appear in the step Property Editor. The **General** tab contains the type, name, and description of the **Calculate** step. The **Options** tab contains the following three sections:

- The **Target** section defines the entity that receives the data.
- The **Expression** section defines the source of the data.
- The **Assignments** section enables you to combine multiple assignments (target-expression pairs) in a single **Calculate** step.

### Procedure Summary

1. Select an orchestration workflow to display it in the orchestration workflow editor.
2. In the **New Items** section of the **Step Palette**, drag and drop a **Calculate** step onto the line between the **Start** and **End** steps.
3. On the **General** tab of the Property Editor, you can change the name of the **Calculate** step and enter a description.

4. On the **Options** tab, in the **Target** section, enter a working data element or an expression that represents the data element that receives the value from the **Expression** section. You can use any of the functions, logical operators, or arithmetic operators available on the **Functions**, **Logical**, and **Operator** menus, respectively.

5. In the **Expression** section, enter an expression that represents the source of the data to be received by the target. You can use any of the functions, logical operators, or arithmetic operators available on the **Functions**, **Logical**, and **Operator** menus, respectively.

   For more information about creating expressions, see About the Expression Editor [page 32].

6. To add another assignment, right-click in the **Assignments** section and select **Add New Assignment**. Define the target and expression for the new assignment.

   You can also split assignments into multiple **Calculate** steps, as well as copy, rename, delete, and reorder assignments within the step.

## Creating an Empty Orchestration Workflow For the Calculate Step

In this exercise, you create an empty synchronous workflow. In the next section, you add and configure a **Calculate** step.

**To create an empty orchestration workflow for the Calculate step:**

1. In App Explorer, under **StepPaletteApp**, under **Application Workflows**, select **StepPaletteAppWF.**

2. In the application workflow editor, right-click the **Calculate** transition, and select **Show Actions** on the menu.

3. On the **Actions** tab of the Property Editor, click **New.**

   The **Action Wizard** asks, "Which type of action do you want to execute?"

   Under **Step 1**, **Orchestration Workflow** should be selected.

4. In the **Step 2** box, click the **and continue executing (asynchronous)** link, select **and wait for reply (synchronous)**, and then click **Next**.

   The **Action Wizard** asks, "When do you want to call the orchestration workflow?"

5. In the **Step 1** box, select **After**, do not change anything in the **Step 2** box, and then click **Next**.

   The **Action Wizard** asks, "Which orchestration workflow do you want to call?"

6. On the menu under **Step 1**, select **(Add new workflow...)**.

7. In the **Event With Reply** dialog box, do the following:

   a. Change the **Name** to `CalculateOrchWFWR`.

   b. On the **Orchestration** menu, select **StepPaletteOrch.**

   c. In the **Workflow** box, change the name to `CalculateOrchWF`.

   d. In the **Fields used by event** column, select the **Title** check box.

   e. In the **Fields returned by event** column, select the **Title** check box.

   f. Click **OK.**

8. In the **Action Wizard**, click **Finish**.

### Practicing With the Calculate Step

In this exercise, you add a **Calculate** step to the CalculateOrchWF orchestration workflow and define values for it. When CalculateOrchWF is invoked, it places the text `are here.` in the **Title** box of the **State1** state form after you click the **OK** button on the transition form between the **Submit** state and **State1**.

**To use the Calculate step in an orchestration workflow:**

1. In App Explorer, under **StepPaletteOrch**, under **Orchestration Workflows**, select **CalculateOrchWF**.

2. In the **New Items** section of the **Step Palette**, drag and drop a **Calculate** step onto the line between the **Start** and **End** steps.

3. On the **Options** tab of the Property Editor, in the **Target** section, enter the following expression: **`EventNoticeWithReply\Extension\Title`**

4. In the **Expression** section, enter the following function:
   `CONCAT(EventNoticeWithReply\Extension\Title, " are here.")`

   For more information about creating expressions, see About the Expression Editor [page 32].

5. Select the **End** step.

6. On the **Data Mapping** tab, under **Extension**, locate the **Title** data element, select the cell corresponding to the **Source elements** column, and then click the down arrow.

7. In the **Select a source** popup, expand **Inputs**, **EventNoticeWithReply**, and **Extension**; select **Title**; and then click **OK**.

## Using the Decision Step

You use the **Decision** step to decide between two or more possible outcomes. This step lets you branch the flow of control based on *rules*. Rules are expressions, as described in About the Expression Editor [page 32].

Using the **Insert New Branch** command, you can add branches to the **Decision** step. When you select a branch, the **General** and **Options** tabs appear in the Property Editor. The **General** tab provides a name and an optional description for the branch. The **Options** tab defines the rule associated with the branch. (The default branch, named **Otherwise**, has no rule. It is invoked if all the other branches are false.)

**Procedure Summary**

1. Select an orchestration workflow to display it in the orchestration workflow editor.

2. In the **New Items** section of the **Step Palette**, drag a **Decision** step onto the line between the **Start** and **End** steps.
3. On the **General** tab of the Property Editor, you can change the name of the **Decision** step and you can also enter a description.
4. To add a branch, right-click the **Decision** step, and then select **Insert New Branch** on the menu.

   A **Branch** is added above the **Otherwise** branch.
5. On the **General** tab of the Property Editor, you can change the name of the **Branch** and you can also enter a description.
6. On the **Options** tab of the Property Editor, in the **Rule** section, enter an expression that represents the rule for that branch. You can use any of the functions, logical operators, or arithmetic operators available on the **Functions**, **Logical**, and **Operator** menus, respectively.

   (For more information about expressions, see About the Expression Editor [page 32].)
7. Select the new (non-Otherwise) **Branch**.
8. Add a step or steps from the **Step Palette** to the **Branch** to define the actions that should be taken or the calculations that should be performed while the rule defined for the branch is true.
9. Repeat the previous step to add and configure other decision branches.

   **Tip:** You can switch from branch to branch by selecting a branch on the menu of the Property Editor. Also, when you right-click the name of a branch in the orchestration workflow editor, you can select menu items that let you duplicate and rename the branch, and to reorder branches.
10. Add a step or steps from the **Step Palette** to the **Otherwise** branch. You do not define a rule for this branch.

    **Tip:** To hide the branches as you work on another part of the orchestration workflow, click the minus sign to the left of the **Decision** step.

## Creating an Empty Orchestration Workflow For the Decision Step

In this exercise, you create an empty synchronous orchestration workflow. In the next section, you add and configure a **Decision** step.

**To create an empty orchestration workflow for the Decision step:**

1. In App Explorer, under **StepPaletteApp**, under **Application Workflows**, select **StepPaletteAppWF.**
2. In the application workflow editor, right-click the **Decision** transition, and select **Show Actions**.
3. On the **Actions** tab of the Property Editor, click **New.**

   The **Action Wizard** asks, "Which type of action do you want to execute?"

Under **Step 1**, **Orchestration Workflow** should be selected.

4. In the **Step 2** box, click the **and continue executing (asynchronous)** link, select **and wait for reply (synchronous)**, and then click **Next**.

   The **Action Wizard** asks, "When do you want to call the orchestration workflow?"

5. In the **Step 1** box, select **After**, do not change anything in the **Step 2** box, and then click **Next**.

   The **Action Wizard** asks, "Which orchestration workflow do you want to call?"

6. On the menu under **Step 1**, select **(Add new workflow...)**.

7. In the **Event With Reply** dialog box that opens:

   a. Change the **Name** to `DecisionOrchWFWR`.

   b. On the **Orchestration** menu, select **StepPaletteOrch.**

   c. In the **Workflow** box, change the name to `DecisionOrchWF`.

   d. In the **Fields used by event** column, select the **Title** check box.

   e. In the **Fields returned by event** column, select the **Title** check box.

   f. Click **OK.**

8. In the **Action Wizard**, click **Finish.**

## Practicing with the Decision Step

In this exercise, you add a **Decision** step to the DecisionOrchWF orchestration workflow and define values for it. When DecisionOrchWF is invoked, it inserts `One`, `Two`, or `Otherwise` (depending on your input) in the **Title** box of the **State2** state form after you click the **OK** button on the transition form between **State1** and **State2**. If you enter the number `1` and then click **OK**, `One` is inserted into the **Title** box. If you enter the number `2`, `Two` is inserted. If you enter any other text, `Otherwise` is inserted.

### To use the Decision step in an orchestration workflow:

1. In App Explorer, under **StepPaletteOrch**, under **Orchestration Workflows**, select **DecisionOrchWF**.

2. In the **New Items** section of the **Step Palette**, drag and drop a **Decision** step onto the line between the **Start** and **End** steps.

3. Right-click the **Decision** step, and then select **Insert New Branch**.

   A new branch is added above the **Otherwise** branch.

4. On the **General** tab of the Property Editor, change the **Name** to `One`, and then press the Tab key.

5. On the **Options** tab, in the **Rule** section, enter the following function:
   `NORMALIZESPACE(EventNoticeWithReply\Extension\Title)="1"`.

6. In the **New Items** section of the **Step Palette**, drag and drop a **Calculate** step onto the **One** branch.
7. On the **Options** tab of the **Calculate** step Property Editor, in the **Target** section, enter the following expression: `EventNoticeWithReply\Extension\Title`
8. In the **Expression** section, enter the following text, including the quotation marks: **`"One"`**
9. Right-click the **Decision** step, and then select **Insert New Branch**.

   A new branch is added between the **One** branch and the **Otherwise** branch.
10. On the **General** tab of the Property Editor, change the **Name** to **`Two`**, and then press the Tab key.
11. On the **Options** tab, in the **Rule** section, enter the following function: **`NORMALIZESPACE(`**`EventNoticeWithReply\Extension\Title`**`)`**`="2".`
12. In the **New Items** section of the **Step Palette**, drag and drop a **Calculate** step onto the **Two** branch.
13. On the **Options** tab of the Property Editor, in the **Expression** section, enter the following text, including the quotation marks: **`"Two"`**
14. In the **Target** section, enter the following expression: `EventNoticeWithReply\Extension\Title`
15. In the **New Items** section of the **Step Palette**, drag and drop a **Calculate** step onto the **Otherwise** branch.
16. On the **Options** tab of the Property Editor, in the **Expression** section, enter the following text, including the quotation marks: **`"Otherwise"`**
17. In the **Target** section, enter the following expression: **`EventNoticeWithReply\Extension\Title`**
18. Select the **End** step.
19. On the **Data Mapping** tab, expand **Extension**, locate the **Title** data element, select the cell corresponding to the **Source elements** column, and then click the down arrow.
20. In the **Select a source** popup, expand **DecisionOrchWF**, **Inputs**, **EventNoticeWithReply**, and **Extension**; select **Title**; and then click **OK.**

## Using the ForEach Step

The **ForEach** step repeats the operations that you specify for each element in a list or multi-item data element. These operations continue to execute until the last element has been processed. Then the orchestration workflow continues to the next step. The ForEach index counts the number of loops, and the ForEach item is the value of the data element at the end of a particular loop.

When you select a **ForEach** step, the **General** and **Options** tabs appear in the Property Editor. The **General** tab provides a name and optional description for the step. The **Options** tab defines the source for the multi-item data element.

**Note:** During loops, the Web service and subroutine inputs retain the data for all transactions. In some cases, this caching may cause issues for subsequent loops. For information on resetting the data collected in these inputs, refer to Resetting Data [page 18].

**Procedure Summary**

1. Select an orchestration workflow to display it in the orchestration workflow editor.

2. On the **Data Mapping** tab of the Property Editor, create an array of working data. (See Creating Arrays of Working Data [page 61].)

3. In the **New Items** section of the **Step Palette**, drag and drop a **ForEach** step onto the line between the **Start** and **End** steps.

   The lines from the **ForEach** step icon travel in two directions. One points to subsequent steps and the other loops back to the **ForEach** step icon. You place the steps or calculations that are to be repeated on the loop that points to the **ForEach** step icon.

4. On the **General** tab of the Property Editor, you can change the name of the **ForEach** step and you can also enter a description.

5. On the **Options** tab, in the **Source** section, enter an expression that describes the source of the data to be processed. You can use any of the functions, logical operators, or arithmetic operators available on the **Functions**, **Logical**, and **Operator** menus, respectively.

   For more information about expressions, see About the Expression Editor [page 32].

6. Add other steps from the **Step Palette** to the repeating section (the top loop) to define the actions that should be taken or calculations that should be performed while there are still members of the data element to be processed.

7. Add other steps from the **Step Palette** to the non-repeating section (the bottom loop) to define the actions that should be taken or calculations that should be performed after all of the members of the data element have been processed.

   **Tip:** To duplicate or delete a **ForEach** step, right-click the step, and then select the appropriate option. To hide the loops while you work on another part of the orchestration workflow, click the minus sign to the left of the **ForEach** step.

## Creating an Empty Orchestration Workflow for the ForEach Step

In this exercise, you create an empty synchronous orchestration workflow. In the next section, you add a **ForEach** step and then configure it.

**To create an empty orchestration workflow for the ForEach step:**

1. In App Explorer, under the **StepPaletteApp** heading, under **Application Workflows**, select **StepPaletteAppWF.**

2. In the application workflow editor, right-click the **ForEach** transition, and select **Show Actions**.
3. On the **Actions** tab of the Property Editor, click **New.**

   The **Action Wizard** asks, "Which type of action do you want to execute?"

   Under **Step 1**, **Orchestration Workflow** should be selected.
4. In the **Step 2** box, click the **and continue executing (asynchronous)** link, select **and wait for reply (synchronous)** on the menu, and then click **Next.**

   The **Action Wizard** asks, "When do you want to call the orchestration workflow?"
5. In **Step 1**, select **After**, do not change anything in the **Step 2** box, and then click **Next**.

   The **Action Wizard** asks, "Which orchestration workflow do you want to call?"
6. On the menu under **Step 1**, select **(Add new workflow....)**.
7. In the **Event With Reply** dialog box that opens:
   a. Change the **Name** to `ForEachOrchWFWR`.
   b. On the **Orchestration** menu, select **StepPaletteOrch.**
   c. In the **Workflow** box, change the name to `ForEachOrchWF`.
   d. In the **Fields used by event** column, select the **Title** check box.
   e. In the **Fields returned by event** column, select the **Title** check box.
   f. Click **OK.**
8. In the **Action Wizard**, click **Finish.**

## Practicing with the ForEach Step

In this exercise, you add a **ForEach** step to the ForEachOrchWF orchestration workflow and define values for it. When ForEachOrchWF is invoked, it appends an equals sign and the current loop index value to the default value of each data element in the list of data elements under `Array[]`. After `Array[5]` is processed, the resulting values are inserted in the **Title** box of the **State3** state form after you click the **OK** button on the transition form between **State2** and **State3**.

**To use the ForEach step in an orchestration workflow:**

1. In App Explorer, under **StepPaletteOrch**, under **Orchestration Workflows**, select **ForEachOrchWF**.
2. On the **Data Mapping** tab of the Property Editor, create an array of data elements as follows:
   a. Right-click the **WorkingData** step input, select **Add New**, and then select **Complex Type**.
   b. Change the name of the new data element (**ComplexType**) to `ArrayContainer`.

c. Right-click **ArrayContainer**, select **Add Child**, and then select **String**.

d. Change the name of the new data element (**String**) to `Array`.

e. Right-click **Array**, and then select **Properties Mode**.

f. In the properties area under **Misc**, select the **IsUnbounded** row.

g. Click the down arrow next to **False**, and then select **True**.

h. To close the properties area, right-click **Array[]**, and then select **Mapping Mode**.

i. Right-click **Array[]**, and then select **Add Array Element**. **Array[1]** is added to **Array[]**.

j. Repeat the previous step four more times. Array elements **Array[2]**, **Array[3]**, **Array[4]**, and **Array[5]** are added to **Array[]**, under **Array[1]**.

k. In the **Default value** column for **Array[1]**, type the word `One`; for **Array[2]**, type `Two`; for **Array[3]**, type `Three`; for **Array[4]**, type `Four`; and for **Array[5]**, type `Five`.

3. Create a ForEach index:

   a. Right-click the **WorkingData** step input, select **Add New**, and then select **Integer**.

   b. Change the name of the new data element (**Integer**) to `Index`.

   c. In the **Default value** column, type `-1`.

4. Create a ForEach item value:

   a. Right-click the **WorkingData** step input, select **Add New**, and then select **String**.

   b. Change the name of the new data element (**String**) to `Value`.

   c. In the **Default value** column, type `Test`.

5. In the **New Items** section of the **Step Palette**, drag and drop a **ForEach** step onto the line between the **Start** and **End** steps.

6. On the **Options** tab of the Property Editor, in the **Source** section, enter the following expression: `ArrayContainer.Array`

7. In the **New Items** section of the **Step Palette**, drag and drop a **Calculate** step onto the line that loops back to the **ForEach** step.

8. On the **General** tab of the Property Editor, change the **Name** to `SetIndex`, and then press the Tab key.

9. On the **Options** tab, in the **Target** section, type the following: `Index`

10. In the **Expression** section, enter the following expression: `ForEach.index`

11. In the **New Items** section of the **Step Palette**, drag and drop a **Calculate** step onto the line that loops back to the **ForEach** step, to the right of the **SetIndex** step.
12. On the **General** tab of the Property Editor, change the **Name** to `SetValue`, and then press the Tab key.
13. On the **Options** tab, in the **Target** section, type `Value`
14. In the **Expression** section, enter `ForEach\item`
15. In the **New Items** section of the **Step Palette**, drag and drop a **Calculate** step onto the line that loops back to the **ForEach** step, to the right of the **SetValue** step.
16. On the **General** tab of the Property Editor, change the **Name** to `MakeTitle`, and then press the Tab key.
17. On the **Options** tab, in the **Target** section, enter the following: `EventNoticeWithReply\Extension\Title`
18. In the **Expression** section, enter the following:
    ```
    CONCAT(EventNoticeWithReply\Extension\Title," ", STRING(Index)," =",
    STRING(Value), " ")
    ```
19. Select the **End** step.
20. On the **Data Mapping** tab of the Property Editor, under **Extension**, locate the **Title** data element, select the corresponding cell in the **Source elements** column, and then click the down arrow.
21. In the **Select a source** popup, expand **Inputs**, **EventNoticeWithReply**, and **Extension**; select **Title**, and then click **OK.**

## Using the While Step

The **While** step repeatedly executes the operations you specify while the conditions defined in the rule are true, or until the conditions are false. The rule is an expression, as described in About the Expression Editor [page 32].

When you click a **While** step, the **General** and **Options** tabs appear in the Property Editor. The **General** tab provides a name and an optional description for the step. The **Options** tab defines the rule for the step.

**Note:** During loops, the Web service and subroutine inputs retain the data for all transactions. In some cases, this caching may cause issues for subsequent loops. For information on resetting the data collected in these inputs, refer to Resetting Data [page 18].

### Procedure Summary

1. Select an orchestration workflow to display it in the orchestration workflow editor.
2. In the **New Items** section of the **Step Palette**, drag and drop a **While** step onto the line between the **Start** and **End** steps.
3. On the **General** tab of the Property Editor, you can change the name of the **While** step and you can also enter a description.

4. On the **Options** tab, in the **Rule** section, enter an expression that represents the rule that must hold true for the **While** step actions to repeat. You can use any of the functions, logical operators, or arithmetic operators available on the **Functions**, **Logical**, and **Operator** menus, respectively.

   For more information about expressions, see About the Expression Editor [page 32].

5. Add other steps from the **Step Palette** to the repeating section (the top loop) to define the actions that should be taken or calculations that should be performed while the **While** step is true.

6. Add other steps from the **Step Palette** to the nonrepeating section (the bottom loop) to define the actions that should be taken or calculations that should be performed if the **While** step is false.

   **Tip:** To duplicate or delete the **While** step, right-click the step, and then select the appropriate options on the menu. To hide the loops as you work on another part of the orchestration workflow, click the minus sign to the left of the **While** step.

## Creating an Empty Orchestration Workflow For the While Step

In this exercise, you create an empty synchronous orchestration workflow with reply. In the next section, you add a **While** step and then configure it.

**To create an empty orchestration workflow for the While step:**

1. In App Explorer, under **StepPaletteApp**, under **Application Workflows**, select **StepPaletteAppWF.**

2. In the application workflow editor, right-click the **While** transition, and select **Show Actions.**

3. On the **Actions** tab of the Property Editor, click **New.**

   The **Action Wizard** asks, "Which type of action do you want to execute?"

   Under **Step 1**, **Orchestration Workflow** should be selected.

4. In the **Step 2** box, click the **and continue executing (asynchronous)** link, select **and wait for reply (synchronous)** on the menu, and then click **Next**.

   The **Action Wizard** asks, "When do you want to call the orchestration workflow?"

5. In the **Step 1** box, select **After**, do not change anything in the **Step 2** box, and then click **Next**.

   The **Action Wizard** asks, "Which orchestration workflow do you want to call?"

6. On the menu under **Step 1**, select **(Add new workflow...)**.

7. In the **Event With Reply** dialog box that opens:

   a. Change the **Name** to `WhileOrchWFWR`.

   b. On the **Orchestration** menu, select **StepPaletteOrch.**

   c. In the **Workflow** box, change the name to `WhileOrchWF`.

   d. In the **Fields used by event** column, select the **Title** check box.

   e. In the **Fields returned by event** column, select the **Title** check box.

   f. Click **OK.**

8. In the **Action Wizard**, click **Finish**.

## Practicing With the While Step

In this exercise, you add a **While** step to the WhileOrchWF orchestration workflow and define values for it. When WhileOrchWF is invoked, it counts the number of the strings in the **Title** box that consist of the letters `one`. If you type `oneoneoneoneone` in the **Title** box, the following text is inserted in the **Title** box of the **State3** state form when you click the **OK** button on the transition form between **State2** and **State3**: `I found 5 ones.`

**To use the While step in an orchestration workflow:**

1. In App Explorer, under **StepPaletteOrch**, under **Orchestration Workflows**, select **WhileOrchWF**.
2. On the **Data Mapping** tab of the Property Editor, create a loop counter as follows:
   a. Right-click the **Working Data** step input, select **Add New**, and then select **Integer**.
   b. Change the name of the new data element (**Integer**) to **`LoopCounter`**.
   c. In the **Default value** column, type the number **`0`**.
3. Create a temporary storage area for the working title as follows:
   a. Right-click the **Working Data** step input, select **Add New**, and then select **String**.
   b. Change the name of the new data element (**String**) to **`WorkingTitle`**.
   c. For the **WorkingTitle** data element, select the cell corresponding to the **Source elements** column, and then click the down arrow.
   d. In the **Select a source** popup, expand **Inputs**, **EventNoticeWithReply**, and **Extension**; select **Title**; and then click **OK**.
4. In the **New Items** section of the **Step Palette**, drag a **While** step onto the orchestration workflow editor, and drop it between the **Start** and **End** steps.
5. On the **Options** tab of the Property Editor, in the **Rule** section, enter the following function: `CONTAINS(WorkingTitle,"one")`
6. In the **New Items** section of the **Step Palette**, drag and drop a **Calculate** step onto the top loop (the repeating section) of the **While** step.
7. On the **General** tab of the Property Editor, change the **Name** to **`Counter`**, and then press the Tab key.
8. On the **Options** tab, in the **Target** section, enter the following: **`LoopCounter`**

9. In the **Expression** section, enter the following: `LoopCounter+1`

10. In the **New Items** section of the **Step Palette**, drag and drop a **Calculate** step onto the top loop of the **While** step, to the right of the **Counter** step.

11. On the **General** tab of the Property Editor, change the **Name** to `Title`, and then press the Tab key.

12. On the **Options** tab, in the **Expression** section, enter the following: `SUBSTRINGAFTER(WorkingTitle,"one")`

13. In the **Target** section, enter the following: `WorkingTitle`

14. In the **New Items** section of the **Step Palette**, drag and drop a **Calculate** step onto the bottom loop (the non-repeating section) of the **While** step.

15. On the **General** tab of the Property Editor, change the **Name** to `EndTitle`, and then press the Tab key.

16. On the **Options** tab, in the **Expression** section, enter the following function: `CONCAT("I found",STRING(LoopCounter)," ones.")`

17. In the **Target** section, enter the following: `WorkingTitle`

18. Select the **End** step.

19. On the **Data Mapping** tab, expand **Extension**, locate the **Title** data element, select the corresponding cell in the **Source Elements** column, and then click the down arrow.

20. In the **Select a source** popup, expand **WhileOrchWF**, **Inputs**, **EventNoticeWithReply**, and **Extension**; select **Title**; and then click **OK**.

## Using the Service Step

You use the **Service** step to call a Web service from an orchestration workflow. On the Property Editor for the **Service** step, you specify the WSDL file that describes the Web service and which of its defined operations to use.

When you select a **Service** step, the **General** and **Data Mapping** tabs appear in the Property Editor. The **General** tab provides a name and an optional description for the step and specifies which Web service and operation are to be used. The **Data Mapping** tab defines the default values or source elements for the inputs to the Web service.

### Procedure Summary

1. Select an orchestration workflow to display it in the orchestration workflow editor.

2. In the **Configured Items** section of the **Step Palette**, drag and drop a **Service** step onto the line between the **Start** and **End** steps.

3. On the **General** tab of the Property Editor, you can change the name of the **Service** step and you can also enter a description.

4. On the **Service** menu, select the WSDL file for the Web service that you want to use.

5. On the **Data Mapping** tab, you can identify the inputs, or data elements, that are required by the Web service for the selected operation by the *required* flag that appears on the icon associated with the data element.

   If you do not see the required flag, right-click in any part of the data mapping area and select **Show Required Flag**.

6. To define default values for any of the Web service inputs, enter them in the **Default values** column.

7. To map the source values for a specific input using suggested mappings, see Setting Source Values Using Suggested Mappings [page 64].

   The outputs of the Web service appear as potential inputs for subsequent steps in the orchestration workflow.

   **Tip:** To duplicate a **Service** step, right-click the step, and then select **Duplicate**.

## Creating an Empty Orchestration Workflow For the Service Step

In this exercise, you create an empty synchronous orchestration workflow with reply. In the next section, you add a **Service** step and then configure it.

**To create an empty orchestration workflow for the Service step:**

1. In App Explorer, under **StepPaletteApp**, under **Application Workflows**, select **StepPaletteAppWF.**

2. In the application workflow editor, right-click the **Service** transition, and select **Show Actions**.

3. On the **Actions** tab of the Property Editor, click **New.**

   The **Action Wizard** asks, "Which type of action do you want to execute?"

   Under **Step 1**, **Orchestration Workflow** should be selected.

4. In the **Step 2** box, click the **and continue executing (asynchronous)** link, select **and wait for reply (synchronous)**, and then click **Next**.

   The **Action Wizard** asks, "When do you want to call the orchestration workflow?"

5. In the **Step 1** box, select **Before**, and then click **Next**.

   The **Action Wizard** asks, "Which orchestration workflow do you want to call?"

6. On the menu under **Step 1**, select **(Add new workflow...)**.

7. In the **Event With Reply** dialog box that opens:

   a. Change the **Name** to `ServiceOrchWFWR`.

   b. On the **Orchestration** menu, select **StepPaletteOrch.**

   c. In the **Workflow** box, change name to `ServiceOrchWF`.

   d. In the **Fields used by event** column, select the **Title** check box.

    e. In the **Fields returned by event** column, select the **Title** check box.
    f. Click **OK.**

8. In the **Action Wizard**, click **Finish**.
9. In App Explorer, under **StepPaletteApp**, under **Orchestration Links**, select **ServiceOrchWFWR**.
10. On the **ServiceOrchWFWR** tab:
    a. In the **Fields used by event** column, select the **Item Id** check box. The **Title** check box should already be selected.
    b. In the **Fields returned by event** column, the **Title** check box should be selected.

## Practicing with the Service Step

In this exercise, you add a **Service** step to the ServiceOrchWF orchestration workflow and define values for it. When ServiceOrchWF is invoked, it inserts your login ID in the **Title** box of the transition page between **State4** and **State5** when you click the **Service** button on the **State4** state form.

**To use the Service step in an orchestration workflow:**

1. In App Explorer, under **StepPaletteOrch**, under **Orchestration Workflows**, select **ServiceOrchWF**.
2. In the **New Items** section of the **Step Palette**, drag a **Service** step onto the orchestration workflow editor, and drop it between the **Start** and **End** steps.
3. On the **General** tab of the Property Editor, on the **Service** menu, select the **sbmappservices72** Web service.
4. On the **Operation** menu, select the **GetItem** Web service operation.
5. On the **Data Mapping** tab, expand the **Auth** data element, and in the **Default value** column, enter your SBM user ID and password.
6. Still on the **Data Mapping** tab, locate the **TableId_ItemId** data element, select the corresponding cell in the **Source elements** column, and then click the down arrow.
7. In the **Select a source** popup, expand **Inputs**, **EventNoticeWithReply**, and **Extension**; select **ItemId_TableRecId**; and then click **OK**.
8. In the orchestration workflow editor, select the **End** step.
9. On the **Data Mapping** tab of the Property Editor, under **Extension**, locate the **Title** data element, select the corresponding cell in the **Source elements** column, and then click the down arrow.
10. In the **Select a source** popup, expand **sbmappservices72_GetItem**, **Outputs**, **GetItemResponse**, **return**, **item**, and **createdBy**; select **loginId**, and then click **OK.**

## Data Mapping Tab of the Service Step Property Editor

The **Data Mapping** tab of the **Service** step Property Editor is available when you select a **Service** step in an orchestration workflow in the orchestration workflow editor.

| Element | Description |
|---|---|
| Step inputs | This column lists the inputs to the Web service associated with the selected **Service** step.<br>Right-click a step input to display a menu of applicable commands. |
| Source elements | This column shows the source of each of the step inputs.<br>Select a cell in this column and click the down arrow that appears to open the **Select a source** popup, which offers options beyond the suggested mappings.<br>Right-click a source element to display a menu of applicable commands. |
| Default value | This column allows you to view or edit the default values, if any, for the step inputs.<br>**Note:** Any value specified in the **Source elements** column overrides the value in the corresponding **Default value** column. |
| *Vertical divider* | Clicking the vertical divider to the right of the Property Editor switches the Property Editor between mapping mode and properties mode. Properties mode displays additional information for the selected data element (such as its type and namespace). Drag the divider to the left to expand the properties mode panel.<br>To see the data sent with events raised during the execution of application workflow transitions, click on a blank area of the orchestration workflow editor, and then click the vertical divider. |

## Mapping SOAP Header Data

SBM Composer lets you map SOAP Header information, if the WSDL file you are using defines SOAP Header data. This data is listed on the **Data Mapping** tab of a **Service** step in the **Step Inputs** column, and its name ends in _Envelope.

**Note:** SBM Composer does not support `headerfault message` elements.

**Note:** See Using SOAP Headers to Enable WS-Security [page 90] for information about using SOAP header data for security purposes.

**To map SOAP header data:**

1. In the orchestration workflow editor, select the **Service** step for which you want to map data that will be sent in the SOAP Header. The Web service's WSDL file must define SOAP Header data.
2. On the **General** tab of the Property Editor, make sure that the **Service** step is associated with a Web service and an operation.
3. On the **Data Mapping** tab of the Property Editor, expand the **_Envelope** step input.

   All defined SOAP Header data is listed as data elements under this step input. Step inputs that appear below and at the same level as the **_Envelope** step input and all of their data elements are defined in the SOAP Body.
4. For each of the required data elements, do one of the following:
   - In the **Source elements** column:
     1. Click the down arrow.
     2. In the **Select a source** popup, select the data element to be used from another location in the workflow, such as the output of a **Service** step or a working data element.
     3. Click **OK**.
   - In the **Default values** column, type the path of the data element to be used, such as `SomeWorkflow\FirstCategory\SecondItem`.

## Using Basic Access Authentication

Some Web services require authentication, such as a user name and a password. In one common method, called *basic access authentication*, this information is included in the HTTP header of the Web service call. SBM Composer supports basic access authentication for individual Web service operations within an orchestration workflow.

For example, you can create two working data elements to hold a user name and a password, enter text as default values, and set up the **Authentication** input's **UserName** and **Password** components for multiple **Service** steps to use those working data elements. That way, if the user name or password changes, you only need to update the two working data elements to keep all of the **Service** steps set up to call the Web service.

**Important:** The **UserName** and **Password** values are ignored if an administrator configures the basic access authentication type for the endpoint pointing to the Web service manually in Application Repository. In this case, the credentials specified in Application Repository are used instead of those specified in this procedure. Process apps are more portable if the Application Repository credentials are used, because you can bind to the appropriate endpoint for your environment when you deploy.

**To use basic access authentication:**

1. Select an orchestration workflow to display it in the orchestration workflow editor.
2. Select a **Service** step for which you want to set up basic access authentication.
3. On the **General** tab of the Property Editor, make sure the **Service** step is associated with a Web service and an operation, and that the Web service supports basic access authentication.
4. In the orchestration workflow editor, right-click the **Service** step, and then select **Use Basic Access Authentication**.

   This adds the **Authentication** step input (with the **UserName** and **Password** data elements), to the **Data Mapping** tab of the Property Editor. These data elements can be used for the **Service** step and for any other **Service** steps in the orchestration workflow that are associated with the same Web service operation.
5. For each of the **Service** steps to which the **Authentication** step input was added, do one of the following for both the **UserName** and **Password** data elements:
   - In the **Source elements** column:
     1. Click the down arrow.
     2. In the **Select a source** popup that opens, select the source value to be used from some location in the workflow, such as the output of another **Service** step or a working data element.
     3. Click **OK**.
   - In the **Default values** box, type the value to be used.

## Using SOAP Headers to Enable WS-Security

WS-Security ensures the identify, integrity, and security of a SOAP message. It is applied at the Web service layer, as opposed to basic access authentication, which is applied at the HTTP transport layer. SOAP messages are typically exchanged over HTTP, so you can use basic access authentication for them. However, WS-Security provides an extra level of security for messages that take a complicated path or that use a non-HTTP transport mechanism.

WS-Security elements are embedded within the SOAP message `<env:Header>` element. These take the form of one or more `<wss:security>` elements that contain the appropriate security information as required by the particular service deployment.

Although it is possible to declare WS-Security elements in a Web service WSDL file, this is not typically done. Instead, WS-Policy is used to document the security that a particular service requires. SBM does not currently support WS-Policy. However, it is possible to use data mapping in SBM Composer to create SOAP header elements.

To do this, there must be at least one declared header in the Web service WSDL. Any header will do; It does not have to be a WS-Security header. Because headers are not typically present in Web service WSDLs, you probably need to make a local copy of the WSDL and then edit it to add a placeholder header.

**CAUTION:**

Do not import the WS-Security schema into your copy of the WSDL or into SBM Composer to get the WS-Security `<env:Header>` elements and types. The way the schema is declared is problematic, and there is inadequate support for it from SBM Composer.

**To enable WS-Security:**

1. Create a `Placeholder` element and a `Header` message in the WSDL file as shown in bold and italics in the following example. The `Placeholder` element causes SBM Composer to provide an `_Envelope` element in the SOAP request message. The `Header` message will contain the WS-Security data, which mirrors the `UsernameToken` element in the WS-Security schema.

   **Note:** This example contains only one operation. If there are more operations you will need to declare a header for each one you want to use. You can typically reference the same header message declaration.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
→xmlns:tns="http://www.example.org/SOAPHeaderExample_1/"
→xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd=
→"http://www.w3.org/2001/XMLSchema" name="SOAPHeaderExample_1"
→targetNamespace="http://www.example.org/SOAPHeaderExample_1/">
  <wsdl:types>
    <xsd:schema targetNamespace="http://www.example.org/
→SOAPHeaderExample_1/">
      <xsd:element name="NewOperation">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="in" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="NewOperationResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="out" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
```

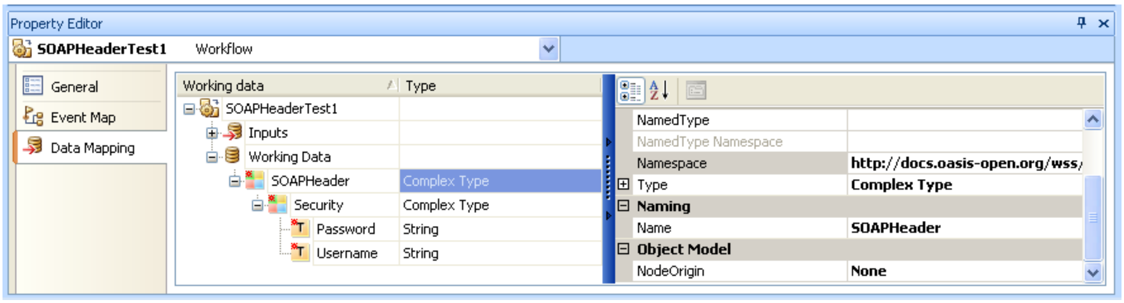
```
     <xsd:element name="Placeholder" type="xsd:string"/>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="NewOperationRequest">
    <wsdl:part element="tns:NewOperation" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="NewOperationResponse">
    <wsdl:part element="tns:NewOperationResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="Header">
    <wsdl:part element="tns:Placeholder" name="placeholder"/>
  </wsdl:message>
  <wsdl:portType name="SOAPHeaderExample_1">
    <wsdl:operation name="NewOperation">
      <wsdl:input message="tns:NewOperationRequest"/>
      <wsdl:output message="tns:NewOperationResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SOAPHeaderExample_1SOAP" type="tns:
→SOAPHeaderExample_1">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/
→soap/http"/>
    <wsdl:operation name="NewOperation">
      <soap:operation soapAction="http://www.example.org/
→SOAPHeaderExample_1/NewOperation"/>
    <wsdl:input>
      <soap:body use="literal"/>
      <soap:header message="tns:Header" part="placeholder"
→use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="SOAPHeaderExample_1">
    <wsdl:port binding="tns:SOAPHeaderExample_1SOAP" name=
→"SOAPHeaderExample_1SOAP">
      <soap:address location="http://www.example.org/"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

**Tip:** In the example above, note that the `Placeholder` namespace matches the namespace that is used for `http://www.w3.org/2001/XMLSchema` (**xsd**). If the namespace was `<wsdl:definitions` **`xmlns`**`:`**`s`**`="http://www.w3.org/2001/XMLSchema"`, you would insert `<`**`s`**`:element name="Placeholder" type="`**`s`**`:string"/>` to match the namespace.

2. Import the WSDL file into SBM Composer:
   a. Select the **Web Services** header under the orchestration in App Explorer.
   b. Right-click and then select **Add New Web Service**.

c. In the **Web Service Configuration** dialog box that opens, browse to the WSDL file and then click **OK**.

3. Click the **Data Mapping** tab in the orchestration workflow editor, and add working data elements as shown below to store the WS-Security data. Use the exact names and types, because they mirror the structure of the particular WS-Structure you need to use. Declare each element with the following namespace: `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wsssecurity-secext-1.0.xsd`.



| Working data | Type |
|---|---|
| SOAPHeaderTest1 | |
| Inputs | |
| Working Data | |
| SOAPHeader | Complex Type |
| Security | Complex Type |
| Password | String |
| Username | String |

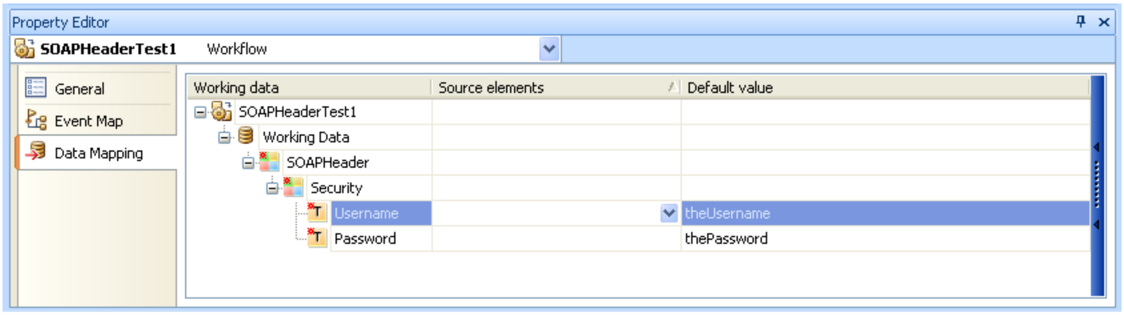
| | |
|---|---|
| NamedType | |
| NamedType Namespace | |
| Namespace | http://docs.oasis-open.org/wss/ |
| Type | Complex Type |
| **Naming** | |
| Name | SOAPHeader |
| **Object Model** | |
| NodeOrigin | None |

**Note:** SBM Composer has some limitations as to the structures it can create. You can usually work around these limitations by declaring the structure you need in the WSDL schema. Due to problems with the WS-Security schema, you cannot use it directly and will need to recreate the appropriate WS-Security structure in the Web service WSDL.

In this example used in this topic, the WS-Security `UsernameToken` structure is used:

```
<wsse:Security>
    <wsse:UsernameToken>
        <wsse:Username>theUsername</wsse:Username>
        <wsse:Password>thePassword</wsse:Password>
    </wsse:UsernameToken>
</wsse:Security>
```

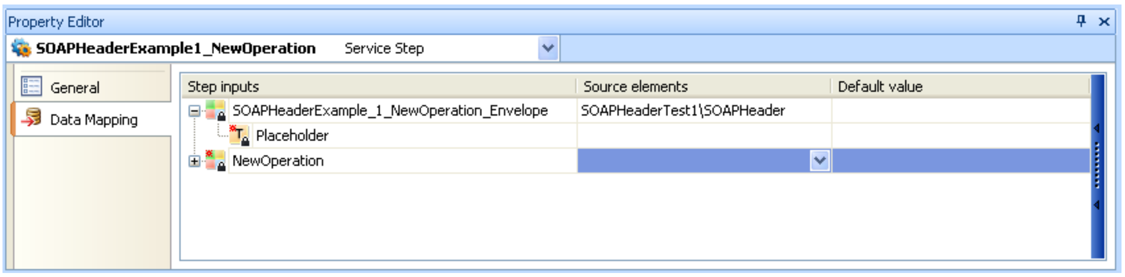
4. Type a default value for the **Username** and **Password**, or click in the **Source elements** column and then map to values. If you do not do this, the values will not appear in the SOAP message.



| Working data | Source elements | Default value |
|---|---|---|
| SOAPHeaderTest1 | | |
| Working Data | | |
| SOAPHeader | | |
| Security | | |
| Username | | theUsername |
| Password | | thePassword |

5. In the orchestration workflow editor, select the **Service** step.

6. On the **Data Mapping** tab of the step Property Editor, map the `_Envelope` step input to the `SOAPHeader` working data element that will store the security data.


Property Editor

SOAPHeaderExample1_NewOperation Service Step

General
Data Mapping

| Step inputs | Source elements | Default value |
|---|---|---|
| SOAPHeaderExample_1_NewOperation_Envelope | SOAPHeaderTest1\SOAPHeader | |
| Placeholder | | |
| NewOperation | | |

The following example SOAP request shows the result of this procedure.

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/
→envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <env:Header>
      <ns9:Security xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/
→business-process/" xmlns:defaultNS="http://SOAPHeaderTest1"
xmlns:defaultNS1="http://www.example.org/SOAPHeaderExample_1/"
xmlns:ns9="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
→wssecurity-secext-1.0.xsd" xmlns:tns="http://SOAPHeaderTest1">
         <ns9:Username>theUsername</ns9:Username>
         <ns9:Password>thePassword</ns9:Password>
      </ns9:Security>
   </env:Header>
   <env:Body>
      <defaultNS1:NewOperation
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:defaultNS="http://www.example.org/SOAPHeaderExample_1/"
xmlns:defaultNS1="http://www.example.org/SOAPHeaderExample_1/"
xmlns:ns8="http://www.example.org/SOAPHeaderExample_1/">
         <in xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://www.eclipse.org/alf/schema/EventBase/1"
xmlns:s="http://www.eclipse.org/alf/schema/EventBase/1"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">hello</in>
      </defaultNS1:NewOperation>
   </env:Body>
</env:Envelope>
```

## Using Dynamic Endpoints

SBM Composer provides the option of setting the endpoints for individual Web service operations dynamically, when the orchestration workflow is run, to a URL returned by a different **Service** step.

For example, a login **Service** step returns the URL of a server to be used for subsequent **Service** steps. For each subsequent **Service** step, you can use the procedure below to map the **DynamicEndpointURL** input to the server URL included in the output of the login **Service** step.

**To use dynamic endpoints:**

1. Select an orchestration workflow to display it in the orchestration workflow editor.
2. Select a **Service** step for which you want to set a dynamic endpoint.

3. On the **General** tab of the Property Editor, make sure the **Service** step is associated with a Web service and an operation.

4. Right-click the **Service** step in the orchestration workflow, and then select **Use Dynamic Endpoint**.

   This adds the **DynamicEndpointURL** step input to the **Data Mapping** tab of the Property Editor. This data element can be used for the **Service** step and for any other **Service** steps in the orchestration workflow that are associated with the same Web service operation.

5. For each **Service** step to which the **DynamicEndpointURL** step input has been added:

   a. Select the **Source elements** column in the cell corresponding to the new **DynamicEndpointURL** step input, and then click the down arrow.

   b. In the **Select a Source** popup, select the URL to be used.

   c. Click **OK**.

**Note:** You can return a Web service operation to using a static endpoint rather than a dynamic endpoint. To do this, in the orchestration workflow editor, right-click a **Service** step associated with the Web service operation, and then select **Use Dynamic Endpoint**.

## Running the StepPalette Process App

You are now ready to deploy and run the StepPalette process app.

**To deploy and run the StepPalette process app:**

1. Publish and deploy the StepPalette process app.

   See Step 6: Publish the Process App [page 192] and Step 7: Deploy the Process App [page 192] for instructions.

2. Log on to SBM Work Center.

3. Click the **StepPaletteApp** icon.

4. Click **+New**.

5. On the **Browse** tab, click the **StepPaletteAppWF Project** link.

   The **Submit** transition form opens.

6. On the transition form, type `The process apps` in the **Title** box, and then click the **OK** button.

   The Calculate orchestration workflow is invoked, and it prepends the text in the **Title** box of the **State1** state form, which now contains `The process apps are here.`

7. On the **State1** state form, click the **Decision** button.

   The transition form between the **State1** and **State2** state forms opens.

8. On the transition form, delete the text in the **Title** box, type `1`, `2`, or any other text, and then click the **OK** button.

The Decision orchestration workflow is invoked, and the results are displayed in the **Title** box of the **State2** state form. If you entered `1` in the **Title** box, then `One` is displayed. If you entered `2`, the **Title** box contains `Two`. And if you entered any other text, the **Title** box contains `Otherwise`.

9. On the **State2** state form, click the **ForEach** button.

   The transition form between the **State2** and **State3** state forms opens.

10. On the transition form, delete the text in the **Title** box, type **`Results:`** and a space, and then click the **OK** button.

    The ForEach orchestration workflow is invoked, and the results are displayed in the **Title** box of the **State4** form. The **Title** box contains `Results: One = 1 Two = 2 Three = 3 Four = 4 Five = 5`.

11. On the **State3** state form, click the **While** button.

    The transition form between the **State3** and **State4** state forms opens.

12. Delete the text in the **Title** box, enter the word **`one`** zero to eight times, and then click **OK**.

    The While orchestration workflow is invoked, and the results are displayed in the **Title** box of the **State3** state form. If *n* is the number of times you entered `one` in the **Title** box, `I found` *`n`* `ones.` is displayed.

13. On the **State4** form, click the **Service** button.

    The transition form between the **State4** and **State5** state forms opens.

14. On the transition form, click the **OK** button.

    The Service orchestration workflow is invoked, and your login ID appears in the **Title** box of the **State5** state form.

### Using the Group Step

The **Group** step is used to group steps logically. This can simplify large workflows because a group of related steps can be organized in a collapsible group with a descriptive label.

The **Group** step provides no functionality, unlike the **Scope** step, which is used to group steps and also handle faults that occur during Web service execution. It is recommended that you use the **Group** step to organize steps that do not involve Web service execution; if you use a **Scope** step to organize steps but do not include fault handling, validation warnings will be sent.

## Using the Scope, Throw, and Compensate Steps to Handle Faults From Web Services

The **Scope**, **Throw**, and **Compensate** steps in SBM Composer graphically represent BPEL fault-handling elements. You use these steps to handle Web service faults. This section provides a brief overview of BPEL fault handling and how it is implemented in SBM Composer.

## Types of Web Service Faults

The two types of Web service faults are "named" faults and "generic" faults. A named fault is associated with a specific fault situation, such as entering an incorrectly formatted user name or attempting to withdraw money from a bank account that is inactive. A generic fault is returned by a Web service if the service does not specify any named faults. Web services return faults in SOAP messages that contain SOAP faults. (See About SOAP Messages [page 36].)

If a Web service fault is not handled, or caught, it can stop the entire business process. Unhandled Web service faults are the most common causes of orchestration workflow failures. This section explains the various ways you can use the fault-handling components in SBM Composer to keep your business processes flowing smoothly.

## Scope Step

The **Scope** step lets you group related activities. It contains a **FaultHandler** section and a **CompensationHandler** section. Scopes can be nested, that is, one scope can enclose another scope. Orchestration workflows are contained within implicit, or global, scopes.

### FaultHandler Section

If a Web service inside of a scope generates a SOAP fault or if an internal BPEL fault occurs, the SBM Orchestration Engine checks whether a fault handler is defined for the scope. The catch branch within the fault handler "catches" the faults so they can be handled. The fault handler can contain an unlimited number of catch branches.

If the SBM Orchestration Engine finds a fault handler, it selects the catch branch with the value that best matches the fault according to the following rules:

- You can right-click a **Service** step in a scope and select **Add Catch Branches to Scope** to add a catch branch for each fault the step can throw. The SBM Orchestration Engine selects the appropriate catch branch when the step throws a fault.
- **ServiceFlowFault** is the only fault that a **Throw** step can throw. The catch branch with **ServiceFlowFault** selected as the **Fault name** on the **General** tab of its Property Editor is selected for **Throw** steps that have a **Fault message** displayed on the **General** tab of the step Property Editor.

  **Important:** If there are several **Throw** steps within the same **Scope**, each one takes this branch, even if they have different **Fault message** values.
- If no catch or catch all branches are selected, the fault is not caught and is thrown back to the immediately enclosing scope.

### CompensationHandler Section

The SBM Orchestration Engine invokes compensation handlers when there is a **Compensate** step in the enclosing scope. You can use any steps in the **Step Palette** to create a compensation handler. For an example, see Using the Compensate Step [page 124].

## Throw Step

During the execution of an orchestration workflow, errors might prevent the workflow from completing. For example, suppose you design an orchestration workflow that transfers money between two accounts by invoking a few Web services. However, the first Web service call returns an "invalid account number" fault.

The orchestration workflow stops, unless it contains some logic in the fault handler of the scope that encloses the Web service. For example, you might want to inform the user that he or she entered an invalid account number. The best way to do this by using the **Throw** step to generate a custom error message such as "The account number you entered is not valid. Please try again."

### Orchestration Workflows as Web Services

An orchestration workflow itself is invoked as a Web service, although this Web service can only be accessed by the SBM Application Engine. An orchestration workflow is invoked in one of two modes: Request Only (EventNotice) or Request and Respond (EventNoticeWithReply). EventNotice is used when an asynchronous orchestration workflow is invoked, and EventNoticeWithReply is used when a synchronous orchestration workflow is invoked.

A Web service could respond with a SOAP fault if something went wrong during its execution. For each orchestration workflow that is invoked, a WSDL file is generated for its corresponding Web service.

The following sample code is a EventNoticeWithReply Web service operation that is defined in a WSDL file. It shows that the Web service for the orchestration workflow can return a SOAP fault named ServiceFlowFault. All orchestration workflows can return, or throw, a ServiceFlowFault. In fact, this is the only SOAP fault that they can throw.

```
<wsdl:operation name="EventNoticeWithReply">
  <wsdl:input message="tns:XxxxxEventNoticeWithReply" />
  <wsdl:output message="tns:XxxxxEventNoticeWithReplyResponse" />
  <wsdl:fault name="ServiceFlowFault" message="tns:ServiceFlowFault" />
  </wsdl:operation>
```

### Why Use a Throw Step?

Use a **Throw** step when you want an orchestration workflow to explicitly signal an internal fault. By inserting a **Throw** step in a synchronous orchestration workflow, you can do the following:

1. Notify a user that the synchronous orchestration workflow encountered a problem and cannot continue.
2. Together with a fault handler, notify a user that the invocation of a Web service within this orchestration workflow returned a SOAP fault.
3. Immediately stop the execution of the scope that encloses the step.
4. Log the contents of the fault that is thrown by the **Throw** step and display it in the Log Viewer.

If a Web service that is invoked within a synchronous orchestration workflow receives a fault as a reply, the user of your process app does not see the fault message, unless you provided a fault handler to catch the fault. Instead, he or she will see the following generic error message: "An error occurred during the execution of the synchronous orchestration workflow." If you want to display the fault message to the user, enclose the Web service invocation within a scope, catch the fault that is thrown by the Web service, and throw a ServiceFlowFault using a **Throw** step. See Using the Throw Step [page 120] for more information.

If a Web service is invoked within an asynchronous orchestration workflow, you cannot notify a user, as previously described in steps 1 and 2, because asynchronous orchestration workflows do not return anything to the caller, which is the SBM Application Engine. However, an asynchronous orchestration workflow can accomplish the tasks described in items 3 and 4.

**Note:** Prior to adding a **Throw** step in an asynchronous orchestration, you must define an Application Link in the corresponding Event Map for the orchestration. Otherwise, the Throw Step will contain an empty "Fault Name" and "Fault Message" and the fault cannot be used in the Scope's Fault Handler. This differs for Throw steps in synchronous orchestrations because the event definition is already created for the orchestration.

#### How to Use the Throw Step

The procedure for using a **Throw** step is described in Using the Throw Step [page 120].

If a **Throw** step is not enclosed within a scope with a fault handler, the whole workflow is stopped after that step is executed. In addition, the content of its FaultString data element is displayed to users if the orchestration workflow is synchronous.

If a **Throw** step is enclosed within a scope that has a fault handler, the fault handler can catch the ServiceFlowFault that is thrown by the **Throw** step. To do this, you can use either a **Catch** branch for a named fault or the **CatchAll** branch. After the ServiceFlowFault is caught, you could use another **Throw** step to throw a new ServiceFlowFault to the SBM Application Engine. The orchestration workflow that you create in Using the Throw Step [page 120] demonstrates how to do this.

#### Summary

1. The only fault that can be thrown by a **Throw** step is ServiceFlowFault.
2. The content of the FaultString data element of the ServiceFlowFault is shown to users if the fault is thrown within a synchronous orchestration workflow.
3. The content of the ServiceFlowFault is logged when the fault is thrown and can be used during debugging.
4. After a **Throw** step is executed, the activity in the next enclosing scope is stopped.

### Compensate Step

During the execution of an orchestration workflow, you might need to reverse, or compensate, an activity that already completed. For example, say you design an orchestration workflow that transfers money between two accounts by invoking a few Web services. One Web service call in the orchestration workflow withdraws funds from the first account and records the transaction. Another Web service call checks the status of the second account and returns an "inactive account" fault. A third Web service call pays back the funds to the first account because the second account is inactive.

An inner scope contains the Web service that withdraws funds. This scope has a CompensationHandler that calls the Web service that pays back the funds. An outer scope encloses the inner scope. The outer scope contains the Web service that checks whether the second account is valid. The outer scope also contains a FaultHandler, in which a proper **Catch** branch contains the **Compensate** step. When the "inactive account" fault is caught, the **Compensate** step is executed.

**Tip:** You could also use a **Throw** step to generate a custom error message such as "This account is inactive, so the transfer transaction could not be completed."

**Note:** You can only place a **Compensate** step in the **FaultHandler** or **CompensationHandler** section of a scope.

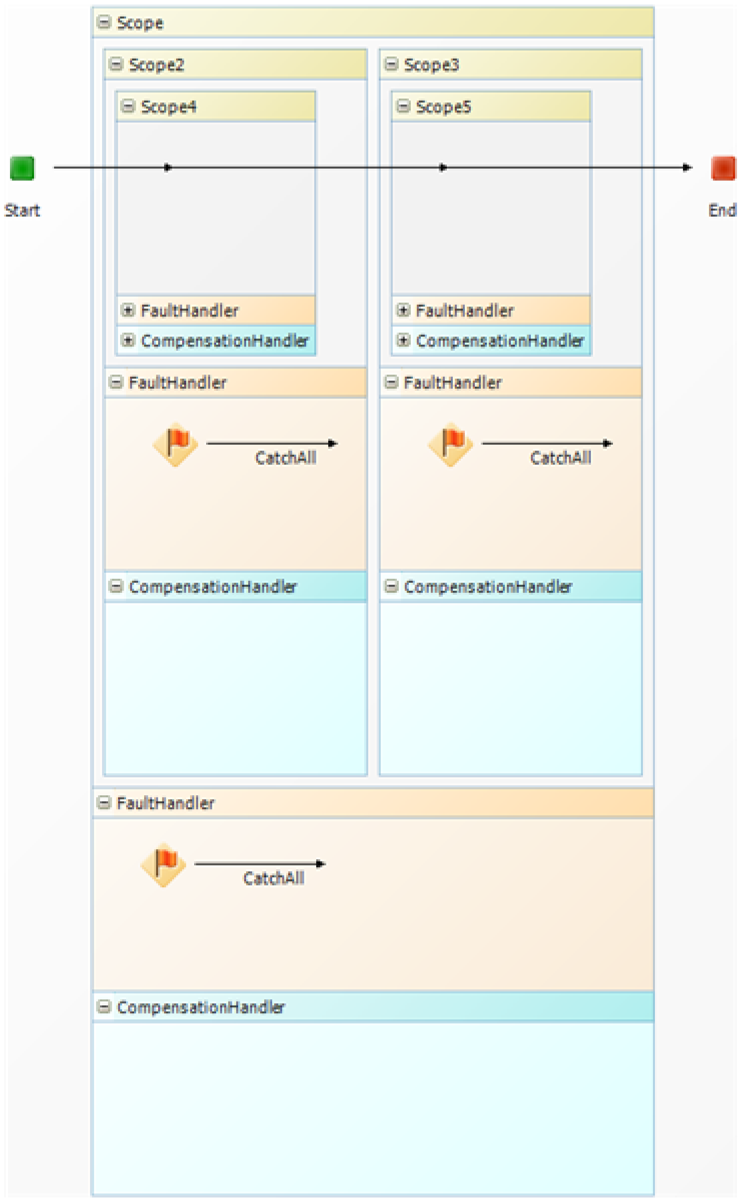
### Configuring a Compensate Step

When you configure the **Compensate** step, you can select the Default Scope or a specific scope.

- If you select the Default Scope, the SBM Orchestration Engine checks all immediately enclosed scopes that completed successfully to determine if they have compensation handlers defined in them. If multiple scopes have completed successfully and all have compensation handlers, these scopes are called one by one in reverse order, that is, the one that completed last is called first. For example, scope "Scope" immediately encloses scopes "Scope2" and "Scope3." If there is a **Compensate** step in the **CompensationHandler** of scope "Scope," the SBM Orchestration Engine runs the compensation handlers in scope "Scope3" and then in "Scope2," if they both completed successfully. To change the order in which the scopes are called, you can place more than one **Compensate** step in a **CompensationHandler** section.

- If you select a specific scope, the BPEL engine only checks that scope to determine if it completed successfully and if it has a compensation handler defined in it. If it completed successfully and has a compensation handler, the compensation handler is called. The compensation handler in this scope can then call other scopes for compensation if it designed to do so.

*Figure 1. Enclosed Scopes*



## Tutorial: Creating a Practice Process App for Fault Handling

In this section, you create a new process app named FaultHandlingProcApp. You will use this process app to practice using the **Scope**, **Throw**, and **Compensate** steps.

You can run the process app at any time after you create an application workflow and deploy the process app. To run the application workflow (project), follow the instructions in the relevant sections of Running the Fault Handling Process App [page 129].

**To create the practice process app:**

1. Start SBM Composer.
2. Click the Composer button, and then click **New**.

The **Create New Process App** dialog box opens.

3. Click **Application Process App**, and then click **Create**.

   The **Configure Process App** dialog box opens.

4. In the **Process app name** and **Application name** boxes, type `FaultHandlingProcApp`, and then click **OK.**

5. In App Explorer, click the **All Items** filter.

6. Under **Tables**, click **FaultHandlingProcessApp**.

7. In the Property Editor, change the **Name** to `FaultHandlingPTable`, and then press the Tab key.

   *PTable* stands for primary table.

8. In the **System Fields** section of the **Table Palette**, drag and drop a **Description** field onto the **FaultHandlingPTable (Primary)** tab.

   The *Description* field is added to the table, in alphabetical order. The *Description* field will also be added to the state and transition forms. Information that is returned by the orchestration workflows you create in this section is displayed in this field and in the *Title* field.

9. In App Explorer, click **FaultHandlingProcessApp**.

10. On the **FaultHandlingProcApp** tab, change the **Logical name** to `FaultHandlingApp`, and then press the Tab key.

11. Perform the following steps to create **GenericFaultAWF**:

    a. In App Explorer, under **Application Workflows**, click **FaultHandlingProcApp**.

    b. On the **General** tab of the Property Editor, change the **Name** to `GenericFaultAWF`, and then press the Tab key.

    c. In the application workflow editor, click the **Submit** transition.

    d. On the **General** tab of the Property Editor, change the **Name** to `VerifyUser`, and then press the Tab key.

    e. Click the **New** state.

    f. On the **General** tab of the Property Editor, change the **Name** to `IsUserValid`, and then press the Tab key.

12. Perform the following steps to create **NamedFaultAWF**:

    a. In App Explorer, right-click **Application Workflows**, and then click **Add New Workflow**.

       Under **Application Workflows**, **FaultHandlingApp Workflow** should be selected.

    b. On the **General** tab of the Property Editor, change the **Name** to `NamedFaultAWF`, and then press the Tab key.

c. In the application workflow editor, click the **Submit** transition.

d. On the **General** tab of the Property Editor, change the **Name** to `GetTickerSymbol`, and then press the Tab key.

e. Click the **New** state.

f. On the **General** tab of the Property Editor, change the **Name** to `TickerSymbol`, and then press the Tab key.

g. In the **States** section of the **Workflow Palette**, drag **Active** onto the application workflow editor and drop it to the right of the **TickerSymbol** state.

h. Change the **Name** to `BuyRating`, and then press the Tab key.

i. In the **Transitions** section of the **Workflow Palette**, drag **Regular** onto the **TickerSymbol** state, release the mouse button, and then click **BuyRating**.

j. Change the name of the **Transition** to `GetBuyRating`, and then press the Tab key.

13. Perform the following steps to create **ThrowAWF**:

    a. In App Explorer, right-click **Application Workflows**, and then select **Add New Workflow**.

       **FaultHandlingAppWorkflow** appears under **NamedFaultAWF**.

    b. Click **FaultHandlingAppWorkflow**.

    c. On the **General** tab of the Property Editor, change the **Name** to `ThrowAWF`, and then press the Tab key.

    d. In the **States** section of the **Workflow Palette**, drag **Active** onto the orchestration workflow editor, and drop it to the right of the **New** state.

    e. On the **General** tab of the Property Editor, change **Name** to `TickerSymbol`.

    f. In the **Transitions** section of the **Workflow Palette**, drag **Regular** onto the **New** state, release the mouse button, and then click the **TickerSymbol** state.

    g. On the **General** tab of the Property Editor, change the **Name** to `GetTickerSymbol`.

14. Perform the following steps to create **CompensateAWF**:

    a. In App Explorer, right-click **Application Workflows**, and then select **Add New Workflow**.

       **FaultHandlingAppWorkflow** appears under **ThrowAWF**.

    b. Click **FaultHandlingAppWorkflow**.

    c. On the **General** tab of the Property Editor, change the **Name** to `CompensateAWF`, and then press the Tab key.

    d. In the **States** section of the **Workflow Palette**, drag **Active** onto the orchestration workflow editor, and drop it to the right of the **New** state.

    e. On the **General** tab of the Property Editor, change **Name** to `TickerSymbol`.

   f. In the **Transitions** section of the **Workflow Palette**, drag **Regular** onto the **New** state, release the mouse button, and then click the **TickerSymbol** state.

   g. On the **General** tab of the Property Editor, change the **Name** to `DemoCompensate`.

15. In App Explorer, right-click **FaultHandlingProcessApp**, point to **Add New**, and then select **Orchestration**.

    The **New Orchestration** dialog box opens.

16. In the **Name** box, type `FaultHandlingOrch`, and then click **OK**.

17. Perform the following steps to import the SerenaSampleTickerService:

    a. In App Explorer, under **FaultHandlingOrch**, right-click **Web Services**, and then select **Add New Web Service**.

       The **Web Service Configuration** dialog box opens.

    b. In the **WSDL** box, enter the following URL: `http://`*`serverName`*`/Ticker/services/SerenaSampleTickerService?wsdl`

       **Note:** In the SBM On-Demand environment, *serverName* is the URL that you use to access Application Repository. For example, the server name for Acme Company is `http://acme.admin.serenaprocessapps.com`. In the on-premise environment, *serverName* is the name of the server (or local machine) that is running the Tomcat server. The *serverName* for most local machines is `localhost:8085`.

    c. Click **OK**.

       **SerenaSampleTickerService** appears in App Explorer under **Web Services**.

18. Save the process app:

    a. On the Quick Access Toolbar, click the **Save locally** button.

       A message box opens reminding you that the design elements have been saved to the Local Cache only.

    b. Click **OK**.

## Using the Scope Step

The **Scope** step includes a **FaultHandler** section and a **CompensationHandler** section. The **FaultHandler** is for handling Web service (SOAP) faults. By default, it has a **CatchAll** branch, which also catches faults generated by the BPEL engine. You can also add other **Catch** branches. Each **Catch** branch can handle a specific named Web service fault.

You can optionally add a **Throw** step to the **FaultHandler** section and a **Compensation** step to the **CompensationHandler** section. The **Throw** step is explained in Using the Throw Step [page 120], and the **CompensationHandler** is covered in Using the Compensate Step [page 124].

### Procedure Summary

1. In App Explorer, select an orchestration workflow to display it in the orchestration workflow editor.

2. In the **New Items** section of the **Step Palette**, drag a **Scope** step onto the orchestration workflow editor, and drop it onto the line between the **Start** and **End** steps.

   You can change the name of the **Scope** step in the Property Editor, and you can also add a description.

3. In the **Step Palette**, drag a **Service** step associated with a Web service onto the orchestration workflow editor, and drop it into the **Scope** section.

4. If the Web service returns named faults, you can perform the following steps to handle them:

   a. Expand the **FaultHandler** section.

   b. Add and configure **Catch** branches one at a time, or automatically add and configure all **Catch** branches for a Web service operation.

      - To add **Catch** branches one at a time:

        1. Right-click the **Fault Handler** step, and then select **Insert New Catch**.

           A new **Catch** branch appears above the **CatchAll** branch.

        2. Select the new **Catch** branch.

        3. On the **General** tab of the Property Editor, specify a Web service fault by selecting it from the **Fault name** list. You can also change the name of the **Catch** branch and provide a description.

           **Note:** If there is an associated fault message, it is displayed in the **Fault message** box as read-only. If there is no message, the **Fault message** box is empty. See Rules for Configuring the Catch Branch [page 119] for more information.

        4. In the **Step Palette**, you can drag any steps that you want to use to handle the fault into the **Scope** section and drop them onto the **Catch** branch. Then configure the steps as required.

        5. Repeat these steps to add and configure other **Catch** branches.

      - To automatically add all **Catch** branches associated with a Web service operation:

        1. In the top section of the **Scope** step, right-click the **Service** step, and then select **Add Catch Branches to Scope**.

           **Catch** branches are automatically added to the **FaultHandler** section.

        2. In the **Step Palette**, you can drag any steps that you want to use to handle the faults into the **Scope** section and drop them onto a **Catch** branch. Then configure the steps as required.

   c. To handle faults that are not handled by the **Catch** branches, you can drag any steps that you want to use to handle these faults into the **Scope** section and drop them onto the **CatchAll** branch. Then configure the steps as required.

      You cannot specify a fault name and a fault message for the **CatchAll** branch, because it is invoked if none of the other branches can handle the fault.

5. If the Web service does not return named faults, perform the following steps to handle the faults returned by the Web service:
   a. Expand the **FaultHandler** section.
   b. In the **Step Palette**, you can drag any steps that you want use to handle the faults into the **Scope** section and drop them onto the **CatchAll** branch. Then configure the steps as required.

**Note:** To hide the sections of the **Scope** step as you work on another part of the orchestration workflow, click the minus sign to the left of the **Scope** step.

## Tutorial: Creating An Empty Synchronous Orchestration Workflow to Handle Generic Web Service Faults

In this exercise, you add an empty synchronous orchestration workflow to the FaultHandlingOrch in the FaultHandlingProcApp.

**To create an empty orchestration workflow that handles generic Web service faults:**

1. In App Explorer, under **Application Workflows**, click **GenericFaultAWF.**
2. In the application workflow editor, right-click the **VerifyUser** transition, and then select **Show Actions**.
3. On the **Actions** tab of the Property Editor, click **New.**

   The **Action Wizard** opens and asks, "Which type of action do you want to execute?"

   Under **Step 1**, **Orchestration Workflow** should be selected.
4. Under **Step 2**, click the **and continue executing (asynchronous)** link, and then select **and wait for reply (synchronous)**.
5. Click **Next**.

   The **Action Wizard** asks, "When do you want to call the orchestration workflow?"
6. Under **Step 1**, select **After**.

   **Step 2** should read **Invoke an orchestration workflow and wait for reply (synchronous), after this transition occurs**.
7. Click **Next**.

   The **Action Wizard** asks, "Which orchestration workflow do you want to call?"
8. Under **Step 1**, select **(Add new workflow...)**.

   The **Event with Reply** dialog box opens.
9. In the **Name** box, type `GenericFaultWFWR`.
10. In the **Workflow** box, type `GenericFaultOWF`.
11. In the **Fields used by event** list, select the **Title** check box.

12. In the **Fields returned by event** list, select the **Description** and **Title** check boxes.

13. Click **OK**.

    Under **Step 1**, **GenericFaultWFWR** is selected on the menu and **EventNoticeWithReply** appears in the list. **Step 2** now reads **Invoke an orchestration workflow and wait for reply (synchrononous), after this transition occurs | Call GenericFaultWFWR.EventNoticeWithReply.**

14. Click **Finish**.

    **GenericFaultWFWR** appears under **Orchestration Links**, and **GenericFaultOWF** appears under **Orchestration Workflows** in App Explorer.

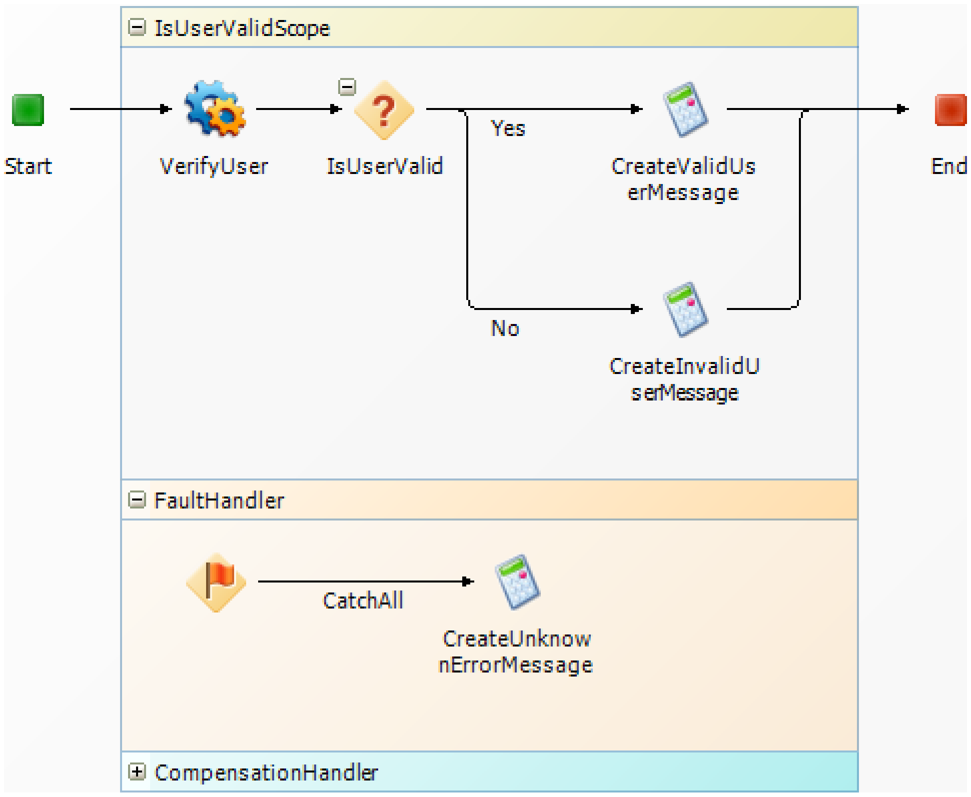
## Tutorial: Practicing With the Scope Step to Handle Generic Web Service Faults

In this exercise, you use a **Scope** step in the GenericFaultOWF orchestration workflow.

**Note:** Even though the SBM Application Engine actually returns a named fault, for demonstration purposes, this exercise assumes that is does not.

After you complete the steps in this exercise, your orchestration workflow should look like the one in the following figure:

*Figure 1. GenericFaultOWF*



Later, when you run the GenericFaultApp Project, you will alter this orchestration workflow so it returns an error message in the **Description** field.

**To use the Scope step in an orchestration workflow to handle generic Web service faults:**

1. In App Explorer, under **Orchestration Workflows**, click **GenericFaultOWF**.
2. Create a working data element of type String to hold the message that is passed to the **Description** field as follows:
   a. Click a blank area of the orchestration workflow editor.
   b. On the **Data Mapping** tab of the Property Editor, under **GenericFaultOWF**, right-click **Working Data**, select **Add New**, and then select **String**.
   c. Change the name of the new data element (**String**) to `Message`.
3. In the **New Items** section of the **Step Palette**, drag a **Scope** step onto the orchestration workflow editor, and drop it between the **Start** and **End** steps.
4. On the **General** tab of the Property Editor, change the **Name** to `IsUserValidScope`, and then press the Tab key.
5. In the **Configured Items** section of the **Step Palette**, drag an **sbmappservices72** Service step onto the orchestration workflow editor, and drop it onto the line inside the top section of the **Scope** step.
6. On the **General** tab of the Property Editor, change the **Name** to `VerifyUser`.
7. From the **Operation** menu, select **IsUserValid**.
8. On the **Data Mapping** tab, locate the **loginId** data element, select the corresponding cell in the **Source elements** column, and then click the down arrow.
9. In the **Select a Source** popup that opens, under **GenericFaultOWF**, **Inputs**, **EventNoticeWithReply**, **Extension**; select **Title**; and then click **OK**.
10. In the **Step Palette**, drag a **Decision** step onto the orchestration workflow editor, and drop it onto the line inside the top section of the **IsUserValidScope** step, to the right of the **VerifyUser** step.

    In steps 11 through 24, you configure this step to decide between two possible outcomes: the user is valid or the user is not valid.
11. In the Property Editor, change the **Name** to `IsUserValid`, and then press the Tab key.
12. Right-click the **IsUserValid** step, and then select **Insert New Branch**.
13. On the **General** tab of the Property Editor, change the **Name** to `Yes`.
14. On the **Options** tab, in the **Rule** section, enter the following expression using the expression editor: `VerifyUser.IsUserValidResponse.return`.

    See About the Expression Editor [page 32].
15. In the **Step Palette**, drag a **Calculate** step onto the orchestration workflow editor, and drop it onto the **Yes** branch.
16. On the **General** tab of the Property Editor, change the **Name** to `CreateValidUserMessage`.

17. On the **Options** tab, in the **Target** section, enter the following using the expression editor: `Message`.
18. In the **Expression** section, type: `"This user is valid."`

    Be sure to include the quotation marks.
19. Select the **Otherwise** branch.
20. On the **General** tab of the Property Editor, change the **Name** to `No`, and then press the Tab key.
21. In the **Step Palette**, drag a **Calculate** step onto the orchestration workflow editor, and drop it onto the **No** branch.
22. On the **General** tab of the Property Editor, change the **Name** to `CreateInvalidUserMessage`.
23. On the **Options** tab, in the **Target** section, enter the following using the expression editor: `Message`
24. In the **Expression** section, type `"This user is not valid."`

    Be sure to include the quotation marks.
25. Select the **End** step.
26. On the **Data Mapping** tab, under **Extension**, locate the **Description** data element, select the corresponding **Source elements** column, and then click the down arrow.
27. In the **Select a Source** popup that opens, under **GenericFaultOWF**, **WorkingData**; select **Message**; and then click **OK**.
28. Expand the **FaultHandler** section of the **IsUserValidScope** step.
29. In the **Step Palette**, drag a **Calculate** step into the **FaultHandler** section and drop it onto the **CatchAll** branch.
30. On the **General** tab of the Property Editor, change the **Name** to `CreateUnknownErrorMessage`.
31. On the **Options** tab, in the **Target** section, enter the following using the expression editor: `Message`.
32. In the **Expression** section, type `"An unknown error occurred at GenericFaultAWF_GenericFaultOWF_VerifyUser."`

    Be sure to include the quotation marks.
33. On the Quick Access Toolbar, click the **Validate** button.

    The following two warning messages appear in the Validation Results:

    - **The required DefaultElement 'GenericFaultOWF\Message' is not mapped or defaulted in 'GenericFaultOWF'**

This message warns you that you did not provide a value for the **Message** working data element. You can ignore the message, or you can set the default value to 0 (zero) to prevent the message from appearing.

- **Compensation handler is empty.**

  You can ignore this message, because a compensation handler is not required for this orchestration workflow.

34. Publish and deploy the **FaultHandlingProcApp.**

    See Step 6: Publish the Process App [page 192] and Step 7: Deploy the Process App [page 192] for instructions.

35. Turn on debug logging as follows:

    a. On the **Home** tab of the Ribbon, in the **Common Views** group, select the **Common Log Viewer** check box.

    b. On the **Overview** tab of the Common Log Viewer, right-click **FaultHandlingApp**, and then select **Debug Logging**.

    c. Right-click **FaultHandlingOrch**, and then select **Debug Logging**.

## Tutorial: Creating An Empty Synchronous Orchestration Workflow for the Scope Step to Handle Named Faults

In this exercise, you add an empty synchronous orchestration workflow to the FaultHandlingOrch in the FaultHandlingProcApp.

**To create an empty orchestration workflow that handles named Web service faults:**

1. In App Explorer, under **Application Workflows**, click **NamedFaultAWF.**

2. In the application workflow editor, right-click the **GetTickerSymbol** transition, and then select **Show Actions**.

3. On the **Actions** tab of the Property Editor, click **New.**

   The **Action Wizard** opens and asks, "Which type of action do you want to execute?"

   Under **Step 1**, **Orchestration Workflow** should be selected.

4. Under **Step 2**, click the **and continue executing (asynchronous)** link, and then select **and wait for reply (synchronous)**.

5. Click **Next**.

   The **Action Wizard** asks, "When do you want to call the orchestration workflow?"

6. Under **Step 1**, select **After**.

   **Step 2** should read **Invoke an orchestration workflow and wait for reply (synchronous), after this transition occurs**.

7. Click **Next**.

   The **Action Wizard** asks, "Which orchestration workflow do you want to call?"

8. Under **Step 1**, select **(Add new workflow...)**.

   The **Event With Reply** dialog box opens.

9. In the **Name** box, type `NamedFaultWFWR1`.

10. In the **Workflow** box, type `NamedFaultOWF1`.

11. In the **Fields used by event** list, select the **Title** check box.

12. In the **Fields returned by event** list, select the **Description** and **Title** check boxes.

13. Click **OK**.

    Under **Step 1**, **NamedFaultWFWR1** is selected on the menu and **EventNoticeWithReply** appears in the list. **Step 2** now reads **Invoke an orchestration workflow and wait for reply (synchronous), after this transition occurs, Call NamedFaultWFWR1.**

14. Click **Finish**.

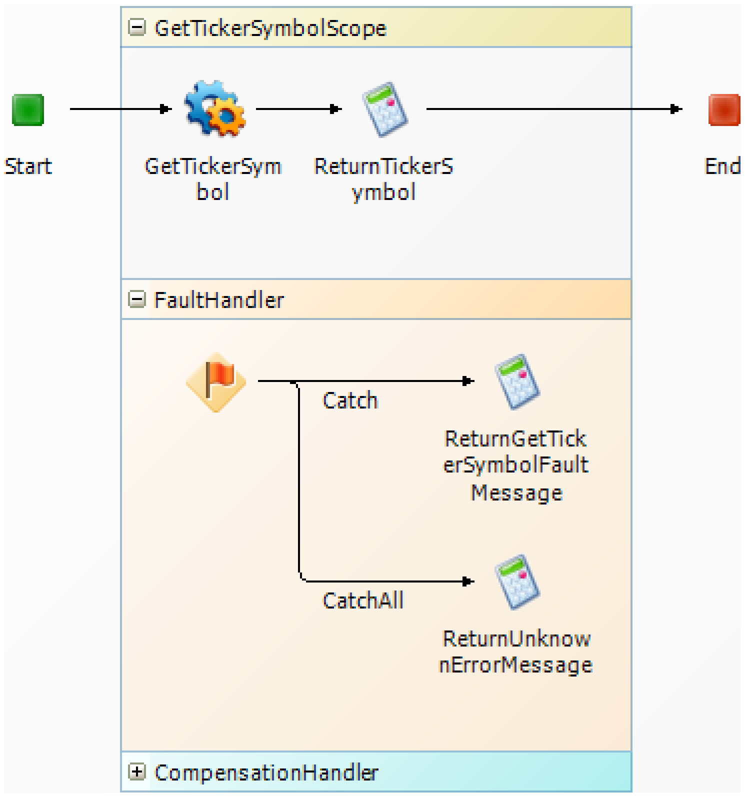
    **NamedFaultWFWR1** appears below **GenericFaultWFWR** under **Orchestration Links**, and **NamedFaultOWF1** appears below **GenericFaultOWF** under **Orchestration Workflows** in App Explorer.

## Tutorial: Practicing With the Scope Step to Handle Named Web Service Faults

In this exercise, you use a **Scope** step in the **NamedFaultOWF1** orchestration workflow.

After you complete the steps in this exercise, your orchestration workflow should look like the one in the following figure:

*Figure 1. NamedFaultOWF1*



**To use the Scope step in an orchestration workflow that handles named Web service faults:**

1. Under **Orchestration Workflows**, click **NamedFaultOWF1**.
2. Create a working data element of type String to hold the message that is passed to the **Description** field as follows:
   a. Click a blank area of the orchestration workflow editor.
   b. On the **Data Mapping** tab of the Property Editor, under **NamedFaultOWF1**, right-click **Working Data**, select **Add New**, and then select **String**.
   c. Change the name of the new data element (**String**) to `Message`.
3. In the **New Items** section of the **Step Palette**, drag a **Scope** step onto the orchestration workflow editor, and drop it between the **Start** and **End** steps.
4. Change the name of the **Scope** step to `GetTickerSymbolScope`, and then press the Tab key.
5. In the **Configured Items** section of the **Step Palette**, drag a **SerenaSampleTickerService** Service step onto the orchestration workflow editor, and drop it onto the line inside the top section of the **GetTickerSymbolScope** step.

6. On the **General** tab of the Property Editor, change the **Name** to `GetTickerSymbol`.
7. From the **Operation** menu, select **GetTickerSymbol**.
8. On the **Data Mapping** tab, locate the **company** data element, select the corresponding cell in the **Source elements** column, and then click the down arrow.
9. In the **Select a Source** popup that opens, under **NamedFaultOWF1**, **Inputs**, **EventNoticeWithReply**, **Extension**; select **Title**; and then click **OK**.
10. In the **New Items** section of the **Step Palette**, drag a **Calculate** step onto the orchestration workflow editor, and drop it onto the line inside the top section of the **GetTickerSymbolScope** step, to the right of the **GetTickerSymbol** step.
11. On the **General** tab of the Property Editor, change the **Name** to `ReturnTickerSymbol`.
12. On the **Options** tab, in the **Target** section, enter the following using the expression editor: `EventNoticeWithReply\Extension\Title`.

    See About the Expression Editor [page 32].
13. In the **Expression** section, enter the following expression using the expression editor: `GetTickerSymbol\GetTickerSymbolResponse\GetTickerSymbolResult`.
14. Select the **End** step.
15. On the **Data Mapping** tab, under **Extension**, locate the **Title** data element, select the corresponding cell in the **Source elements** column, and then click the down arrow.
16. In the **Select a source** popup that opens, under **NamedFaultOWF1**, **Inputs**, **EventNoticeWithReply**, **Extension**; select **Title**; and then click **OK**.
17. On the **Data Mapping** tab, under **Extension**, locate the **Message** data element, select the corresponding cell in the **Source elements** column, and then click the down arrow.
18. In the **Select a source** popup that opens, under **NamedFaultOWF1**, **WorkingData**; select **Message**; and then click **OK**.
19. Expand the **FaultHandler** section of the **GetTickerSymbolScope** step.
20. Right-click the **Throw** step, and then select **Insert New Catch**.
21. On the **General** tab of the **Catch** branch, select **SerenaSampleTickerService-GetTickerSymbolFault** on the **Fault Name** menu.

    **SerenaSampleTickerService-GetTickerSymbolFault** is automatically inserted in the **Fault message** box and is read-only.
22. In the **New Items** section of the **Step Palette**, drag a **Calculate** step into the **FaultHandler** section, and drop it onto the **Catch** branch.
23. On the **General** tab of the Property Editor, change the **Name** to `ReturnGetTickerSymbolFault`.

24. On the **Options** tab, in the **Target** section, enter the following using the expression editor: `Message`.

25. In the **Expression** section, enter the following expression using the expression editor: `Catch\GetTickerSymbolFault\detail`.

    **Note:** For this step, you must first type `Catch` followed by a period. Then you can use the expression editor to complete the expression. Also, any time you rename a **Catch** branch, you must use the new name in the expression.

26. In the **New Items** section of the **Step Palette**, drag a **Calculate** step into the **FaultHandler** section of the **GetTickerSymbolScope** step, and drop it onto the **CatchAll** branch.

27. On the **General** tab of the Property Editor, change the **Name** to `CreateUnknownErrorMessage`.

28. On the **Options** tab, in the **Target** section, enter the following using the expression editor: `Message`.

29. In the **Expression** section, type `"An unknown error occurred at NamedFaultAWF_NamedFaultOWF1_GetTickerSymbol."`

    Be sure to include the quotation marks.

30. On the Quick Access Toolbar, click the **Validate** button.

    The following two warning messages appear in the Validation Results:

    - **The required DefaultElement 'NamedFaultOWF1\Message' is not mapped or defaulted in 'NamedFaultOWF1'**

      This message warns you that you did not provide a value for the **Message** working data element. You can ignore the message, or you can set the default value to 0 (zero) to prevent the message from appearing.

    - **Compensation handler is empty.**

      You can ignore this message, because a compensation handler is not required for this orchestration workflow.

31. Publish and deploy the **FaultHandlingProcApp.**

    See Step 6: Publish the Process App [page 192] and Step 7: Deploy the Process App [page 192] for instructions.

## Tutorial: Creating an Empty Synchronous Orchestration Workflow for Automatically Adding Catch Branches for Named Faults

In this exercise, you add an empty synchronous orchestration workflow to the **FaultHandlingOrch** in the **FaultHandlingProcApp**.

**To create an empty orchestration workflow for automatically adding Catch branches for named faults:**

1. In App Explorer, under **Application Workflows**, click **NamedFaultAWF.**

2. In the application workflow editor, right-click the **GetBuyRating** transition, and then select **Show Actions** on the menu.
3. On the **Actions** tab of the Property Editor, click **New.**

   The **Action Wizard** opens and asks, "Which type of action do you want to execute?"

   Under **Step 1**, **Orchestration Workflow** should be selected.
4. Under **Step 2**, click the **and continue executing (asynchronous)** link, and then select **and wait for reply (synchronous)**.
5. Click **Next**.

   The **Action Wizard** asks, "When do you want to call the orchestration workflow?"
6. Under **Step 1**, select **After**.

   **Step 2** should read **Invoke an orchestration workflow and wait for reply (synchronous), after this transition occurs**.
7. Click **Next**.

   The **Action Wizard** asks, "Which orchestration workflow do you want to call?"
8. Under **Step 1**, select **(Add new workflow...)**. The **Event With Reply** dialog box opens.
9. In the **Name** box, type `NamedFaultWFWR2`.
10. In the **Workflow** box, type `NamedFaultOWF2`.
11. In the **Fields used by event** list, select the **Description** and **Title** check boxes.
12. In the **Fields returned by event** list, select the **Description** and **Title** check boxes.
13. Click **OK**.

    Under **Step 1**, **NamedFaultWFWR2** is selected on the menu and **EventNoticeWithReply** appears in the list. **Step 2** now reads **Invoke an orchestration workflow and wait for reply (synchronous), after this transition occurs, Call NamedFaultWFWR2.EventNoticeWithReply.**
14. Click **Finish**.

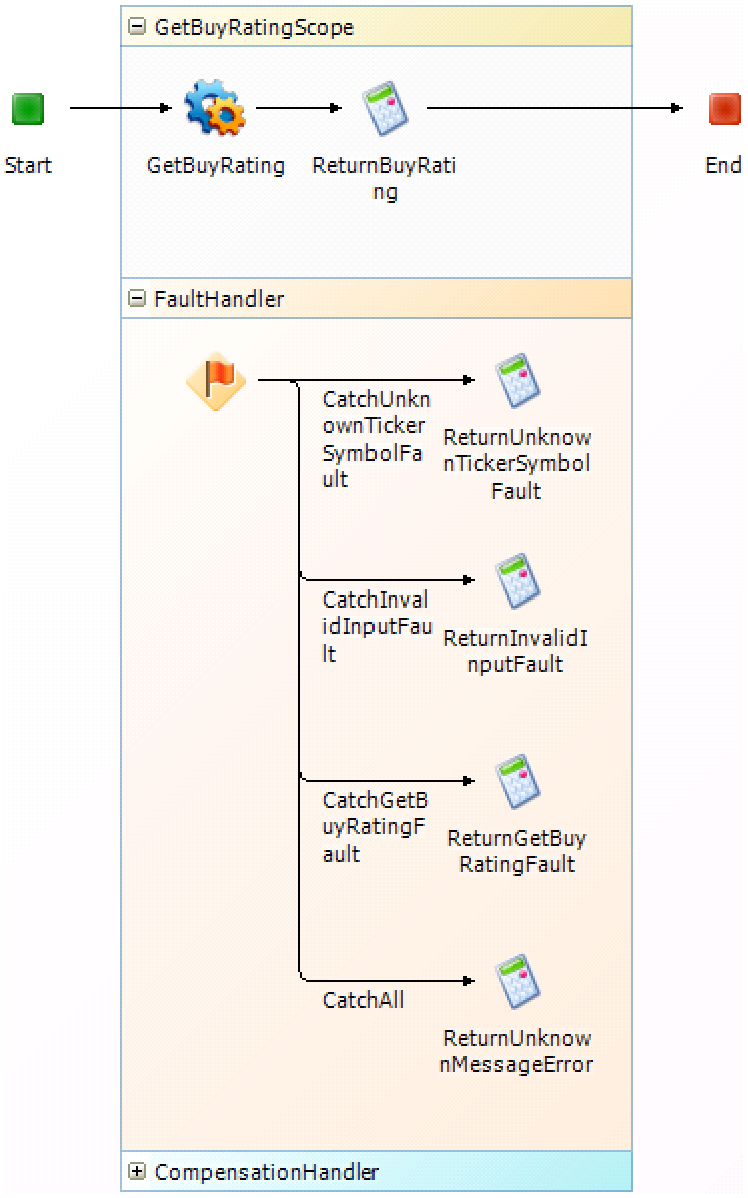
    **NamedFaultWFWR2** appears below **NamedFaultWFWR1** under **Orchestration Links**, and **NamedFaultOWF2** appears below **NamedFaultOWF1** under **Orchestration Workflows** in App Explorer.

## Tutorial: Practicing Automatically Adding Catch Branches for Named Faults

In this exercise, you use a **Scope** step in the **NamedFaultOWF2** orchestration workflow and you add **Catch** branches to its **FaultHandler** section automatically.

After you complete the steps in this exercise, your orchestration workflow should look like the one in the following figure:

*Figure 1. NamedFaultOWF2*



**To automatically add Catch branches for named Web service faults:**

1. Under **Orchestration Workflows**, click **NamedFaultOWF2**.
2. Create a working data element of type String to hold the message that is passed to the **Description** field as follows:
   a. Click a blank area of the orchestration workflow editor.
   b. On the **Data Mapping** tab of the Property Editor, under **NamedFaultOWF2**, right-click **Working Data**, select **Add New**, and then select **String**.

c. Change the name of the new data element (**String**) to `Message`.

3. In the **New Items** section of the **Step Palette**, drag a **Scope** step onto the orchestration workflow editor, and drop it between the **Start** and **End** steps.
4. Change the name of the **Scope** step to `GetBuyRatingScope`, and then press the Tab key.
5. In the **Configured Items** section of the **Step Palette**, drag a **SerenaSampleTickerService** Service step onto the orchestration workflow editor, and drop it onto the line inside the top section of the **GetBuyRatingScope** step.
6. On the **General** tab of the Property Editor, change the **Name** to `GetBuyRating`.
7. From the **Operation** menu, select **GetBuyRating**.
8. On the **Data Mapping** tab, locate the **symbol** data element, select the corresponding cell in the **Source elements** column, and then click the down arrow.
9. In the **Select a Source** popup that opens, under **NamedFaultOWF2**, **Inputs**, **EventNoticeWithReply**, **Extension**; select **Title**; and then click **OK**.
10. In the **New Items** section of the **Step Palette**, drag a **Calculate** step onto the orchestration workflow editor, and drop it onto the line inside the top section of the **GetBuyRatingScope** step, to the right of the **GetBuyRating** step.
11. On the **General** tab of the Property Editor, change the **Name** to `ReturnBuyRating`.
12. On the **Options** tab, in the **Target** section, enter the following using the expression editor: `Message`.

    See About the Expression Editor [page 32].
13. In the **Expression** section, enter the following expression using the expression editor: `GetBuyRating.GetBuyRatingResponse.GetBuyRatingResult`.
14. Select the **End** step.
15. On the **Data Mapping** tab, under **Extension**, locate the **Description** data element, select the corresponding cell in the **Source elements** column, and then click the down arrow.
16. In the **Select a source** popup that opens, under **NamedFaultOWF2**, **WorkingData**; select **Message**; and then click **OK**.
17. Expand the **FaultHandler** section of the **GetBuyRatingScope** step.
18. In the top section of the **GetBuyRatingScope** step, right-click the **GetBuyRating** step, and then select **Add Catch Branches to GetBuyRatingScope**.

SBM Composer scans the **SerenaSampleTickerService** WSDL for any named faults defined for the **GetBuyRating** operation and adds the following three **Catch** branches to the **FaultHandler** section: **CatchUnknownTickerSymbolFault**, **CatchInvalidInputFault**, and **CatchGetBuyRatingFault**.

**Note:** If the Web service operation does not return any named faults, the **Add Catch Branches to Scope** option is not available.

19. Select each **Catch** branch, and note on the **General** tab of the Property Editor that the **Fault name** and **Fault message** are automatically configured.

20. In the **New Items** section of the **Step Palette**, drag a **Calculate** step into the **FaultHandler** section, and drop it onto the **CatchUnknownTickerSymbolFault** branch.

21. On the **General** tab of the Property Editor, change the **Name** to `ReturnUnknownTickerSymbolFault`.

22. On the **Options** tab, in the **Target** section, enter the following using the expression editor: `Message`

23. In the **Expression** section, enter the following expression using the expression editor: `CatchUnknownTickerSymbolFault\UnknownTickerSymbolFault\detail`

    You can use `errorCode` rather than `detail` if you specified a decision based on the error code.

    **Note:** For this step, you must first type `CatchUnknownTickerSymbolFault` followed by a period. Then you can use the expression editor to complete the expression. Also, any time you rename a **Catch** branch, you must use the new name in the expression.

24. In the **New Items** section of the **Step Palette**, drag a **Calculate** step into the **FaultHandler** section, and drop it onto the **CatchInvalidInputFault** branch.

25. On the **General** tab of the Property Editor, change the **Name** to `ReturnInvalidInputFault`.

26. On the **Options** tab, in the **Target** section, enter the following using the expression editor: `Message`.

27. In the **Expression** section, enter the following expression using the expression editor: `CatchInvalidInputFault\InvalidInputFault`.

    **Note:** For this step, you must first type `CatchInvalidInputFault` followed by a period. Then you can use the expression editor to complete the expression.

28. In the **New Items** section of the **Step Palette**, drag a **Calculate** step into the **FaultHandler** section, and drop it onto the **CatchGetBuyRatingFault** branch.

29. On the **General** tab of the Property Editor, change the **Name** to `ReturnGetBuyRatingFault`.

30. On the **Options** tab, in the **Target** section, enter the following using the expression editor: `Message`.

31. In the **Expression** section, enter the following expression using the expression editor: `CatchGetBuyRatingFault\GetBuyRatingFault\detail`.

    You can use `errorCode` rather than `detail` if you created a system that correlates the message detail with the error code.

    **Note:** For this step, you must first type `CatchGetBuyRatingFault` followed by a period. Then you can use the expression editor to complete the expression. Also, any time you rename a **Catch** branch, you must use the new name in the expression.

32. In the **New Items** section of the **Step Palette**, drag a **Calculate** step into the **FaultHandler** section of the **GetBuyRatingScope** step, and drop it onto the **CatchAll** branch.

33. On the **General** tab of the **Property Editor**, change the **Name** to `CreateUnknownErrorMessage`.

34. On the **Options** tab, in the **Target** section, enter the following using the expression editor: `Message`.

35. In the **Expression** section, type: `"An unknown error occurred at NamedFaultAWF_NamedFaultOWF2_GetBuyRating"`

    Be sure to include the quotation marks.

36. On the Quick Access Toolbar, click the **Validate** button.

    The following warning messages appear in the Validation Results:

    - **The required DefaultElement 'NamedFaultOWF2\Message' is not mapped or defaulted in 'NamedFaultOWF2'**

      This message warns you that you did not provide a value for the **Message** working data element. You can ignore the message, or you can set the default value to 0 (zero) to prevent the message from appearing.

    - **Compensation handler is empty**

      You can ignore this message, because a compensation handler is not required for this orchestration workflow.

    - **The required DataElement 'GetBuyRating\GetBuyRating\type' is not mapped or defaulted in 'NamedFaultOWF2**

      You can ignore this message.

37. Publish and deploy the **FaultHandlingProcApp.**

    See Step 6: Publish the Process App [page 192] and Step 7: Deploy the Process App [page 192] for instructions.

## Rules for Configuring the Catch Branch

When you configure a **Catch** branch, you specify a fault name, a fault message, or both. Before you can determine which item or items to choose, you need to know how the Web service that you are invoking returns faults. This information is usually provided in the Web service documentation.

In SBM Composer, the rules are applied as follows:

- If the Web service returns a fault with no associated message, after you select a **Fault name**, the **Fault message** box is empty.
- If the Web service returns a fault with a name and an associated message, after you select a **Fault name**, the name of the associated message is displayed in the **Fault message** box. The message is read-only.

## Using the Throw Step

You use the **Throw** step to return a fault named ServiceFlowFault in an orchestration workflow.

#### Procedure Summary

1. In App Explorer, select an orchestration workflow to display it in the orchestration workflow editor.
2. In the **New Items** section of the **Step Palette**, drag a **Throw** step onto the orchestration workflow editor, and drop it wherever you want to throw a ServiceFlowFault.
3. On the **General** tab of the Property Editor, you can change the name of the **Throw** step and provide a description.
4. On the **Data Mapping** tab, you can enter an optional fault code in the **Default** column of the **FaultCode** data element.

   You can use the fault code as a short way to represent a long fault string.
5. Still on the **Data Mapping** tab, you can provide a value for the **FaultString** data element. You can map a value in the **Source elements** column, or you can enter a String value, such as a message, in the **Default value** column.

### Tutorial: Creating an Empty Synchronous Orchestration Workflow for the Throw Step

In this exercise, you add an empty synchronous orchestration workflow to the **FaultHandlingOrch** in the **FaultHandlingProcApp**.

#### To create an empty synchronous orchestration workflow that uses the Throw step:

1. In App Explorer, under **Application Workflows**, click **ThrowAWF**.
2. In the application workflow editor, right-click the **GetTickerSymbol** transition, and then select **Show Actions**.
3. On the **Actions** tab of the Property Editor, click **New**.

   The **Action Wizard** opens and asks, "Which type of action do you want to execute?"

   Under **Step 1**, **Orchestration Workflow** should be selected.
4. Under **Step 2**, click the **and continue executing (asynchronous)** link, and then select **and wait for reply (synchronous)**.
5. Click **Next**.

The **Action Wizard** asks, "When do you want to call the orchestration workflow?"

6. Under **Step 1**, select **After**.

   **Step 2** should read **Invoke an orchestration workflow and wait for reply (synchronous), after this transition occurs**.

7. Click **Next**.

   The **Action Wizard** asks, "Which orchestration workflow do you want to call?"

8. Under **Step 1**, select **(Add new workflow...)**.

   The **Event With Reply** dialog box opens.

9. In the **Name** box, type `ThrowWFWR`.

10. In the **Workflow** box, type `ThrowOWF`.

11. In the **Fields used by event** list, select the **Title** check box.

12. In the **Fields used by event** list, select the **Description** and **Title** check boxes.

13. Click **OK**.

    Under **Step 1**, **ThrowWFWR** is selected on the menu and **EventNoticeWithReply** is selected in the list. **Step 2** now reads **Invoke an orchestration workflow and wait for reply (synchronous), after this transition occurs, Call ThrowWFWR**.

14. Click **Finish**.

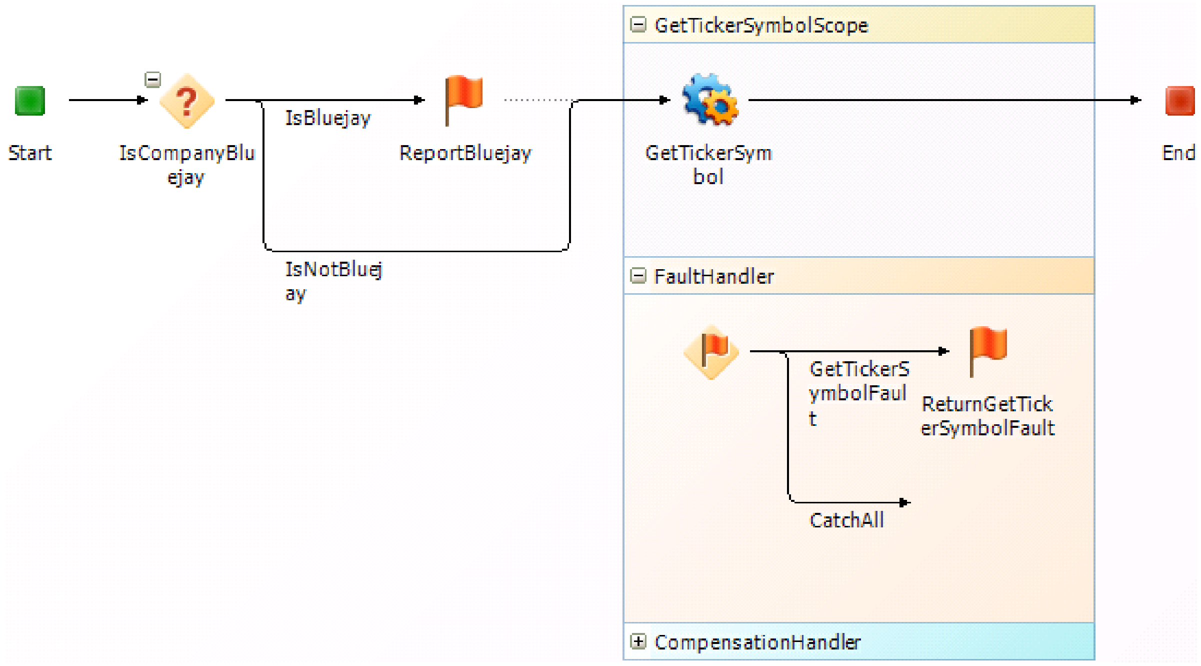
    **ThrowWFWR** appears below **NamedFault2WRWR** under **Orchestration Links**, and **ThrowOWF** appears below **NameFault2OWF** under **Orchestration Workflows** in App Explorer.

## Tutorial: Practicing With the Throw Step

In this exercise, you use a **Scope** step in the **NamedFaultOWF2** orchestration workflow and add **Catch** branches to its **FaultHandler** section automatically.

After you complete the steps in this exercise, your orchestration workflow should look like the one in the following figure:

*Figure 1. ThrowOWF*



**To use the Throw step in an orchestration workflow that throw custom and named faults:**

1. In App Explorer, under **Orchestration Workflows**, click **ThrowOWF**.
2. In the **New Items** section of the **Step Palette**, drag a **Decision** step onto the orchestration workflow editor, and drop it onto the line between the **Start** and **End** steps.

   In steps 3 through 10, you configure this step to decide between two possible outcomes: the company is Bluejay Bird Supply or the company is not Bluejay Bird Supply.
3. On the **General** tab of the Property Editor, change the **Name** to `IsCompanyBluejay`.
4. Right-click the **IsCompanyBluejay** step, and then select **Insert New Branch**.
5. On the **General** tab of the Property Editor, change the **Name** to `IsBluejay`.
6. On the **Options** tab of the Property Editor, enter the following expression in the **Rule** section using the expression editor:
   ```
   EventNoticeWithReply\Extension\Title="Bluejay Bird Supply"
   ```
   See About the Expression Editor [page 32].
7. In the **Step Palette**, drag a **Throw** step onto the orchestration workflow editor and drop it on the **IsBluejay** branch.

This step is used to show an error message to users when the **Title** field contains "Bluejay Bird Supply." It is also used to display an optional error code that is not shown to users, but can be used, for example, for debugging purposes.

8. On the **General** tab of the Property Editor, change the **Name** to `ReportBluejay`, and then press the Tab key.
9. On the **Data Mapping** tab, locate the **FaultCode** data element and enter the following in the corresponding cell of the **Default value** column: `ERROR-BLUEJAY`.
10. Locate the **FaultString** data element, and enter the following in the corresponding **Default value** column: `Bluejay Bird Supply is a credit risk.`
11. In the **New Items** section of the **Step Palette**, drag a **Scope** step onto the orchestration workflow editor, and drop it between the **ReportBluejay** and **End** steps.

    This step is used to enclose the **GetTickerSymbol** Web service operation and to catch any faults that it generates.
12. On the **General** tab of the Property Editor, change the **Name** to `GetTickerSymbolScope`.
13. In the **Configured Items** section of the **Step Palette**, drag a **SerenaSampleTickerService** step into the top section of the **GetTickerSymbolScope** step, and drop it. This step is used to get the ticker symbol for the company name passed from the **Title** field, or to return a fault for all other company names except Bluejay Bird Supply (see step 4).
14. On the **General** tab of the Property Editor, change the **Name** to `GetTickerSymbol`.
15. On the **Operation** menu, select **GetTickerSymbol**.
16. On the **Data Mapping** tab, locate the **company** data element, select the corresponding cell in the **Source elements** column, and then click the down arrow.
17. In the **Select a Source** popup that opens, under **ThrowOWF**, **Inputs**, **EventNoticeWithReply**, **Extension**; select **Title**; and then click **OK**.
18. Expand the **FaultHandler** section.
19. Right-click the **Fault Handler** step, and then select **Insert New Catch**.

    This **Catch** branch handles the SerenaSampleTickerService-GetTickerSymbolFault. All other faults generated by the **GetTickerSymbol** step are handled by the **CatchAll** branch.
20. On the **General** tab of the Property Editor for the **Catch** branch, change the **Name** to `GetTickerSymbolFault`.
21. Select **SerenaSampleTickerService-GetTickerSymbolFault** on the **Fault Name** menu.

    **SerenaSampleTickerService-GetTickerSymbolFault** is automatically inserted in the **Fault message** box and is read-only.

22. In the **Step Palette**, drag a **Throw** step into the **FaultHandler** section and drop it onto the **GetTickerSymbolFault** branch.

    This step is used to return a customized fault message. It is also used to generate an optional error code.

23. On the **General** tab of the Property Editor, change the **Name** to `ReturnGetTickerSymbolFault`.

24. On the **Data Mapping** tab, locate the **FaultCode** data element and enter the following in the corresponding cell in the **Default value** column: `ERROR-SYSTEM`.

25. Locate the **FaultString** data element, select the corresponding cell in the **Source elements** column, and then click the down arrow.

26. In the **Select a Source** popup that opens, under **ThrowOWF**, **GetTickerSymbolFault**, **Outputs**, **GetTickerSymbolFault**; select **detail**; and then click **OK**.

27. Select the **End** step.

    The contents of the **Title** field are passed to the **End** step. This step is used to display a valid ticker symbol in the **Description** field. Although Bluejay Bird Supply is listed in the SerenaSampleTickerService, its symbol is not returned because it was caught by the **ReportBluejay** Throw step.

28. On the **Data Mapping** tab, under **Extension**, locate the **Description** data element, select the corresponding cell in the **Source elements** column, and then click the down arrow.

29. In the **Select a Source** popup that opens, under **GetTickerSymbol**, **Outputs**, **GetTickerSymbolResponse**; select **GetTickerSymbolResult**; and then click **OK**.

30. On the Quick Access Toolbar, click the **Validate** button.

    The following a warning message appears in the Validation Results: **Compensation handler is empty.**

    You can ignore this message, because a compensation handler is not required for this orchestration workflow.

31. Publish and deploy the **FaultHandlingProcApp.**

    See Step 6: Publish the Process App [page 192] and Step 7: Deploy the Process App [page 192] for instructions.

## Using the Compensate Step

You use the **Compensate** step to reverse, or undo, the operations in an enclosed scope that already completed.

### Procedure Summary

1. In App Explorer, select an orchestration workflow that with nested **Scope** steps.

2. If necessary, expand the outermost **Scope** step so that all sections are visible.

3. In the **Step Palette**, drag a **Compensate** step onto the orchestration workflow editor and drop it in the **FaultHandler** or **CompensationHandler** section of the outermost **Scope** step.

   If you want to compensate for a specific named fault only, place the **Compensate** step in a **FaultHandler** section with a named fault. Otherwise, place the **Compensate** step in the **CompensationHandler** section.

4. In the Property Editor of the **Compensate** step, select the **Scope** to be compensated. You can also enter a description.

5. To specify the compensation actions, add other steps from the **Step Palette** to the **CompensationHandler** section, to the right of the **Compensate** step.

6. If you are using nested scopes, you can use a **Throw** step to pass a fault from any inner scopes to an outer scope.

### Tutorial: Creating an Empty Asynchronous Orchestration Workflow for the Compensate Step

You use the **Compensate** step in combination with nested **Scope** steps. The Web services or actions defined in the **Compensate** step should compensate or roll back whatever was done in the main part of the inner nested **Scope** step. This is particularly useful if these actions have taken some time to complete.

In this exercise, you add an empty asynchronous orchestration workflow to the **FaultHandlingOrch** in the **FaultHandlingProcApp**.

**To create an empty asynchronous orchestration workflow that uses the Compensate step:**

1. In App Explorer, under **Application Workflows**, click **CompensateAWF**.

2. In the application workflow editor, right-click the **DemoCompensate** transition, and then select **Show Actions**.

3. On the **Actions** tab of the Property Editor, click **New**.

   The **Action Wizard** opens and asks, "Which type of action do you want to execute?"

   Under **Step 1**, **Orchestration Workflow** should be selected and **Step 2** should read **Invoke and orchestration workflow and continue executing (asynchronous) using the local event**.

4. Click **Next**.

   The **Action Wizard** asks, "What do you want to affect?"

   Under **Step 1**, **This item** should be selected and **Step 2** should read **Invoke an orchestration workflow and continue executing (asynchronous) using the local event, affect this item**.

5. Click **Next**.

   The **Action Wizard** asks, "Which condition do you want to check?"

   Under **Step 1**, **Unconditionally** should be selected and **Step 2** should still read **Invoke an orchestration workflow and continue executing (asynchronous) using the local event, affect this item**.

6. Click **Next**.

   The **Action Wizard** asks, "Which orchestration workflow to you want to invoke?"

7. Under **Step 1**, select **(Add new workflow...)**.

   **NewDemoCompensateWorkflow** appears under **Step 1** and should be selected. **Step 2** should read **Invoke an orchestration workflow and continue executing (asynchronous) using the local event, affect this item | invoke event (FaultHandlingPTable:CompensateAWF_New_DemoCompensate)**.

   **SubmitGetTickerSymbolWorkflow** appears below **ThrowSyncOWF** under **Orchestration Workflows** in App Explorer.

8. Click **Finish**.

9. Click **NewDemoCompensateWorkflow**, and then, on the **General** tab of the Property Editor, change the **Name** to `CompensateOWF`.

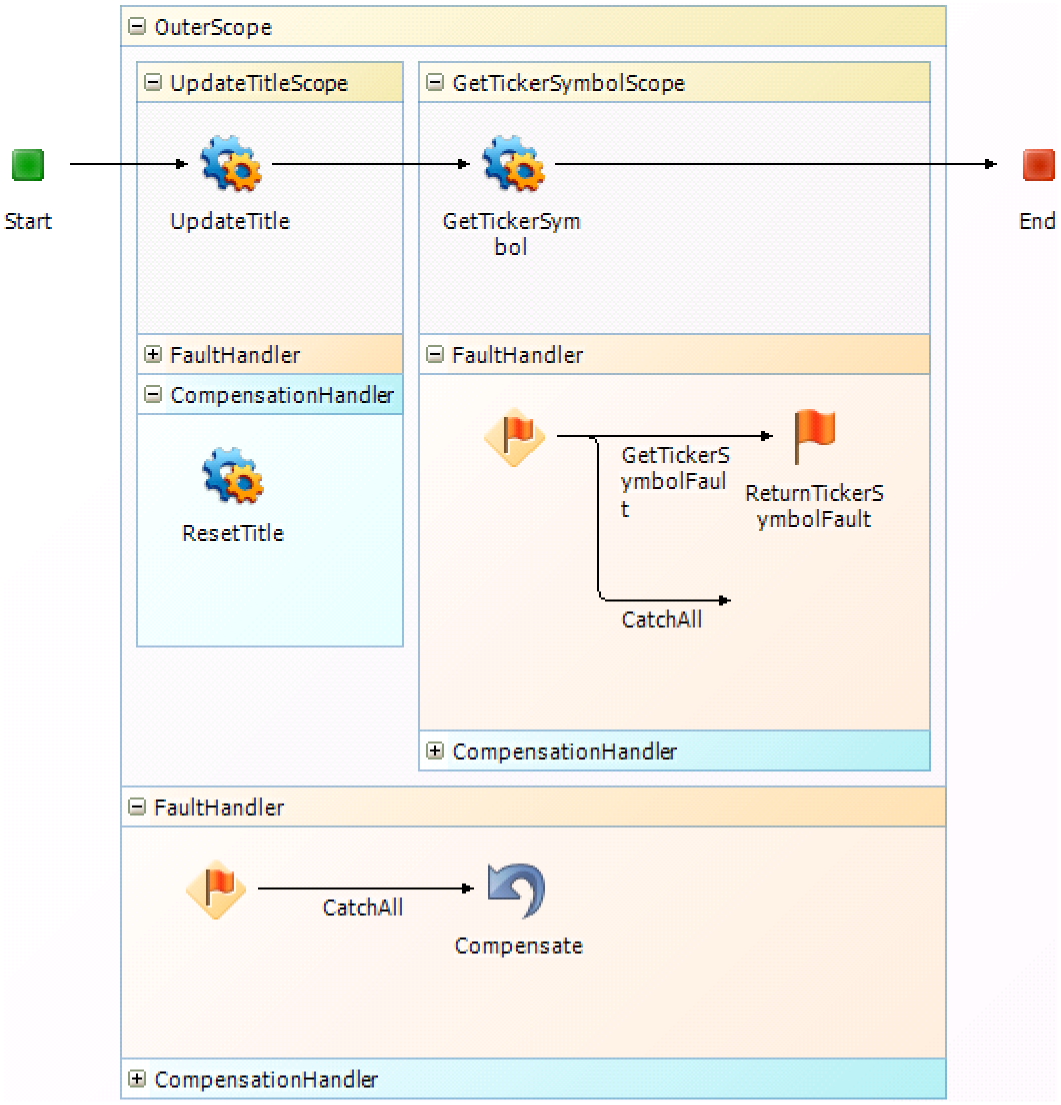
## Tutorial: Practicing with the Compensate Step

In this exercise, you use a **Compensate** step in the **CompensateOWF** orchestration workflow.

**Note:** For this tutorial, the purpose of the SerenaSampleTickerService Web service is to return a named fault, not to get the results of any of its operations.

After you complete the steps in this exercise, your orchestration workflow should look like the one in the following figure:

*Figure 1. CompensateOWF*



**To use the Compensate step in an asyncronous orchestration workflow:**

1. In App Explorer, select **CompensateOWF**.
2. In the **New Items** section of the **Step Palette**, drag a **Scope** step onto the orchestration workflow editor, and drop it onto the line between the **Start** and **End** steps.
3. On the **General** tab of the Property Editor, change the **Name** to `OuterScope`, and then press the Tab key.
4. In the **New Items** section of the **Step Palette**, drag a **Scope** step onto the orchestration workflow editor, and drop it inside the top section of the **OuterScope** step.

5. On the **General** tab of the Property Editor, change the **Name** to `UpdateTitleScope`.
6. In the **Configured Items** section of the **Step Palette**, drag an **sbmappservices72** Service step into the **UpdateTitleScope** step, and drop in the top section.
7. On the **General** tab of the Property Editor, change the **Name** to `UpdateTitle`.
8. From the **Operation** menu, select **TransitionItem.**
9. On the **Data Mapping** tab, expand **item**, **id**; locate the **tableIdItemId** data element; select the corresponding cell in the **Source elements** column; and then click the down arrow.
10. In the **Select a source** popup that opens, under **CompensateOWF**, **Inputs**, **EventNotice**, **Extension**; select **ItemId_TableRecId**; and then click **OK**.
11. Locate the **title** data element, and enter the following in the corresponding cell in the **Default value** column: `Updated by CompensationOWF`.
12. Expand the **CompensationHandler** of the **UpdateTitleScope** step.
13. In the **Configured Items** section of the **Step Palette**, drag an **sbmappservices72** Service step into the **CompensationHandler** of the **UpdateTitleScope** step.
14. On the **General** tab of the Property Editor, change the **Name** to `ResetTitle`.
15. From the **Operations** menu, select **TransitionItem**.
16. On the **Data Mapping** tab, expand **item**, expand **id**; locate the **tableIdItemid** data element; select the corresponding cell in the **Source elements** column; and then click the down arrow.
17. In the **Select a source** popup that opens, under **CompensateOWF**, **Inputs**, **EventNotice**, **Extension**; select **ItemId_TableRecId**; and then click **OK**.
18. Locate the **description** data element, and enter the following in the corresponding **Default value** column: `Returning Title field to its original value`.
19. Collapse the **UpdateTitleScope** step.
20. In the **New Items** section of the **Step Palette**, drag a **Scope** step into the top section of the **OuterScope** step, and drop it on the right of the **UpdateTitleScope** step.
21. On the **General** tab of the Property Editor, change the **Name** to `GetTickerSymbolScope`.
22. In the **Configured Items** section of the **Step Palette**, drag a **SerenaSampleTickerService** Service step into the top section of the **GetTickerSymbolScope** step.
23. On the **General** tab of the Property Editor, change the **Name** to `GetTickerSymbol`.
24. On the **Operation** menu, select **GetTickerSymbol**.

25. On the **Data Mapping** tab, locate the **company** data element; select the corresponding cell in the **Source elements** column; and then click the down arrow.

26. In the **Select a source** popup that opens, under **CompensateOWF**, **Inputs**, **EventNotice**, **Extension**; select **Title**; and then click **OK**.

27. Expand the **FaultHandler** section of the **GetTickerSymbolScope** step.

28. Right-click the **Fault Handler** step, and then select **Insert New Catch**.

29. Select **SerenaSampleTickerService-GetTickerSymbolFault** from the **Fault name** list.

30. On the **General** tab of the Property Editor, change the **Name** to `GetTickerSymbolFault`.

    **SerenaSampleTickerService-GetTickerSymbolFault** automatically appears in the **Fault message** box.

31. In the **New Items** section of the **Step Palette**, drag a **Throw** step into the **FaultHandler** section of the **GetTickerSymbolScope** step, and drop it onto the **GetTickerSymbolFault** branch.

32. On the **Data Mapping** tab, locate the **FaultString** data element, select the corresponding cell in the **Source elements** column; and then click the down arrow.

33. In the **Select a source** popup that opens, under **GetTickerSymbolFault**, **Outputs**, **GetTickerSymbolFault**; select **detail**; and then click **OK**.

34. Expand the **FaultHandler** section of the **OuterScope** step.

35. In the **New Items** section of the **Step Palette**, drag a **Compensate** step into the **FaultHandler** section, and drop it onto the **CatchAll** branch.

    On the **General** tab of the Property Editor, the **Scope** should be **[Default Scope]**.

36. On the Quick Access Toolbar, click the **Validate** button.

    The following warning message appears in the Validation Results: **Compensation handler is empty.**

    You can ignore this message, because a compensation handler is not required for this orchestration workflow.

37. Publish and deploy the **FaultHandlingProcApp.**

    See Step 6: Publish the Process App [page 192] and Step 7: Deploy the Process App [page 192] for instructions.

## Running the Fault Handling Process App

This section contains exercises for running the various FaultHandlingProcApp projects (application workflows). The steps in each tutorial are correlated with the way elements in the orchestration workflows and the application workflows interact.

Following is a list of the tutorials in this section.

- SerenaSampleTickerService Company Names and Ticker Symbols [page 130]

- Tutorial: Running the GenericFaultAWF Project [page 130]
- Tutorial: Altering the GenericFaultOWF to Return a Web Service Fault [page 131]
- Tutorial: Running the GenericFaultAWF Project and Invoking the CatchAll Branch [page 131]
- Tutorial: Running the NamedFaultAWF Project and Invoking a Catch Branch [page 133]
- Tutorial: Running the ThrowAWF Project [page 134]
- Tutorial: Running the CompensateAWF Project [page 135]

### SerenaSampleTickerService Company Names and Ticker Symbols

Several of the tutorials in this section use the SerenaSampleTickerService. The companies listed on this Web service and their ticker symbols are shown in the following table. In some of the steps, you are required to use the values in this table.

| Company Name | Ticker Symbol |
|---|---|
| Aftabitorium | AFT |
| Bluejay Bird Supply | BBS |
| Carol Creations | CCR |
| Meg N Ah Electric Co. | MAC |
| Rob & Bert Manufacturing | RBM |
| Tim Buck 2 Entertainment | TBE |

### Tutorial: Running the GenericFaultAWF Project

In this exercise, you run the GenericFaultAWF Project in SBM Work Center.

**To run the GenericFaultAWF Project:**

1. Log on to SBM Work Center.
2. Click the **FaultHandlingApp** icon.
3. Click **+New**.
4. On the **Browse** tab, click the **GenericFaultAWF Project** link.

   The **Submit** transition form opens.
5. In the **Title** field, enter your user ID, and then click **OK**.

   The message "This user is valid" is returned in the **Description** field.
6. Click **+New**.

7. On the **Browse** tab, click the **GenericFaultAWF Project** link.
8. In the **Title** field of the **Submit** transition form, enter an invalid user ID such as `InvalidUser`, and then click **OK**.

   The message "This user is not valid" is returned in the **Description** field.

## Tutorial: Altering the GenericFaultOWF to Return a Web Service Fault

In this exercise, you alter the **GenericFaultOWF** orchestration workflow by mapping an invalid user ID. When the orchestration workflow is invoked, a Web service fault is generated.

**To alter the GenericFaultOWF to return a Web service fault:**

1. Open SBM Composer.
2. Click the Composer button, and then click **Open**.

   The **Open Process App** dialog box opens.
3. Select **FaultHandlingProcApp**, and then select **Open**.
4. In App Explorer, click **FaultHandlingProcApp**, and then click the **All Items** filter.
5. Under **Orchestration Workflows**, click **GenericFaultOWF**.
6. In the orchestration workflow editor, select the **VerifyUser** step.
7. On the **Data Mapping** tab of the **Property Editor**, expand the **auth** data element, locate the **userId** data element, and enter an invalid user ID such as `zzzz` in the corresponding **Default value** column.
8. Publish and deploy the **FaultHandlingProcApp.**

   See Step 6: Publish the Process App [page 192] and Step 7: Deploy the Process App [page 192] for instructions.

## Tutorial: Running the GenericFaultAWF Project and Invoking the CatchAll Branch

In this exercise, you run the altered GenericFaultAWF Project in SBM Work Center.

**To run the GenericFaultAWF Project and return an error message:**

1. Log on to SBM Work Center.
2. Click the **FaultHandlingPro** icon.
3. Click **+New**.
4. On the **Browse** tab, click the **GenericFaultAWF Project** link.

   The **Submit** transition form opens.
5. Enter some text in the **Title** field, and then click **OK**.

The message `An unknown error occurred at GenericFaultAWF_GenericFaultOWF_VerifyUser` is returned in the **Description** field.

**Note:** The orchestration workflow will fail when you invoke it, so it does not matter what you enter in the **Title** field. However, you must enter something, because this is a required field.

6. To find out more about the error, you can return to the **FaultHandlingProcessApp** in SBM Composer and look for the associated SOAP fault in the Common Log Viewer as follows:

   a. If the **Common Log Viewer** tab is not visible, on the Ribbon, in the **Common Views** group, select the **Common Log Viewer** check box.

   b. On the **Overview** tab of the Log Viewer, click **Refresh**.

      **GenericFaultOWF** should be selected.

   c. On the **Details** tab, select the latest run on the **Run** tab.

   d. Look for the error message that begins with `A fault occurred during the execution of the orchestration…`

   e. Right-click anywhere in the message row, and then select **Show Message**.

      The **Message Detail** dialog box opens and shows the content of the SOAP message.

   f. Click the **Previous** button until you see the message that contains the SOAP fault. This message begins as follows: `A Web service was invoked at Service step VerifyUser, and now the Orchestration Engine is receiving the following message`.

   g. In the SOAP message that follows, locate the SOAP fault, which begins with `<SOAP-ENV:Fault`.

      The received SOAP message for the **GenericFaultApp Project** should contain the following information:

```
…
<SOAP-ENV:Fault xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <faultcode>SOAP-ENV:Server</faultcode>
  <faultstring>Invalid User ID or Password</faultstring>
  <detail>
    <ae:AEWebservicesFault xmlns:ae='urn:sbmappservices72'>Invalid
      User ID or Password</ae:AEWebservicesFault>
  </detail>
</SOAP-ENV:Fault>
…
```

      Note that the SOAP fault indicates an invalid user ID or password.

   h. Click the **Previous** button again until you see the SOAP message that contains the source of the error. This message begins as follows: `A Web service is being invoked at Service step VerifyUser, and the Orchestration Engine is sending the following message`.

The sent SOAP message for the **GenericFaultApp Project** should contain the following information:

```
…
xmlns:ns8='urn:sbmappservices72'><ns8:auth><ns8:userId>ZZZZZ</ns8:userId>
  </ns8:auth><ns8:loginId
…
```

Note the incorrect entry for `userID`.

7. After you complete this part of the tutorial, you can restore **GenericFaultOWF** to its original, valid state by deleting the invalid **userId** value and redeploying the process app. If you do not do this, you can still run the other projects in the **FaultHandlingProcessApp**.

## Tutorial: Running the NamedFaultAWF Project and Invoking a Catch Branch

In this exercise, you run the NamedFaultAWF Project in SBM Work Center.

**To run the NamedFaultAWF Project:**

1. Log on to SBM Work Center.
2. Click the **FaultHandlingApp** icon.
3. Click **+New**.
4. On the **Browse** tab, click the **NamedFaultAWF Project** link.

   The **Submit** transition form opens.
5. In the **Title** field, enter one of the business names from the table in SerenaSampleTickerService Company Names and Ticker Symbols [page 130], and then click **OK**.

   The ticker symbol is returned in the **Title** field of the state form.
6. Click **GetBuyRating**.

   The **GetBuyRating** transition form opens.
7. Click **OK**.

   The buy rating is returned in the **Description** field of the state form. The **Description** field should contain the following message: "This company has a strong-buy rating."
8. Click **+New**.
9. On the **Browse** tab, click the **NamedFaultAWF Project** link.
10. In the **Title** field, type some invalid input such as `Acme Company`, and then click **OK**.

    The **SerenaSampleTickerService-GetTickerSymbolFault** message appears in the **Description** field. If you typed `Acme Company`, the message should read as follows: "No information is available for 'acme company'. The companies listed on this service are 'Rob & Bert manufacturing', 'Aftabitorium', 'Meg N Ah Electric Co.', 'BlueJay Bird

Supply', 'Tim Buck 2 Entertainment', and 'Carol Creations'." If you click **GetBuyRating**, the **NamedFaultOWF2** orchestration workflow is not invoked and the information in the **Title** and **Description** fields does not change.

## Tutorial: Running the ThrowAWF Project

In this exercise, you run the ThrowAWF Project in SBM Work Center.

**To run the ThrowAWF Project:**

1. Log on to SBM Work Center.
2. Click the **FaultHandlingApp** icon.
3. Click **+New**.
4. On the **Browse** tab, click the **ThrowAWF Project** link.

   The **Submit** transition form opens.
5. In the **Title** field, type one of the company names from the table in SerenaSampleTickerService Company Names and Ticker Symbols [page 130], except Bluejay Bird Supply, and then click **OK**.

   The **ThrowAWF Project** state form opens, and the text appears in the **Title** field.
6. Click **GetTickerSymbol**.

   The **Submit** transition form opens.
7. Click **OK**.

   The **ThrowOWF** orchestration workflow is invoked. The data from the **Title** field is passed to the **IsCompanyBluejay** Decision step, which chooses a branch based on its rule. In this case, the data is not "Bluejay Bird Supply," so it selects the **IsNotBlueJay** branch and passes the data to the SerenaSampleTickerService Web service.

   The Web service's **GetTickerSymbol** operation is invoked, and the Web service returns a ticker symbol for the company name that you typed in the **Title** field. The ticker symbol appears in the **Description** field of the state page.
8. Click **+New**.
9. On the **Browse** tab, click the **ThrowAWF Project** link.

   The **Submit** transition form opens.
10. In the **Title** field, type some text, such as `zzzz`. (Do not use any of the company names from the table in SerenaSampleTickerService Company Names and Ticker Symbols [page 130], including Bluejay Bird Supply.) Click **OK**.

    The **ThrowAWF Project** state form opens, and the text appears in the **Title** field.
11. Click **GetTickerSymbol**.

    The transition form between the **Submit** and **TickerSymbol** states opens.
12. Click **OK**.

The **ThrowOWF** orchestration workflow is invoked. The data from the **Title** field is passed to the **IsCompanyBluejay** Decision step, which chooses a branch based on its rule. In this case, the data in the **Title** field is not Bluejay Bird Supply, so it selects the **IsNotBluejay** branch and passes the data to the SerenaSampleTickerService Web service.

The Web service's **GetTickerSymbol** operation is invoked, and the Web service determines that the data is not valid. The Web service generates the **GetTickerSymbolFault**, which is caught by the **GetTickerSymbolFault** Catch branch.

The **Catch** branch passes the fault detail to the **ReturnGetTickerSymbolFault** Throw step, which throws a fault and the error text to the SBM Application Engine.

The SBM Application Engine stops the workflow and displays the following error message on the state form, below the transition buttons: `Error occurred during web service invocation: No information is available for 'ZZZZ'. The companies listed on this service are 'Rob & Bert manufacturing', 'Aftabitorium', 'Meg N Ah Electric Co.', 'BlueJay Bird Supply', 'Jay Buck 2 Entertainment', and 'Carol Creations'`. (The text after the colon is the fault detail.)

13. Click **+New**.
14. On the **Browse** tab, click the **ThrowAWF Project** link.

    The **Submit** transition form opens.
15. In the **Title** field, type **`Bluejay Bird Supply`**, and then click **OK**.

    The **ThrowAWF Project** state form opens, and the text appears in the **Title** field.
16. Click **GetTickerSymbol**.

    The **Submit** transition form opens.
17. Click **OK**.

    The **ThrowOWF** orchestration workflow is invoked. The data from the **Title** field is passed to the **IsCompanyBluejay** Decision step, which chooses a branch based on its rule. In this case, the data is "Bluejay Bird Supply." The **Decision** step selects the **IsBluejay** branch and passes the data to the **ReportBluejay** Throw step, which throws a fault and the text mapped in the **FaultString** data element to the SBM Application Engine.

    The SBM Application Engine stops the workflow and displays the following error on the state page, below the transition buttons: `Error occurred during web service invocation: SOAPFaultCode: ns1:ERROR_BLUEJAYSOAP Fault String: Bluejay Bird Supply is a credit risk.`

## Tutorial: Running the CompensateAWF Project

In this exercise, you run the CompensateAWF Project in SBM Work Center.

**To run the CompensateAWF Project:**

1. Log on to SBM Work Center.
2. Click the **FaultHandlingPro** icon.

3. Click **+New**.
4. On the **Browse** tab, click the **CompensateAWF Project** link.

   The **Submit** transition form opens.
5. In the **Title** field, type any of the company names from the table in SerenaSampleTickerService Company Names and Ticker Symbols [page 130], and then click **OK**.

   The **CompensateAWF Project** state form opens, and the company name appears in the **Title** field.
6. Click **DemoCompensate**.

   The **Submit** transition form opens.
7. Click **OK**, and then click the **Reload Item** button.

   The **CompensateOWF** orchestration workflow is invoked. The data from the **Title** field is passed to the **UpdateTitle** Service step. The **UpdateTitle** Service step is invoked and passes the text that is mapped to the **Title** field, "Updated by CompensateOWF," to the **GetTickerSymbol** Service step. The Web service's **GetTickerSymbol** Service step is invoked, and the text is returned to the SBM Application Engine, which displays "Updated by CompensateOWF" in the **Title** field of the state page.
8. Click **+New**.
9. On the **Browse** tab, click the **CompensateAWF Project** link.

   The **Submit** transition form opens.
10. In the **Title** field, type any text that is not contained the company names from the table in SerenaSampleTickerService Company Names and Ticker Symbols [page 130], and then click **OK**.

    The **CompensateAWF Project** state form opens, and the text you typed appears in the **Title** field.
11. Click **DemoCompensate**.

    The **Submit** transition form opens.
12. Click **OK**, and then click the **Reload Item** button.

    The **CompensateOWF** orchestration workflow is invoked. The Web service's GetTickerSymbol operation is invoked, and the Web service determines that the data is not valid. The Web service generates the **GetTickerSymbolFault**, which is caught by the **GetTickerSymbolFault** Catch branch. This branch passes the fault detail to the **ReturnTickerSymbolFault** Throw step, which throws a fault to the **Compensate** step in the **FaultHandler** section of the **OuterScope**. The **Compensate** step looks for any enclosed, successfully completed scopes and finds the **UpdateTitle** scope. The **CompensationHandler** section for this scope contains a **Service** step to invoke the SBM Application Engine **TransitionItem** operation, which rolls back the change to the **Title** field that happened in the **UpdateTitle** scope. Note that this scope changed the **Title** field to "Updated by CompensateOWF," but the **Compensate** step is now rolling back that change. It also changes the **Description** field to "Returning Title to its original value." The

Application Engine displays the original data from the **Title** field in the **Title** field of the state form and "Returning Title to its original value." in the **Description** field.

## Raising External Events

See Raising Events from External Products [page 145] for a procedure that shows you how to raise external events.

# Chapter 4: Orchestration Use Cases

The use cases in this section describe how to implement common use cases that use orchestrations.

This section contains the following use cases:

## Building Dynamic Arrays

In SBM Composer, you can build an array with a fixed number of array elements at *design time*. This is known as a *fixed array*.

However, sometimes you must determine the number of array elements at *runtime* (that is, when an orchestration workflow runs). For example, you want to send telephone numbers to an SMS (Short Message Service) Web service. You use a **Service** step in an orchestration workflow to get the telephone numbers of those users who were selected on a form.

Because you do not know the number of selected users and telephone numbers in advance, you build a *dynamic array*. For an orchestration workflow to dynamically add elements to an array, you use a **Calculate** step to target each element in the array in sequential order, beginning with 1.

This section includes use cases that illustrate ways to make Web service calls that accept a dynamically-sized list of elements. They are for demonstration purposes only, and do not represent realistic scenarios.

**Important:** If you do not use dynamic arrays, you must make multiple Web service calls with one element in each array, or create an array in advance that has a fixed length (for example, a 20-element array). These methods have a negative impact on performance.

**CAUTION:**

Array elements must be ordered chronologically. Do not split arrays or interject an array element into the middle of an array. For example, you cannot interject an array element between two existing array elements, and you cannot create a seventh array element if there is no sixth array element.

## Use Case: Creating an Array to Use in a Subsequent Service Step

In this use case, you get an array of file names from an auxiliary table that stores Source Code Management (SCM) application information, and then create a new item for each file in another auxiliary table that stores Issue Defect Management (IDM) information.

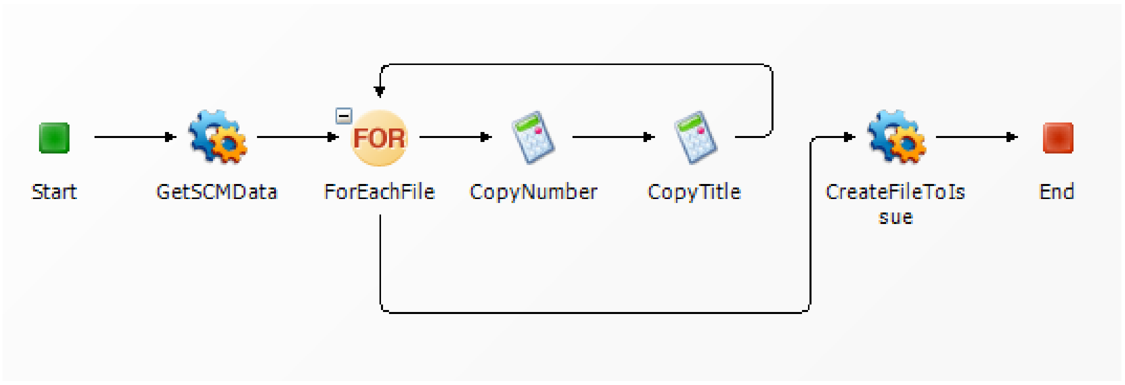
**Note:** Similar data can come from any external Web service. That is, instead of coming from an auxiliary table, the data about files and associated issues can come from your SCM application.

The orchestration workflow loops through each file and then processes it as follows:

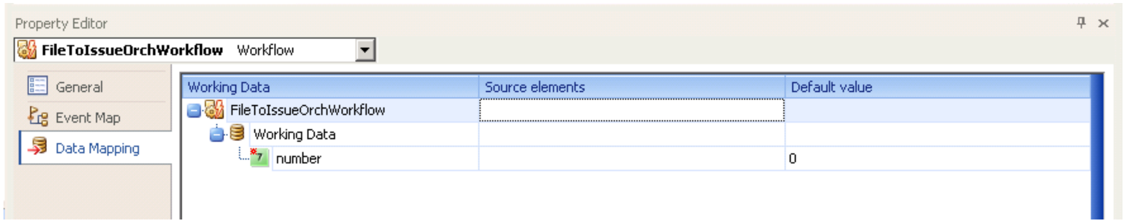
1. Gets an array of files from an auxiliary table.
2. Adds the file names to an input element in a subsequent **Service** step.
3. Creates new items in another auxiliary table.

**To create an array and use it in a subsequent Service step:**

1. Add steps to the orchestration workflow as shown below.

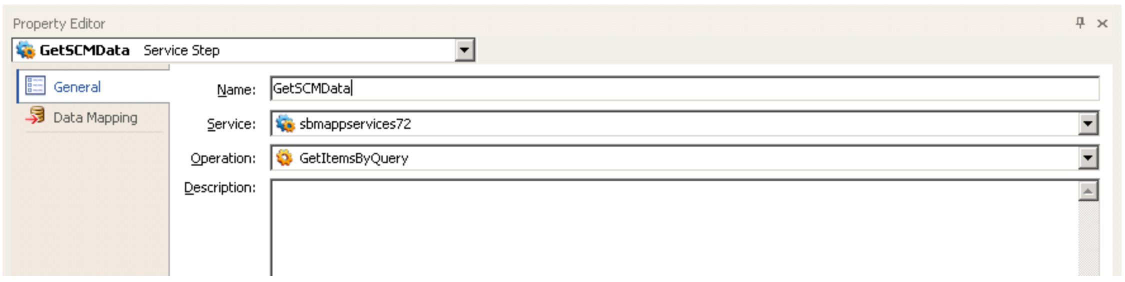


2. In the **Working Data** for the workflow, add a variable to store the number of loops through the **ForEachFile** step, using the type "Integer." Set the default value of the variable to `0`.



| Working Data | Source elements | Default value |
|---|---|---|
| FileToIssueOrchWorkflow | | |
| Working Data | | |
| number | | 0 |

3. Configure the **GetSCMData** step to get a list of file names from the auxiliary table for the SCM application.


Property Editor

**GetSCMData** Service Step

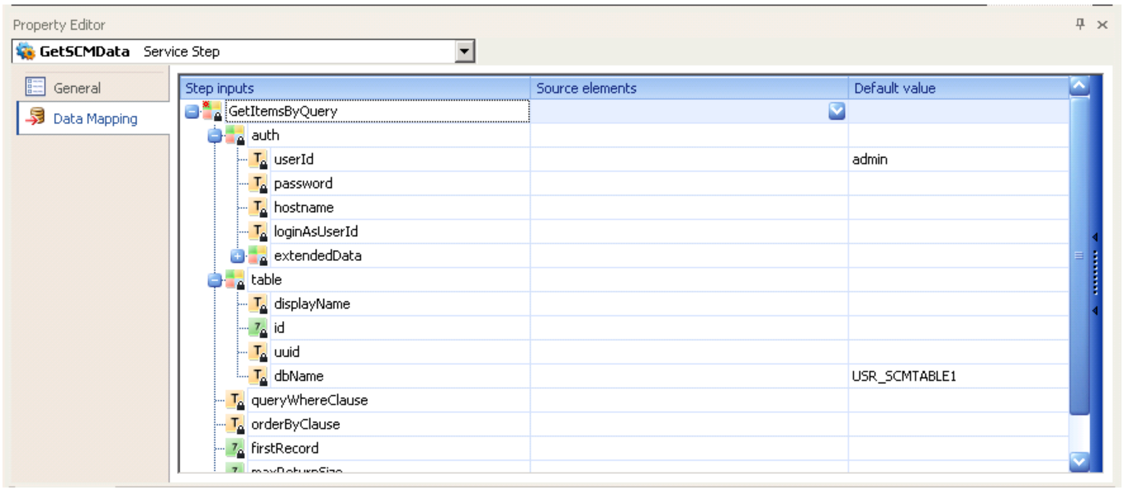
General
Data Mapping

Name: GetSCMData

Service: sbmappservices72

Operation: GetItemsByQuery

Description:

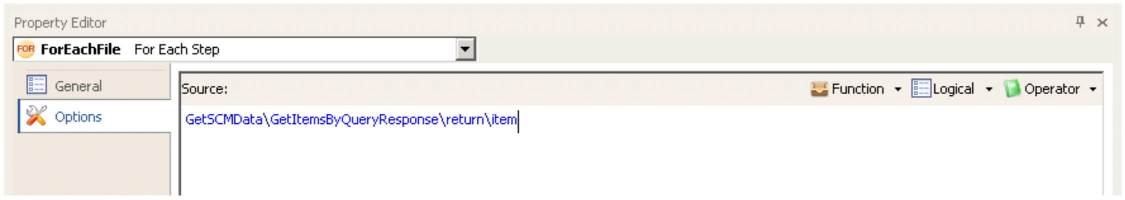

Property Editor

**GetSCMData** Service Step

General
Data Mapping

| Step inputs | Source elements | Default value |
|---|---|---|
| GetItemsByQuery | | |
| auth | | |
| userId | | admin |
| password | | |
| hostname | | |
| loginAsUserId | | |
| extendedData | | |
| table | | |
| displayName | | |
| id | | |
| uuid | | |
| dbName | | USR_SCMTABLE1 |
| queryWhereClause | | |
| orderByClause | | |
| firstRecord | | |

4. Configure the **ForEachFile** step to repeat the operations in the loop for each file returned in the **GetSCMData** response.


Property Editor

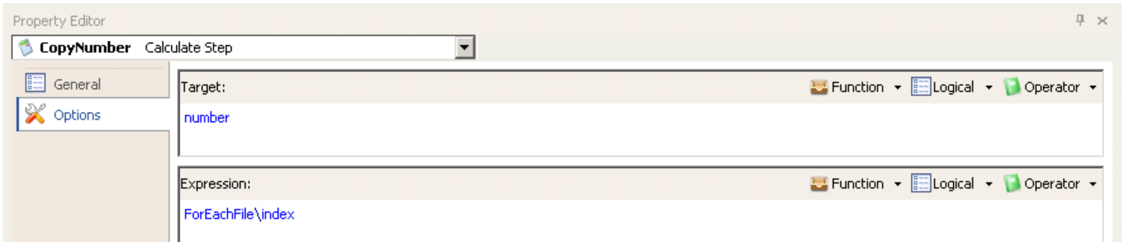
**ForEachFile** For Each Step

General
Options

Source:

GetSCMData\GetItemsByQueryResponse\return\item

5. Configure the **CopyNumber** step to store the number of times the orchestration workflow loops through the **ForEachFile** step.


Property Editor

**CopyNumber** Calculate Step

General
Options

Target:

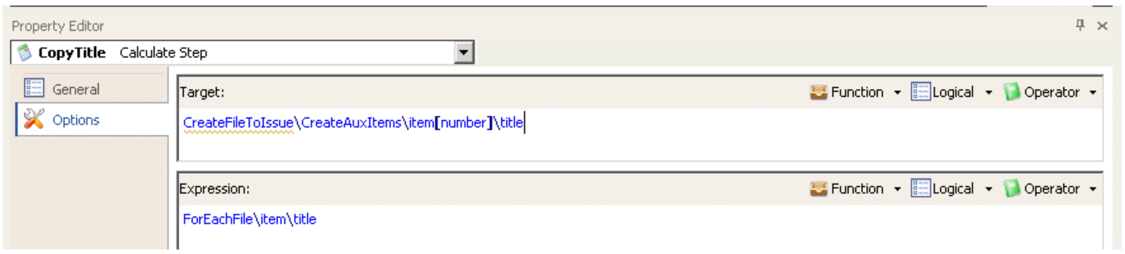
number

Expression:

ForEachFile\index

**Note:** This step must be included. You must copy the number of loops into the working data; you cannot use the **ForEachFile** step by itself.

6. Configure the **CopyTitle** step to set the value of the **title** in the **item[]** array in the **CreateFiletoIssue** step. This value is a file name returned by the **GetSCMData** step.



Property Editor

**CopyTitle** Calculate Step

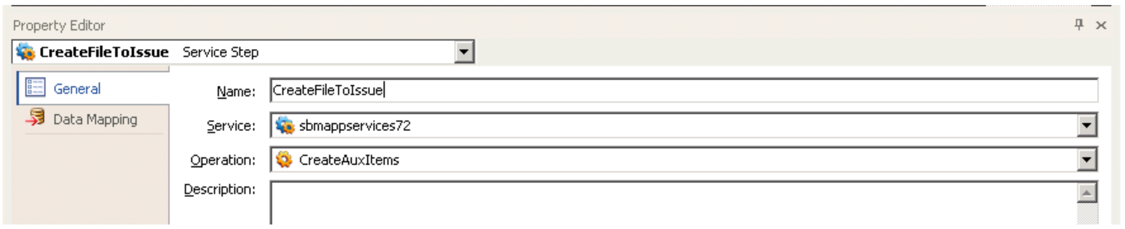
Target:

CreateFileToIssue\CreateAuxItems\item[number]\title

Expression:

ForEachFile\item\title

7. Configure the **CreateFileToIssue** step to create a new item in another auxiliary table for each file that was returned by the **GetSCMData** step.



Property Editor

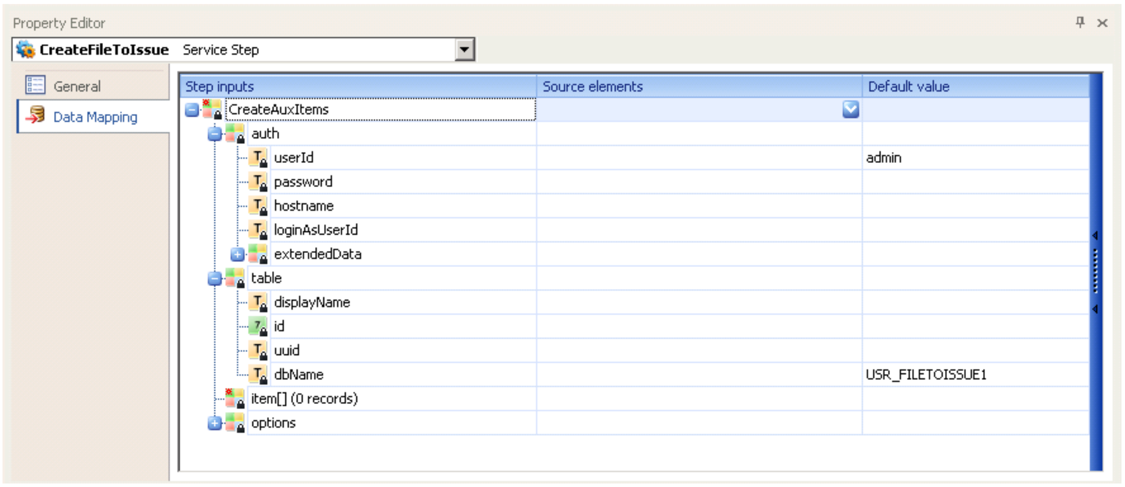
**CreateFileToIssue** Service Step

Name: CreateFileToIssue

Service: sbmappservices72

Operation: CreateAuxItems

Description:



Property Editor

**CreateFileToIssue** Service Step

| Step inputs | Source elements | Default value |
|---|---|---|
| CreateAuxItems | | |
| auth | | |
| userId | | admin |
| password | | |
| hostname | | |
| loginAsUserId | | |
| extendedData | | |
| table | | |
| displayName | | |
| id | | |
| uuid | | |
| dbName | | USR_FILETOISSUE1 |
| item[] (0 records) | | |
| options | | |

## Use Case: Populating Custom Fields

In this use case, you add two *Text* custom fields to the primary table: *Cities* and *Name*. Then you dynamically create an extended field list in a **Service** step that populates these fields.

**Note:** In this use case, you use working data as input to the **ForEachNode** loop, and the **ForEachNode** loop creates an extended field list as an input to the **UpdateItem** step. The same logic can be used when an event contains array records (that is, you have arrays included in the **Extension** data in an event definition). Instead of using the working data as the input to the **ForEachNode** step, you can use the **EventNotice**.

The orchestration workflow loops through each node and processes it as follows:

1. Sets the database name of the first field in the extended field list.

2. Sets the value of the first field in the extended field list.

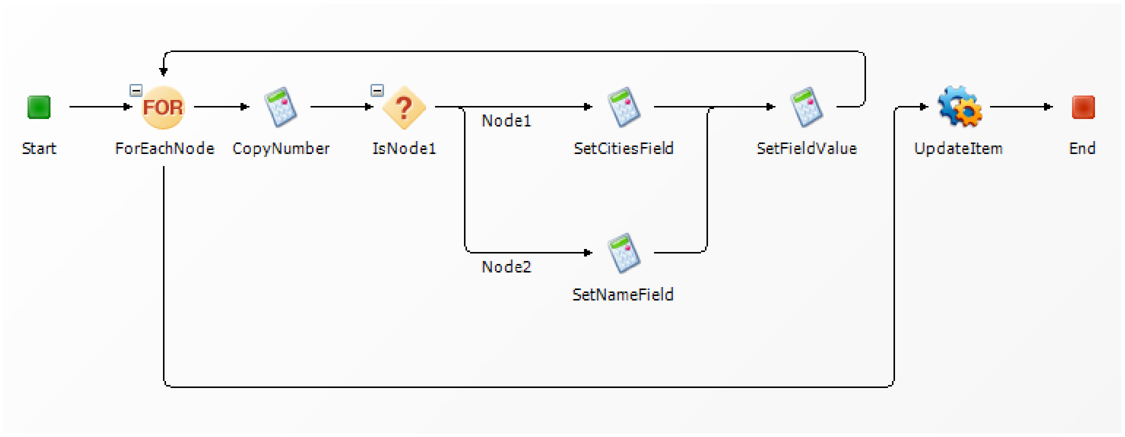
3. Sets the database name of the second field in the extended field list.

4. Sets the value of the second field in the extended field list.

5. Updates the fields in the SBM item with information in the extended field list.

**To dynamically create an extended field list that populates custom fields:**

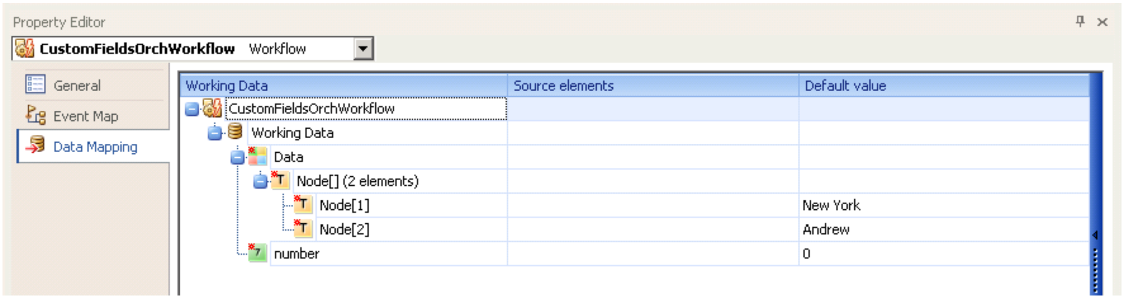
1. Add steps to the orchestration workflow as shown below.



2. Define the working data for the orchestration workflow:

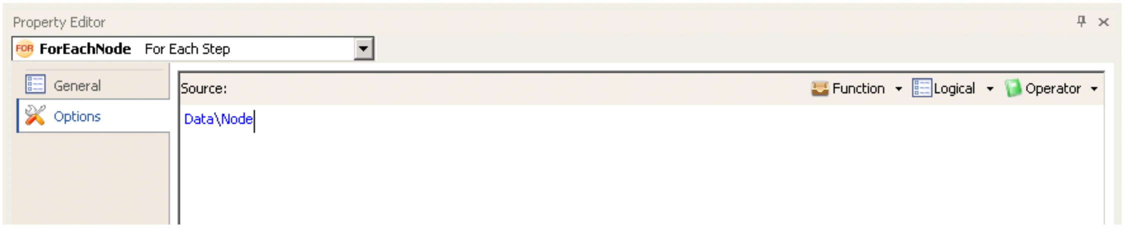
   a. Create an array with two array elements. Set a default value for each array element.

   b. Add a variable to store the number of loops through the **ForEachNode** step, using the type "Integer." Set the default value of the variable to `0`.



| Working Data | Source elements | Default value |
|---|---|---|
| CustomFieldsOrchWorkflow | | |
| Working Data | | |
| Data | | |
| Node[] (2 elements) | | |
| Node[1] | | New York |
| Node[2] | | Andrew |
| number | | 0 |

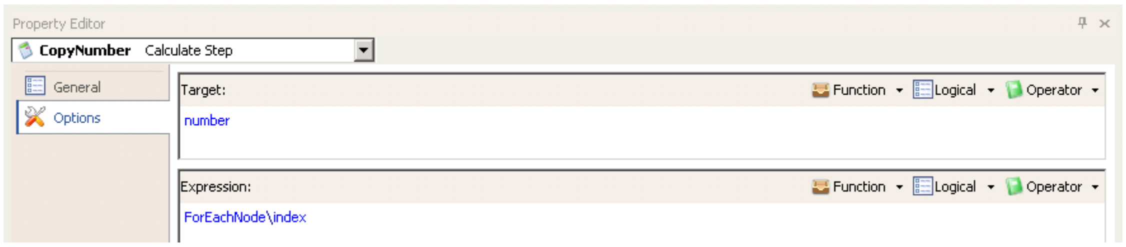
3. In the **ForEachNode** step, enter an expression that describes the source of the data to be processed. In this example, the **Node** array is the source of the data.



Source:

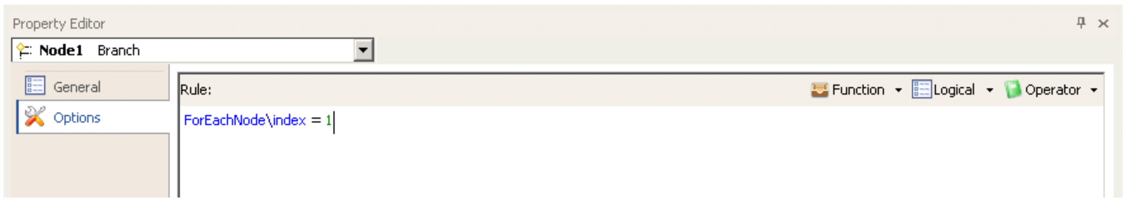
Data\Node

4. Configure the **CopyNumber** step to store the number of times the orchestration workflow loops through the **ForEachNode** step.

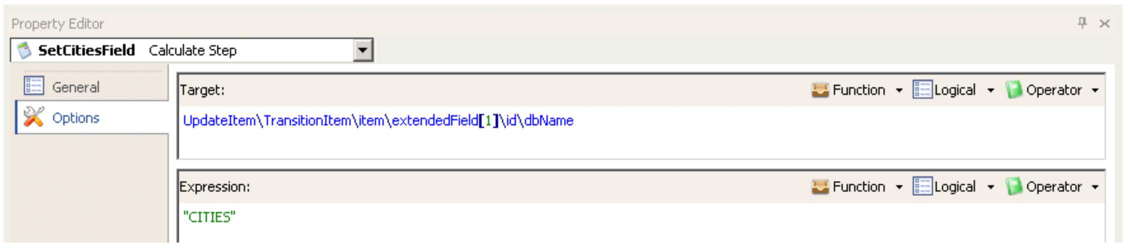

Property Editor
CopyNumber Calculate Step
Target:
number
Expression:
ForEachNode\index

**Note:** This step must be included. You must copy the number of loops into the working data; you cannot use the **ForEachNode** step by itself.

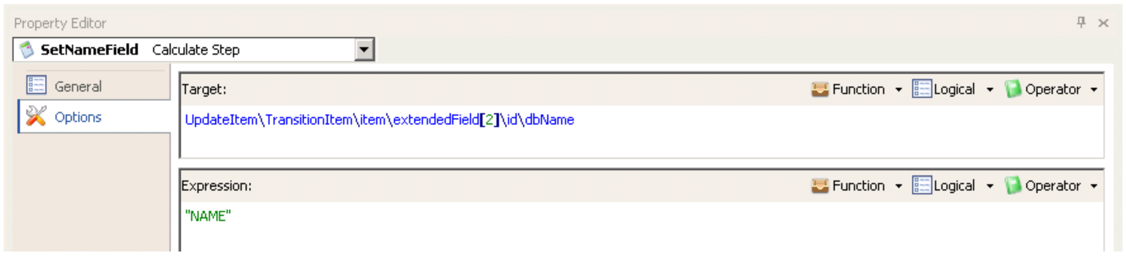
5. Configure the **Node1** branch to set the index value for the "cities" node to "1."


Property Editor
Node1 Branch
Rule:
ForEachNode\index = 1

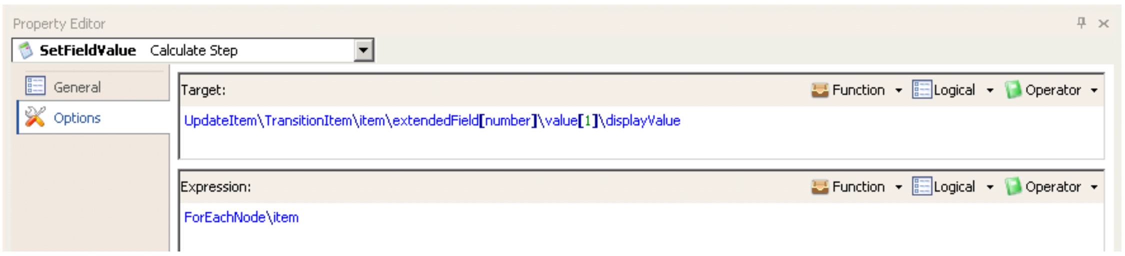
6. Configure the **SetCitiesField** step to set the database name of the first extended field to CITIES.


Property Editor
SetCitiesField Calculate Step
Target:
UpdateItem\TransitionItem\item\extendedField[1]\id\dbName
Expression:
"CITIES"

7. Configure the **SetNameField** step to set the database name of the second extended field to NAME.


Property Editor
SetNameField Calculate Step
Target:
UpdateItem\TransitionItem\item\extendedField[2]\id\dbName
Expression:
"NAME"

8. Configure the **SetFieldValue** step to set the value of the applicable extended field.


Property Editor

SetFieldValue Calculate Step

General | Options

Target:

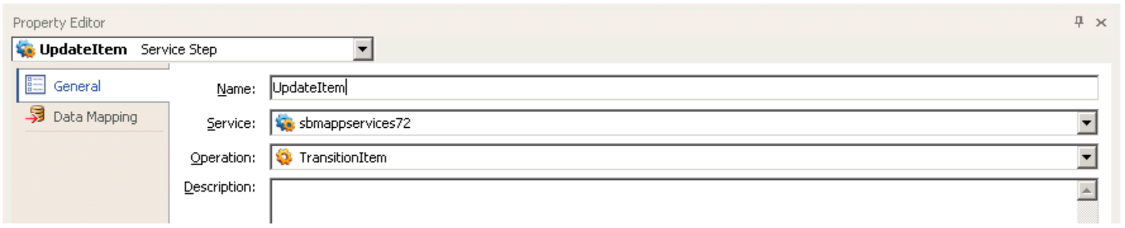
UpdateItem\TransitionItem\item\extendedField[number]\value[1]\displayValue

Expression:

ForEachNode\item

**Note:** In this use case, you use the first array element to create the CITIES extended field and the second array element to create the NAME extended field. The index used in the **Target** expression for the **SetCitiesField** and **SetNameField** steps shows the first array element using **[1]**, and the second array element using **[2]**. The order in which these fields are defined in the application table does not matter; you can use any order in the orchestration workflow. However, after the order is set, the values for those fields must be in the same order. For example, after **[1]** is assigned "CITIES," its value will be **extendedField[1].value[1].displayValue**.

9. Configure the **UpdateItem** step to populate the custom fields in SBM items.


Property Editor

UpdateItem Service Step

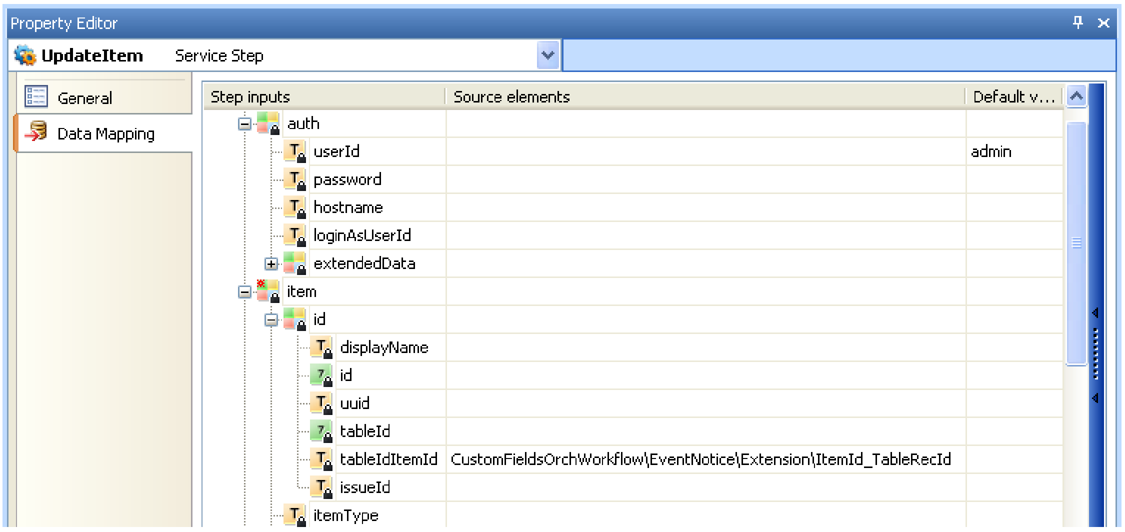
General | Data Mapping

Name: UpdateItem

Service: sbmappservices72

Operation: TransitionItem

Description:


Property Editor

UpdateItem Service Step

General | Data Mapping

| Step inputs | Source elements | Default v... |
|---|---|---|
| auth | | |
| userId | | admin |
| password | | |
| hostname | | |
| loginAsUserId | | |
| extendedData | | |
| item | | |
| id | | |
| displayName | | |
| id | | |
| uuid | | |
| tableId | | |
| tableIdItemId | CustomFieldsOrchWorkflow\EventNotice\Extension\ItemId_TableRecId | |
| issueId | | |
| itemType | | |

## Raising Events from External Products

**Note:** This topic assumes that you understand events and event definitions. For information about them, see About Events [page 27] and About Application Links and Event Definitions [page 28].

Any external product that is capable of calling a Web service can raise events in SBM by calling a corresponding Web service. SBM Composer provides the ability to automatically generate a `.wsdl` file that you can use in the external product to call the Web service.

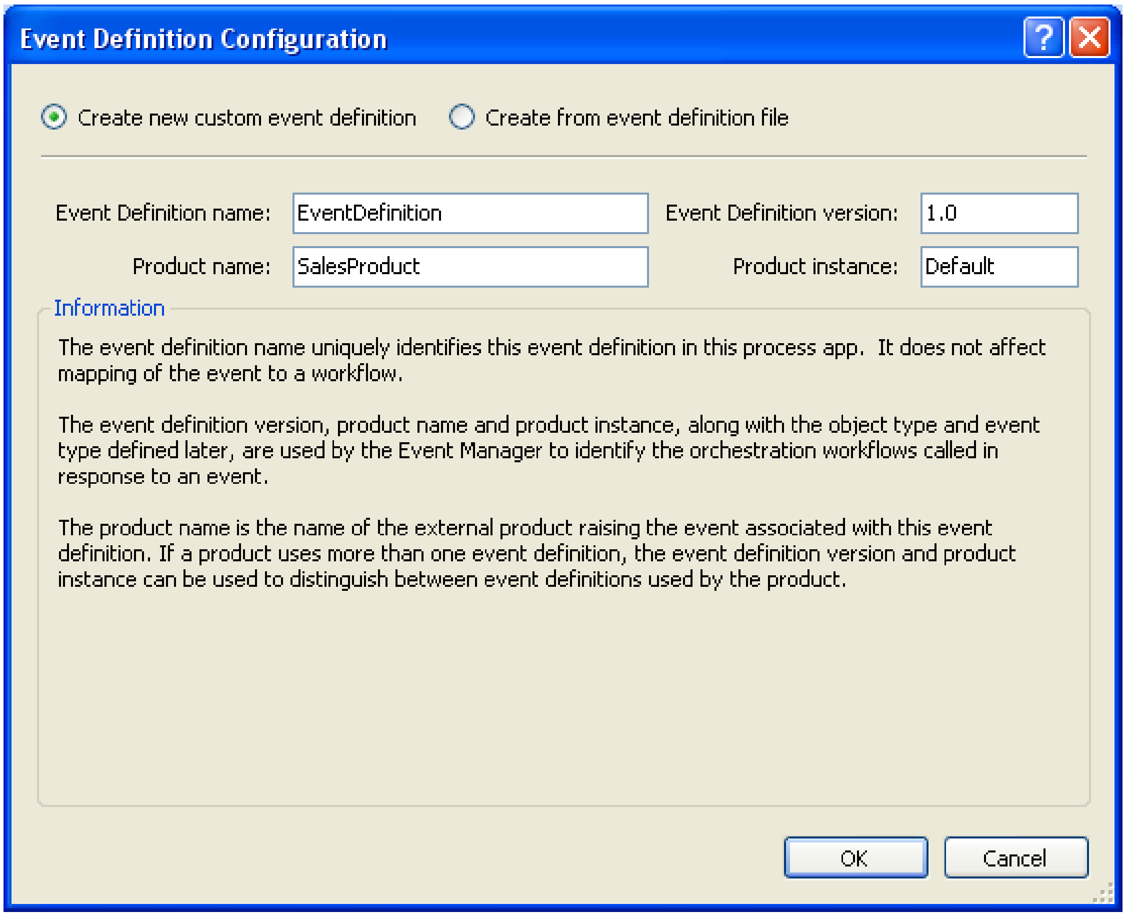
For example, suppose you use Salesforce.com to track customer prospects, and want to have a new item created in SBM whenever a potential customer is initially contacted. You can raise an external event that causes an asynchronous orchestration workflow to run whenever this event occurs. The orchestration workflow creates the item in SBM.

Because the event provides the inputs for the workflow, the `.wsdl` file must be designed specifically for the event and its associated orchestration workflows. You start by defining a custom event definition that defines the data you want to pass to the workflow. This event definition defines both the event that the external product will raise and the inputs for the orchestration workflow. The event definition determines the content of the `.wsdl` file. When you click the **Export external event WSDL** button in the event definition Property Editor, the `.wsdl` file that will be used to raise the event is created.

The following procedure describes a way to raise an external event that updates a field in a SBM item. In this procedure, a sales product sends an external event to update the *Discount Percent* field. In SBM Composer, you define two custom data fields in a custom event definition: **ItemID** and **DiscountPercent**. The orchestration workflow that is invoked when the external event is raised updates the item with the specified discount amount.

**To update the discount percent value in an SBM item:**

1. Create a new orchestration process app.
2. In App Explorer, right-click **Application Links**, and then select **Add New Event Definition**.
3. In the **Event Definition Configuration** dialog box that opens, create a new custom event definition as shown in the following illustration, and then click **OK**.


Event Definition Configuration

[✓ Create new custom event definition] Create from event definition file

| | | | |
|---|---|---|---|
| Event Definition name: | EventDefinition | Event Definition version: | 1.0 |
| Product name: | SalesProduct | Product instance: | Default |

Information

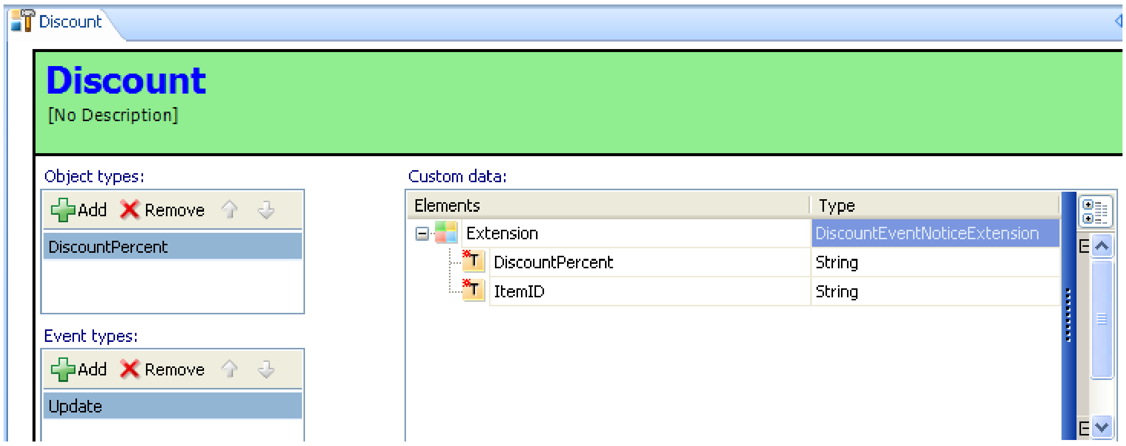
The event definition name uniquely identifies this event definition in this process app. It does not affect mapping of the event to a workflow.

The event definition version, product name and product instance, along with the object type and event type defined later, are used by the Event Manager to identify the orchestration workflows called in response to an event.

The product name is the name of the external product raising the event associated with this event definition. If a product uses more than one event definition, the event definition version and product instance can be used to distinguish between event definitions used by the product.

4. Define values in the event definition editor that opens as shown in the following illustration. See Creating a New Custom Event Definition [page 69] for more information.

   The **Extension** element is automatically added to the **Custom data**. Its **Type** is the event definition name with **EventNoticeExtension** appended to it.


Discount

Discount
[No Description]

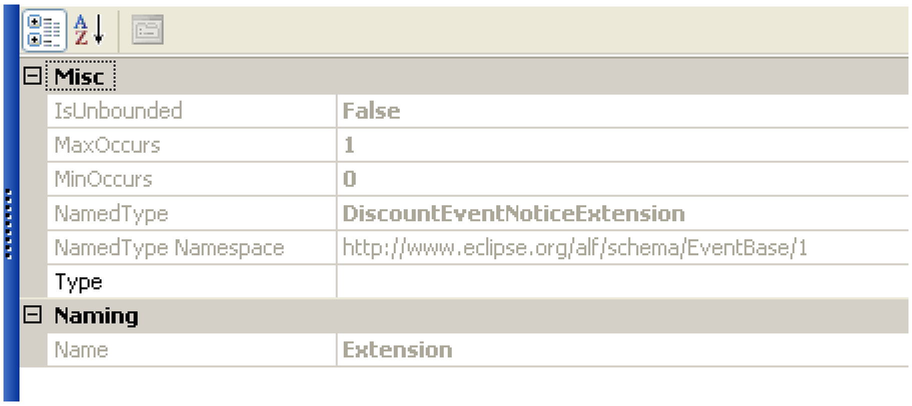
Object types:
DiscountPercent

Event types:
Update

Custom data:

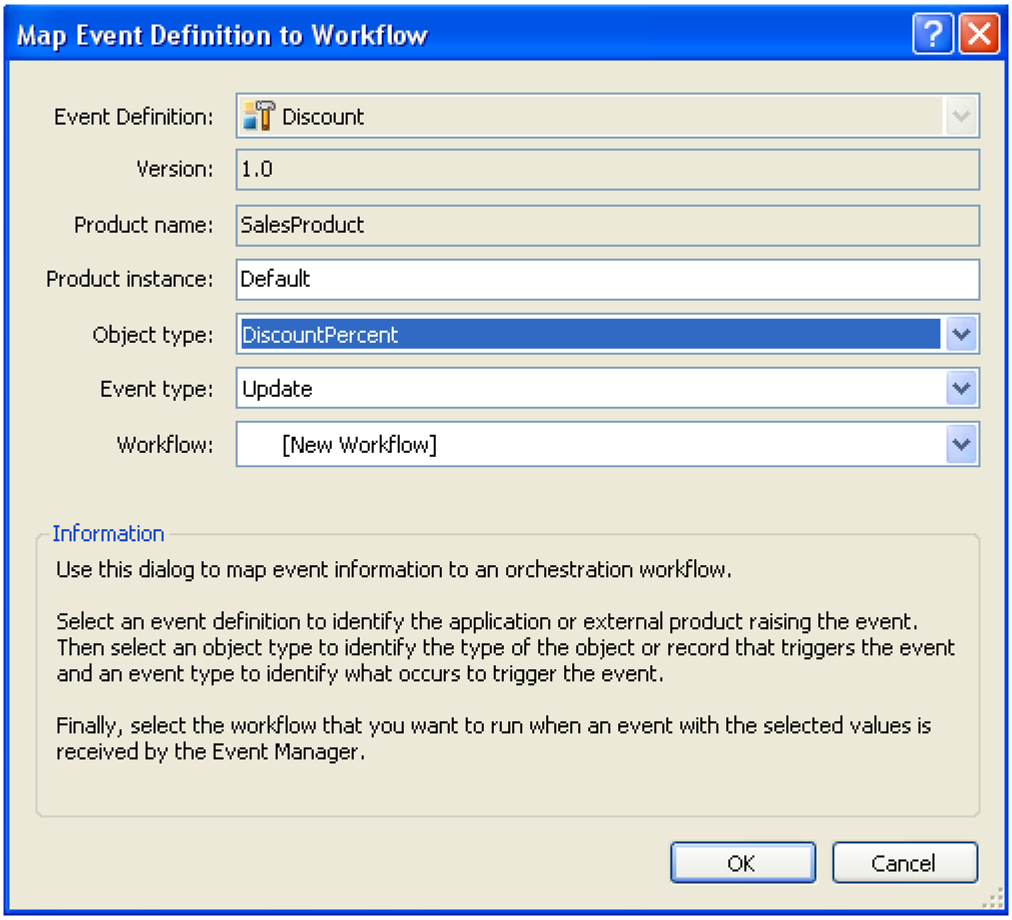
| Elements | Type |
|---|---|
| Extension | DiscountEventNoticeExtension |
| DiscountPercent | String |
| ItemID | String |

The properties for the **Extension** element include its named type and namespace.



| | |
|---|---|
| **Misc** | |
| IsUnbounded | False |
| MaxOccurs | 1 |
| MinOccurs | 0 |
| NamedType | DiscountEventNoticeExtension |
| NamedType Namespace | http://www.eclipse.org/alf/schema/EventBase/1 |
| Type | |
| **Naming** | |
| Name | Extension |

5. On the **Event Map** tab of the event definition Property Editor, click **Add**.
6. In the **Map Event Definition to Workflow** dialog box that opens, select **[New Workflow]** and then click **OK**.



Map Event Definition to Workflow

| | |
|---|---|
| Event Definition: | Discount |
| Version: | 1.0 |
| Product name: | SalesProduct |
| Product instance: | Default |
| Object type: | DiscountPercent |
| Event type: | Update |
| Workflow: | [New Workflow] |

Information

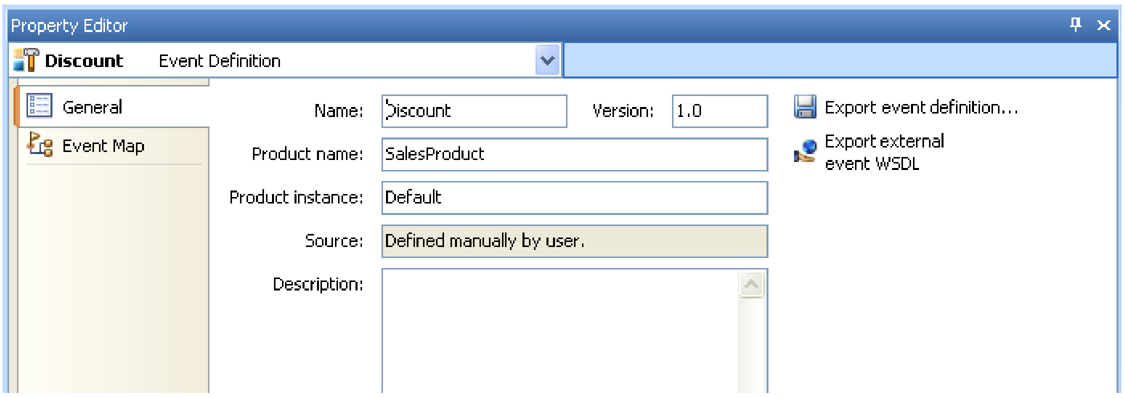
Use this dialog to map event information to an orchestration workflow.

Select an event definition to identify the application or external product raising the event. Then select an object type to identify the type of the object or record that triggers the event and an event type to identify what occurs to trigger the event.

Finally, select the workflow that you want to run when an event with the selected values is received by the Event Manager.

A new orchestration workflow is added to App Explorer.

7. On the **General** tab of the event definition Property Editor, click **Export external event WSDL**, and save the `.wsdl` file to the file system of your computer.


Property Editor

**Discount** Event Definition

General

Event Map

Name: Discount

Version: 1.0

Product name: SalesProduct

Product instance: Default

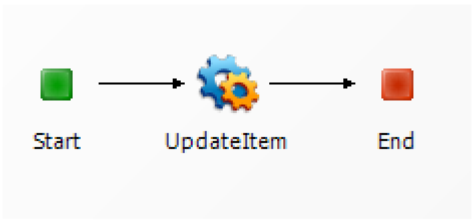
Source: Defined manually by user.

Description:

Export event definition...

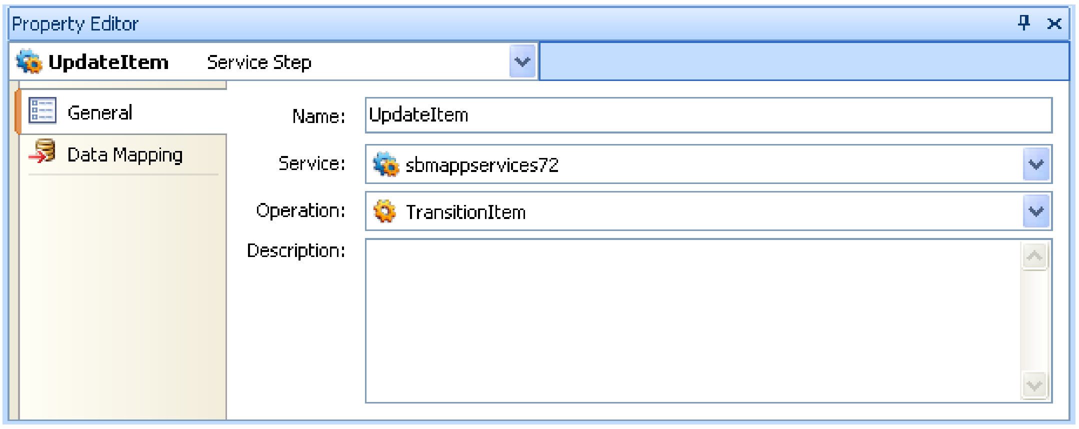
Export external event WSDL

8. Click the orchestration workflow that was added to App Explorer and add a **Service** step to it as shown below. This is the orchestration workflow that will be invoked when the external event is raised.



9. Configure the **UpdateItem** step to update the discount amount:

   a. On the **General** tab of the step Property Editor, select the Web service and operation shown in the following illustration.


Property Editor

**UpdateItem** Service Step

General

Data Mapping

Name: UpdateItem

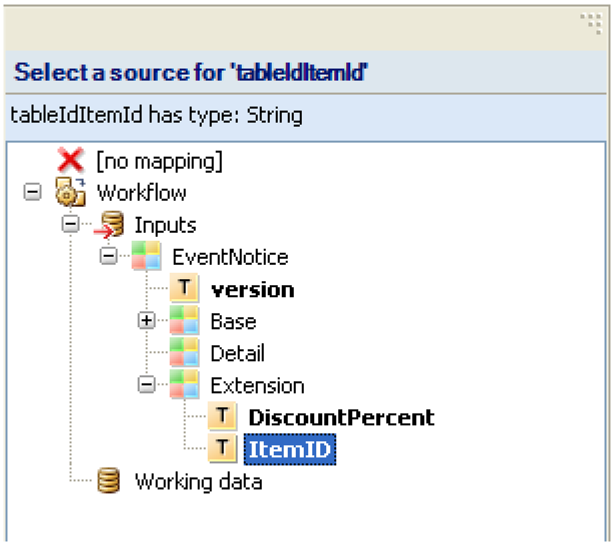
Service: sbmappservices72

Operation: TransitionItem

Description:

   b. Click the **Data Mapping** tab of the step Property Editor.

   c. Expand **id** under **item**, click in the **Source elements** column for the **tableIdItemId** step input, and then select **ItemID** under **Extension** in the **Select a source** popup that opens. (This is one of the custom data fields you added to the custom event definition.)

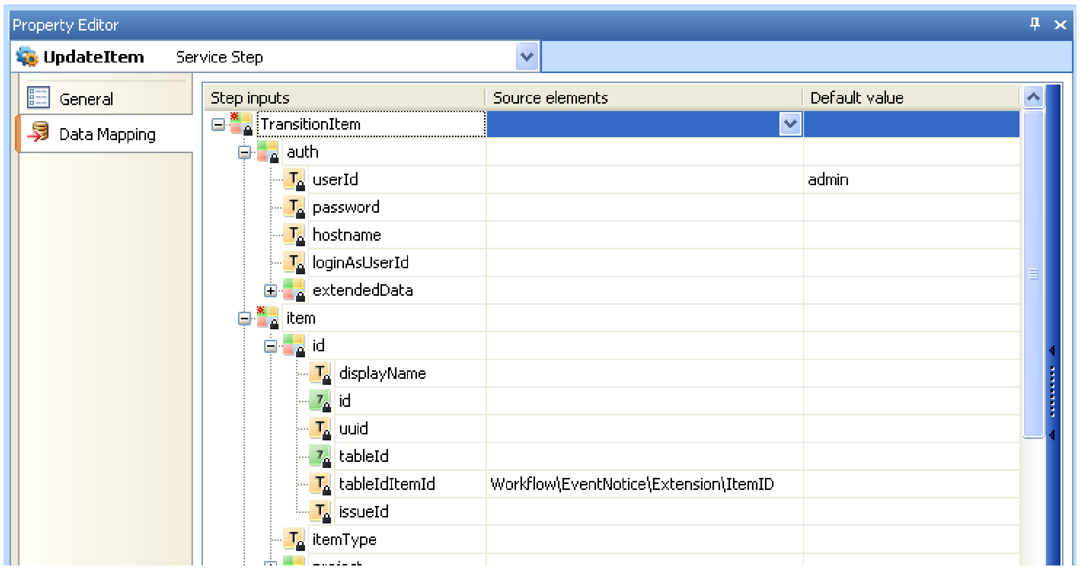


Select a source for 'tableIdItemId'

tableIdItemId has type: String

- [no mapping]
- Workflow
  - Inputs
    - EventNotice
      - version
      - Base
      - Detail
      - Extension
        - DiscountPercent
        - ItemID
  - Working data

d. Click **OK.** The path is added to the **Source elements** column.



Property Editor

UpdateItem    Service Step

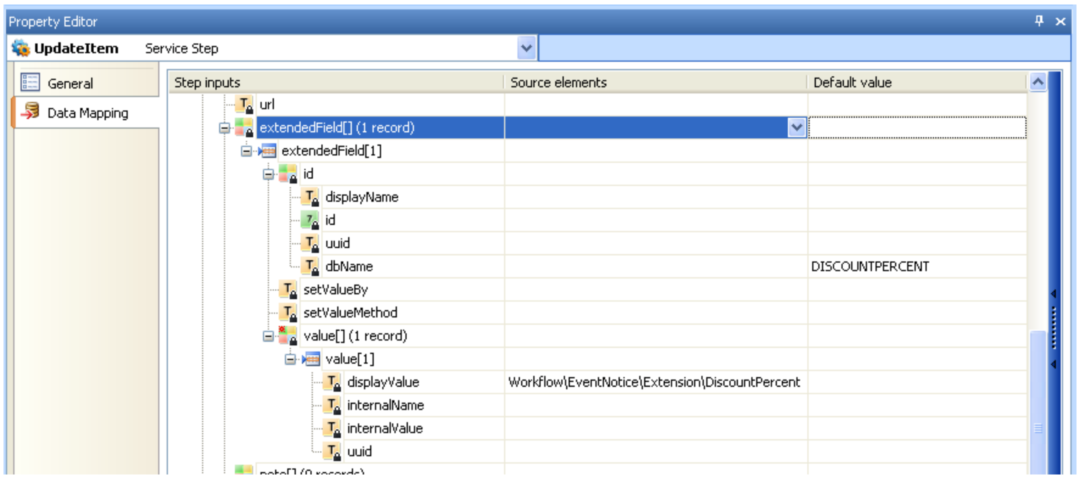
General

Data Mapping

| Step inputs | Source elements | Default value |
|---|---|---|
| TransitionItem | | |
| auth | | |
| userId | | admin |
| password | | |
| hostname | | |
| loginAsUserId | | |
| extendedData | | |
| item | | |
| id | | |
| displayName | | |
| id | | |
| uuid | | |
| tableId | | |
| tableIdItemId | Workflow\EventNotice\Extension\ItemID | |
| issueId | | |
| itemType | | |

e. Right-click the **extendedField[]** step input, and then select **Add Array Record**. Expand **extendedField[1]**, and then type DISCOUNTPERCENT in the **Default value** column for the **dbName** step input. This is the database name of the field that will be updated.

f. Expand the **value[]** step input, right-click, and then select **Add Array Record**. Expand **value[1]** and then and then click in the **Source elements** column for the **displayValue** step input. In the **Select a source** popup that opens, select **DiscountPercent** under **Extension**. (This is the other custom data field you added to the custom event definition.) Click **OK.** The path is added to the **Source elements** column.



Property Editor

**UpdateItem** Service Step

General

Data Mapping

| Step inputs | Source elements | Default value |
|---|---|---|
| url | | |
| extendedField[] (1 record) | | |
| extendedField[1] | | |
| id | | |
| displayName | | |
| id | | |
| uuid | | |
| dbName | | DISCOUNTPERCENT |
| setValueBy | | |
| setValueMethod | | |
| value[] (1 record) | | |
| value[1] | | |
| displayValue | Workflow\EventNotice\Extension\DiscountPercent | |
| internalName | | |
| internalValue | | |
| uuid | | |

10. Deploy the process app.

11. Use the exported `.wsdl` file to create a SOAP request to raise an external event. You can use any third-party product that can create and send SOAP requests to do this. If you are an advanced user, you can alternatively create the SOAP request manually and then send it in an e-mail message. For more information, see Raising an External Event through E-mail [page 228].

## Executing a Post Transition Through a Web Service

A "Post" transition submits a new item from the current item into another project. This use case shows how to use a Web service in an orchestration workflow to execute a Post transition without any user intervention. In this use case, a new item is automatically submitted into a Product Management project after an enhancement item is submitted into a Development project. The **Submit** transition has an action that forces the workflow to immediately execute the **Enh Post** transition. The **Enh Post** transition has an action that invokes the orchestration workflow.

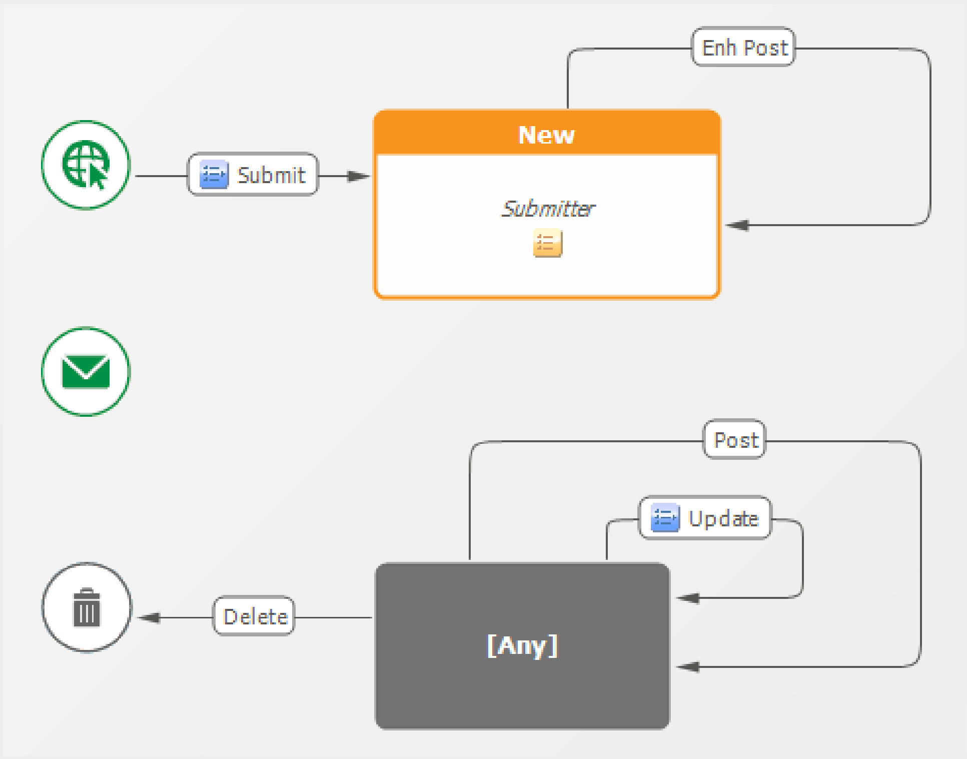
**Prerequisites:**

The following must be set up before you perform this procedure:

- Application process app named `Product Mgt`. (For this tutorial, it is unnecessary to add states or transitions to the application workflow for this process app.)
- Process app named `Development`.
  - Type `EnhApp` as the application name for this process app.
  - Type `EnhApp` as the application workflow name for this process app and configure it as shown in the following illustration.

**Note:**

- The **Post** transition is a "Post" transition. The **Submit** and **Enh Post** transitions are "Regular" transitions. (For information about transition types, see the SBM Composer documentation.)
- On the **Options** tab of the Property Editor for the **Enh Post** and **Post** transitions, select the "Quick transition" and "Hide transition button on state form" options.



**To execute a Post transition through a Web service:**

1. Add a transition action to the **Submit** transition:
   a. Select the **Submit** transition.

b. Click the **Actions** tab in the transition Property Editor.

c. Click **New**.

d. In the Action Wizard that opens, select **Transition** and then click **Next**.

e. Make sure **This item** is selected and then click **Next**.

f. Make sure **Unconditionally** is selected and then click **Next**.

g. Select the **Enh Post** transition and then click **Finish**.

2. Add an orchestration action to the **Enh Post** transition:

   a. Select the **Enh Post** transition.

   b. On the **Actions** tab of the transition Property Editor, click **New**.

   c. In the Action Wizard that opens, make sure that **Orchestration Workflow** is selected as the action type, that **and continue executing (asynchronous) using the local event** is in the rule description, and then click **Next**.

   d. Make sure that **This item** is selected as the affected item, and then click **Next**.

   e. Make sure that **Unconditionally** is selected as the condition, and then click **Next**.

   f. Select **New orchestration**. In the **New Orchestration** dialog box, type **EnhPost** and then click **OK**. The orchestration and an orchestration workflow are added to App Explorer.

   g. Make sure the orchestration workflow is selected, and then click **Finish**. An event definition is added to App Explorer under the **Application Links** heading.

3. Specify the fields that will be sent to the orchestration workflow:

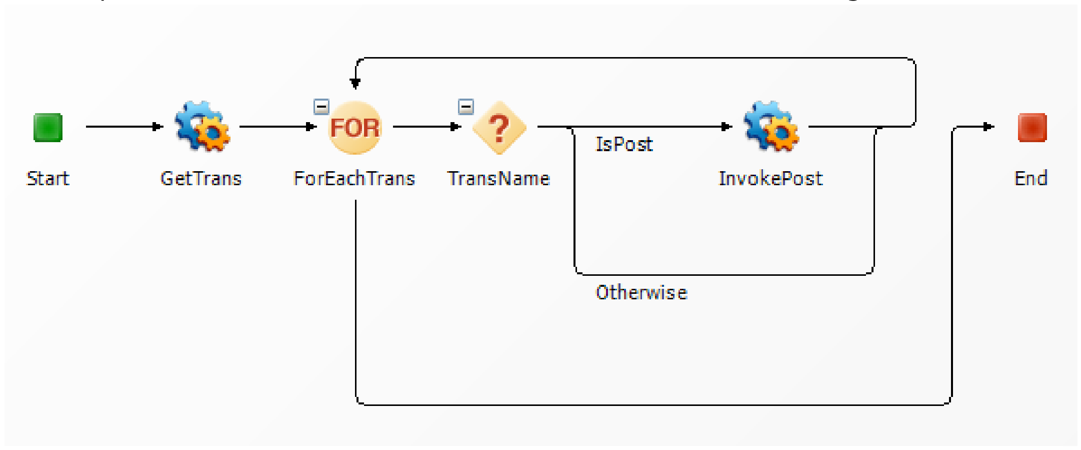
   a. Click **Event without Reply** under the **Orchestration Links** heading in App Explorer.

   b. In the **Event without Reply** dialog box that opens, clear all fields except *Item Id*, *Item Type*, and *Title*. If you click **EnhAppEventDefinition** in App Explorer under the **Application Links** heading, the *ItemId*, *ItemId_TableRecId*, *ItemId_TableId*, *ItemType*, and *Title* fields are listed in the **Custom data** section.

4. Configure the orchestration workflow:

   a. Click **NewEnhPostWorkflow** in App Explorer under the **Orchestration Workflows** heading.

b. Add steps to the orchestration workflow as shown in the following illustration:



c. Click an empty area of the workflow.

d. Click the **Data Mapping** tab in the workflow Property Editor.

e. Right-click **Working data**, select **Add New**, and then select **String**.

f. Change the name of the new data element (**String**) to `TableItemId`.

g. Click the down arrow in the **Source elements** column.

h. In the **Select a source** popup that opens, expand **NewOrchPostWorkflow**, **Inputs**, **EventNotice**, and **Extension**; select **ItemId_TableRecId**, and then click **OK**.

i. On the **General** tab of the **GetTrans** step Property Editor, select **sbmappservices72** in the **Service** box. In the **Operation** box, select **GetAvailableTransitions**.

j. On the **Data Mapping** tab of the **GetTrans** step Property Editor, expand the **auth** input and then type a valid user ID and password in the **Default value** column.

k. Expand the **item** input, and then click the down arrow in the **Source elements** column for the **tableIdItemId** row.

l. In the Select a source popup that opens, select **TableItemId** under **Working data**, and then click **OK**.

m. Enter `GetTrans.GetAvailableTransitionsResponse.return` on the **Options** tab of the **ForEachTrans** step Property Editor.

n. Type `IsPost` on the **General** tab of the branch Property Editor for the top branch from the **TransName** step.

o. Enter `ForEachTrans.item.transition.displayName = 'Post'` on the **Options** tab of the **IsPost** branch Property Editor.

p. On the **General** tab of the **InvokePost** step Property Editor, select **sbmappservices72** from the **Service** list. Select **TransitionItem** from the **Operation** list.

q. On the **Data Mapping** tab of the **InvokePost** step Property Editor, expand the **auth** input and then type a valid user ID and password in the **Default value** column.

r. Expand the **item** and **id** inputs, click the down arrow in the **Source elements** column for the **tableIdItemId** row, and select **tableIdItemId** under **Working data**.

s. Scroll down to the **transition** step input, expand it, and then click the down arrow in the **Source elements** column of the **id** row.

t. In the **Select a source** popup that opens, expand **ForEachTrans**, expand **Outputs**, **item**, and **transition**; select **id**, and then click **OK**.

u. Select **False** in the **Default value** column of the **breakLock** row.

5. Configure the **Post** transition:

   a. Right-click the **References** heading in App Explorer, and select **Add Application Reference**.

   b. In the **Add Application Reference** dialog box that opens, select the application associated with the Product Mgt process app and then click **Add**.

   c. In the application workflow, select the **Post** transition.

   d. Click the **Post Options** tab, and select the Product Mgt application and table.

6. Open and then deploy the Product Mgt process app.

7. Open and then deploy the Development process app.

8. In SBM Application Administrator, specify the project to which to post items.

9. Test the process app.

   a. In SBM Work Center, submit an item into the Development project.

   b. Type a title in the Submit transition form, and then click **OK**.

   c. Perform a search for the item. The posted item is displayed in the search results, with a new item ID.

## Executing a Copy Transition Through a Web Service

A "Copy" transition lets you copy primary items and place them in another location in the project hierarchy within the same table. This use case shows how to use a Web service in an orchestration workflow to execute a Copy transition without any user intervention. In this use case, a copy of an item is automatically submitted into a Documentation project after an enhancement item is submitted into a Development project. The **Submit** transition has an action that forces the workflow to immediately execute the **Enh Copy** transition. The **Enh Copy** transition has an action that invokes the orchestration workflow.

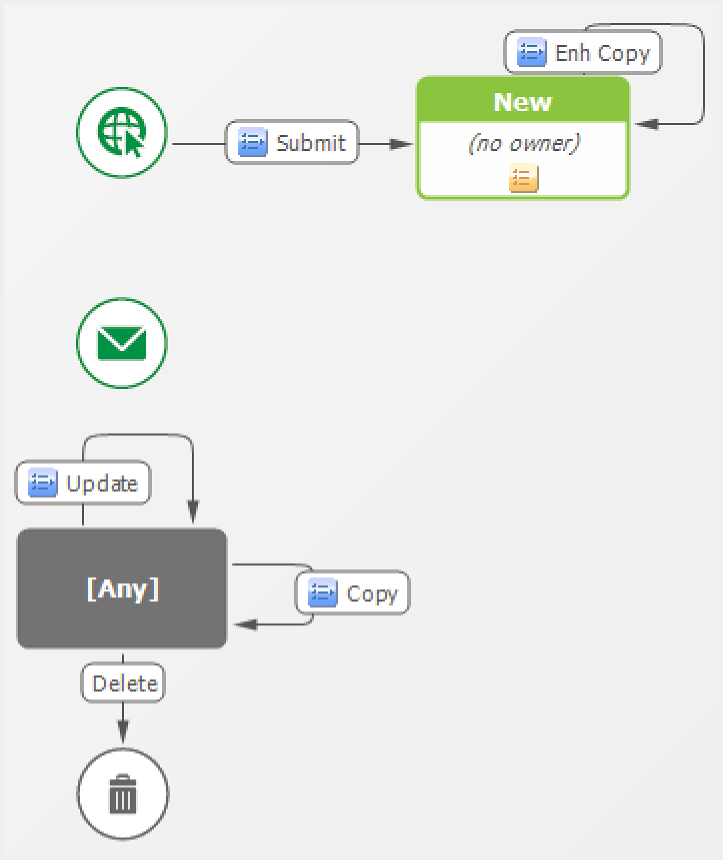
**Prerequisites:**

The following must be set up before you perform this procedure:

- Application process app named `Develop`.
- Application in that process app named `DevApp`.
- Application workflow in that process app named `DevAppWorkflow`, configured as shown in the following illustration.

  **Note:**

  - The **Copy** transition is a "Copy" transition. The **Submit** and **Enh Copy** transitions are "Regular" transitions. (For information about transition types, see the SBM Composer documentation.)
  - On the **Options** tab of the Property Editor for the **Enh Copy** and **Copy** transitions, select the "Quick transition" and "Hide transition button on state form" options.



**To execute a Copy transition through a Web service:**

1. Add a transition action to the **Submit** transition:
   a. Select the **Submit** transition.

b. On the **Actions** tab of the transition Property Editor, click **New**.

c. In the Action Wizard that opens, select **Transition** and then click **Next**.

d. Make sure **This item** is selected and then click **Next**.

e. Make sure **Unconditionally** is selected and then click **Next**.

f. Select the **Enh Copy** transition and then click **Finish**.

2. Add an orchestration action to the **Enh Copy** transition:

   a. Select the **Enh Copy** transition.

   b. On the **Actions** tab of the transition Property Editor, click **New**.

   c. In the Action Wizard that opens, make sure that **Orchestration Workflow** is selected as the action type, that **and continue executing (asynchronous) using the local event** is in the rule description, and then click the **Next** button.

   d. Make sure that **This item** is selected as the affected item, and then click the **Next** button.

   e. Make sure that **Unconditionally** is selected as the condition, and then click the **Next** button.

   f. Select **New orchestration**, and type **EnhCopy** in the **New Orchestration** dialog box. The orchestration and an orchestration workflow are added to App Explorer.

   g. Make sure the new orchestration workflow is selected, and then click **Finish**. An event definition is added to App Explorer under the **Application Links** heading.

3. Specify the fields that will be sent to the orchestration workflow:

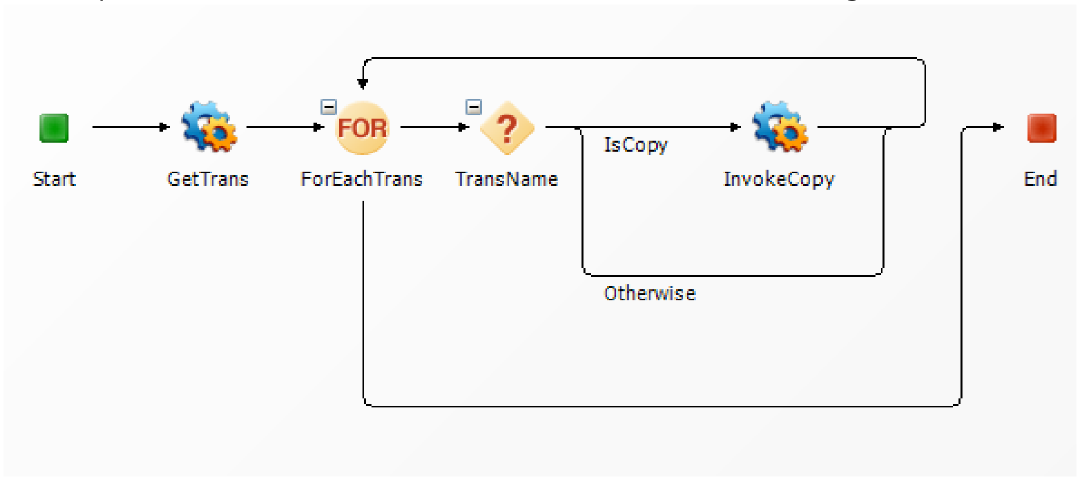
   a. Click **Event without Reply** under the **Orchestration Links** heading in App Explorer.

   b. In the **Event without Reply** dialog box that opens, clear all fields except *Item Id*, *Item Type*, and *Title*. If you click **EnhAppEventDefinition** in App Explorer under the **Application Links** heading, the *ItemId*, *ItemId_TableRecId*, *ItemId_TableId*, *ItemType*, and *Title* fields are listed in the **Custom data** section.

4. Configure the orchestration workflow:

   a. Click **NewEnhCopyWorkflow** in App Explorer under the **Orchestration Workflows** heading.

b. Add steps to the orchestration workflow as shown in the following illustration:



c. Click an empty area of the workflow.

d. Click the **Data Mapping** tab in the workflow Property Editor.

e. Right-click **Working Data**, select **Add New**, and then select **String**.

f. Change the name of the new data element (**String**) to `TableItemId`.

g. Click the down arrow in the **Source elements** column.

h. In the **Select a source** popup that opens, expand **NewOrchCopyWorkflow**, **Inputs**, **EventNotice**, and **Extension**; select **ItemId_TableRecId**, and then click the **OK** button.

i. On the **General** tab of the **GetTrans** step Property Editor, select **sbmappservices72** from the **Service** list. Select **GetAvailableTransitions** from the **Operation** list.

j. On the **Data Mapping** tab of the **GetTrans** step Property Editor, expand the **auth** input and then type a valid user ID and password in the **Default value** column.

k. Expand the **item** input, click the down arrow in the **Source elements** column for the **TableItemId** row, and select **tableIdItemId** under **Working data**.

l. Enter `GetTrans\GetAvailableTransitionsResponse\return` on the **Options** tab of the **ForEachTrans** step Property Editor.

m. Type `IsCopy` on the **General** tab of the branch Property Editor for the top branch from the **TransName** step.

n. Enter `ForEachTrans\item\transition\displayName = 'Copy'` on the **Options** tab of the **IsCopy** branch Property Editor.

o. On the **General** tab of the **InvokeCopy** step Property Editor, select **sbmappservices72** from the **Service** list. Select **TransitionItem** from the **Operation** list.

p. On the **Data Mapping** tab of the **InvokeCopy** step Property Editor, expand the **auth** input and then type a valid user ID and password in the **Default value** column.

q. Expand the **item** and **id** inputs, and then click the down arrow in the **Source elements** column for the **tableIdItemId** row.

r. In the **Select a source** popup that opens, select **TableItemId** under **Working data**.

s. Scroll down to the **transition** step input and expand it.

t. Click the down arrow in the **Source elements** column of the **id** row.

u. In the **Select a source** popup that opens, expand **ForEachTrans**, expand **Outputs**, **item**, and **transition**; select **id**, and then the click **OK** button.

v. Select **False** in the **Default value** column of the **breakLock** row.

5. Deploy the Development process app.

6. In SBM Application Administrator, specify the project to which to post items.

7. Test the process app.

   a. In SBM Work Center, submit an item into the Develop Project project.

   b. Type a title in the Submit transition form, and then click the **OK** button.

   c. Perform a search for the item. The copied item is displayed in the search results, with a new item ID.

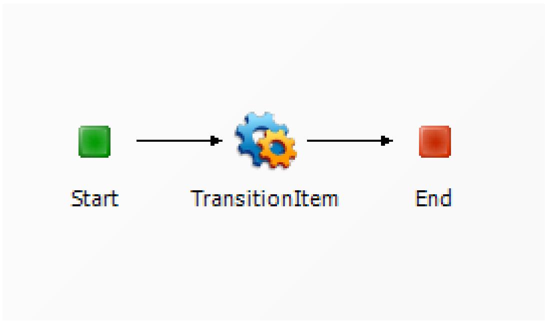
## Sending Multiple Values in an Event

In the following use cases, a *Multi-Selection* field called "Testers" is populated with values specified in orchestration workflow steps.

### Sending Values Asynchronously

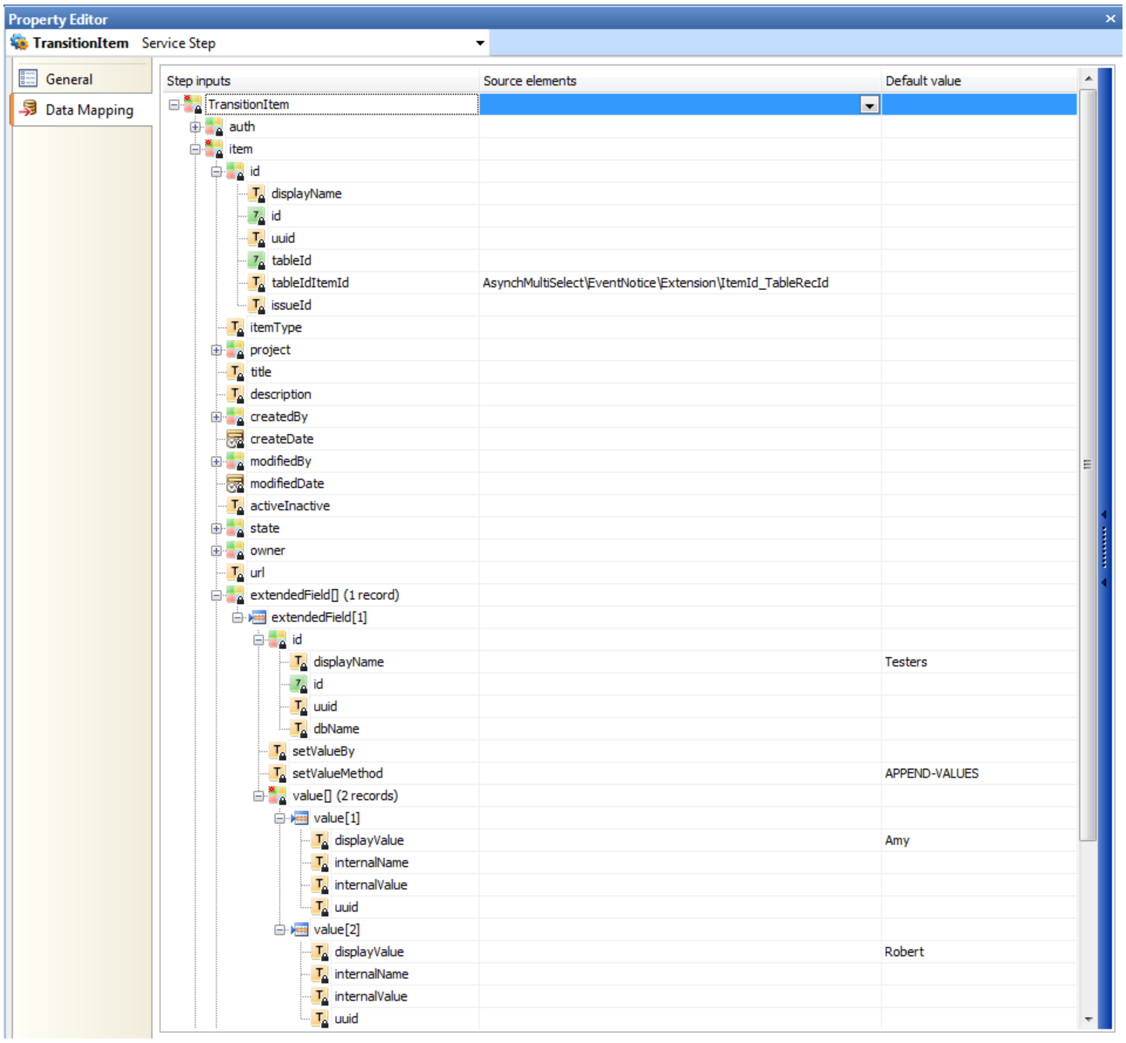
In this use case, the *Testers* field is updated both with values a user selects on a form and values specified in a **Service** step in an asynchronous orchestration workflow. The *Testers* field values are sent in the application link for the orchestration workflow.

1. Add a **Service** step with the **TransitionItem** operation to the orchestration workflow as shown below.



2. Configure the **TransitionItem** step to add "Amy" and "Robert" to the *Testers* field.

   **Note:** This step uses the `APPEND-VALUES` parameter of the `Set-Value-Method`. For more information, see the *SBM Web Services Developer's Guide*.



Property Editor

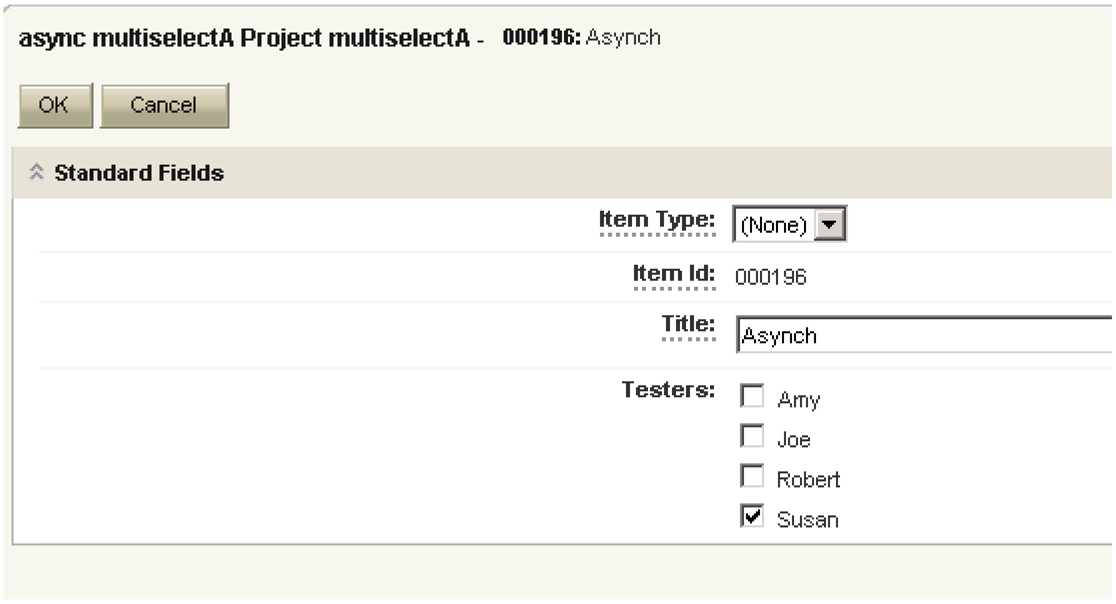
TransitionItem Service Step

General

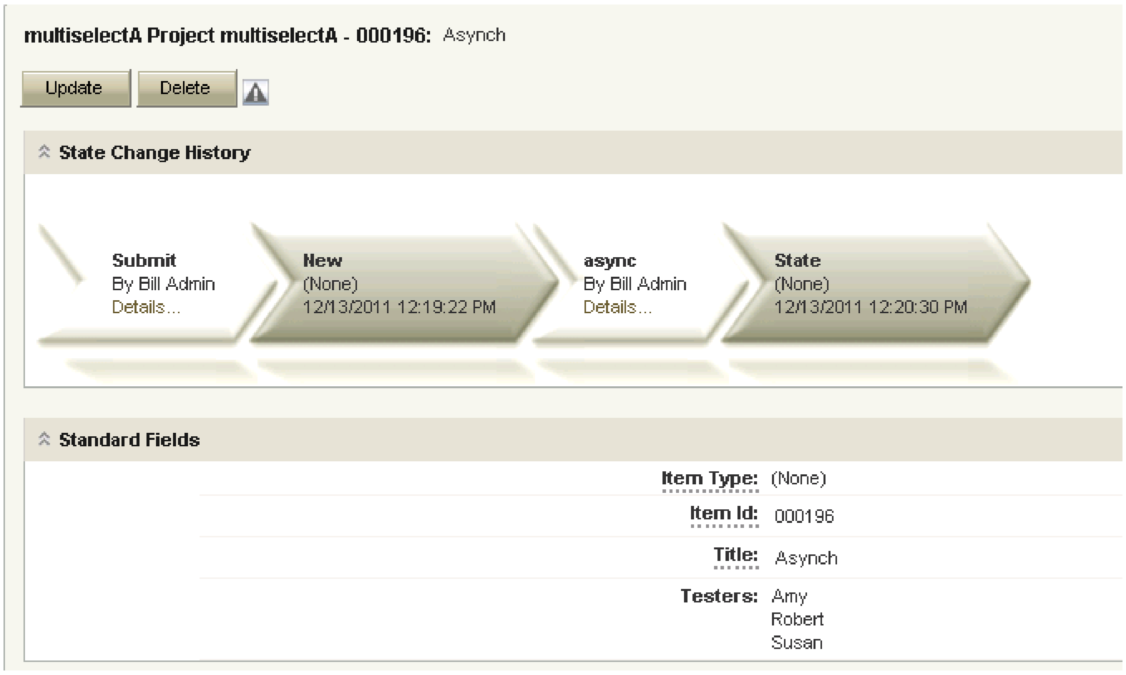
Data Mapping

| Step inputs | Source elements | Default value |
|---|---|---|
| TransitionItem | | |
| auth | | |
| item | | |
| id | | |
| displayName | | |
| id | | |
| uuid | | |
| tableId | | |
| tableIdItemId | AsynchMultiSelect\EventNotice\Extension\ItemId_TableRecId | |
| issueId | | |
| itemType | | |
| project | | |
| title | | |
| description | | |
| createdBy | | |
| createDate | | |
| modifiedBy | | |
| modifiedDate | | |
| activeInactive | | |
| state | | |
| owner | | |
| url | | |
| extendedField[] (1 record) | | |
| extendedField[1] | | |
| id | | |
| displayName | | Testers |
| id | | |
| uuid | | |
| dbName | | |
| setValueBy | | |
| setValueMethod | | APPEND-VALUES |
| value[] (2 records) | | |
| value[1] | | |
| displayValue | | Amy |
| internalName | | |
| internalValue | | |
| uuid | | |
| value[2] | | |
| displayValue | | Robert |
| internalName | | |
| internalValue | | |
| uuid | | |

After the user selects "Susan" and transitions the item, the *Testers* field contains "Amy," "Robert," and "Susan."


async multiselectA Project multiselectA - **000196:** Asynch

Item Type: (None)
Item Id: 000196
Title: Asynch
Testers: [ ] Amy [ ] Joe [ ] Robert [✓ Susan]


multiselectA Project multiselectA - 000196: Asynch

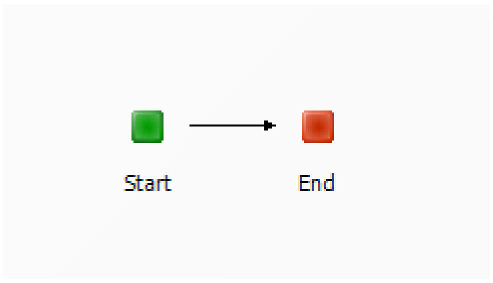
Submit By Bill Admin Details...
New (None) 12/13/2011 12:19:22 PM
async By Bill Admin Details...
State (None) 12/13/2011 12:20:30 PM

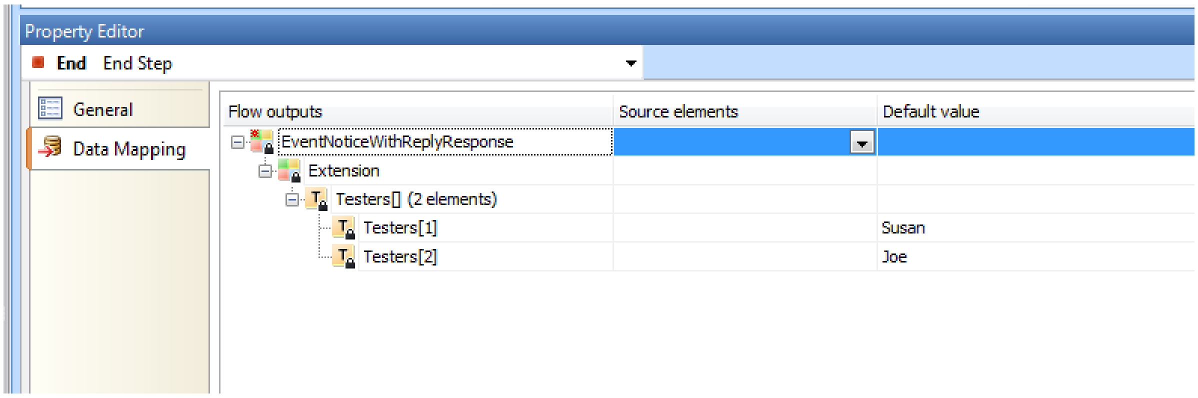
Item Type: (None)
Item Id: 000196
Title: Asynch
Testers: Amy
Robert
Susan

## Sending Values Synchronously

In this use case, the values a user selects on a form for the *Testers* field are overwritten by values specified in the **End** step in a synchronous orchestration workflow. The *Testers* field values are sent in the orchestration link for the orchestration workflow.

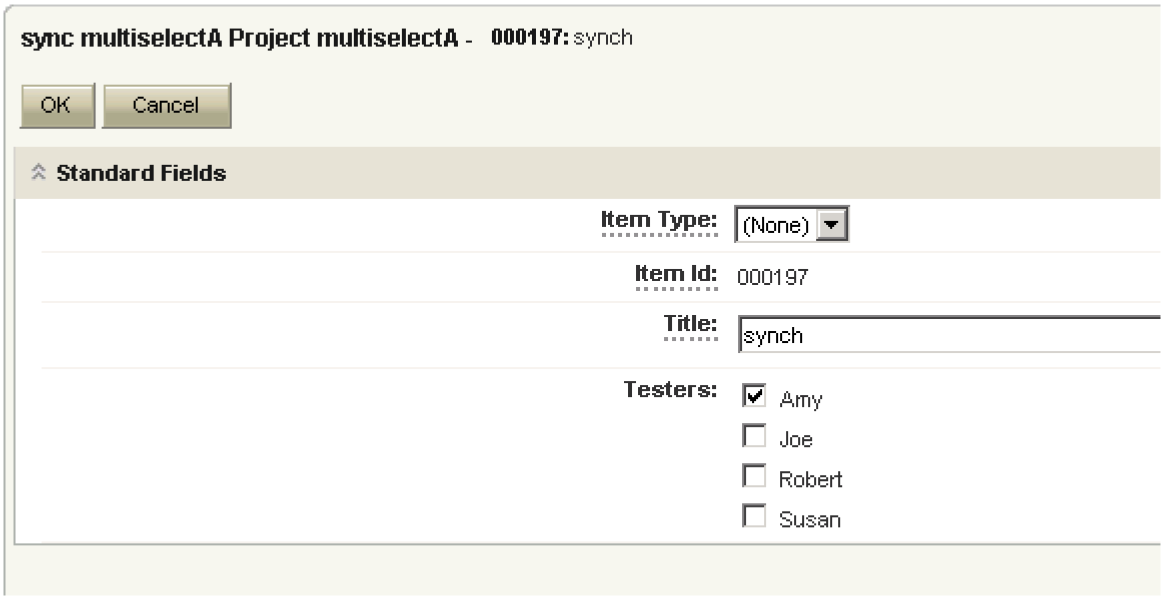
Configure the **End** step to update the *Testers* field to contain only "Susan" and "Joe."





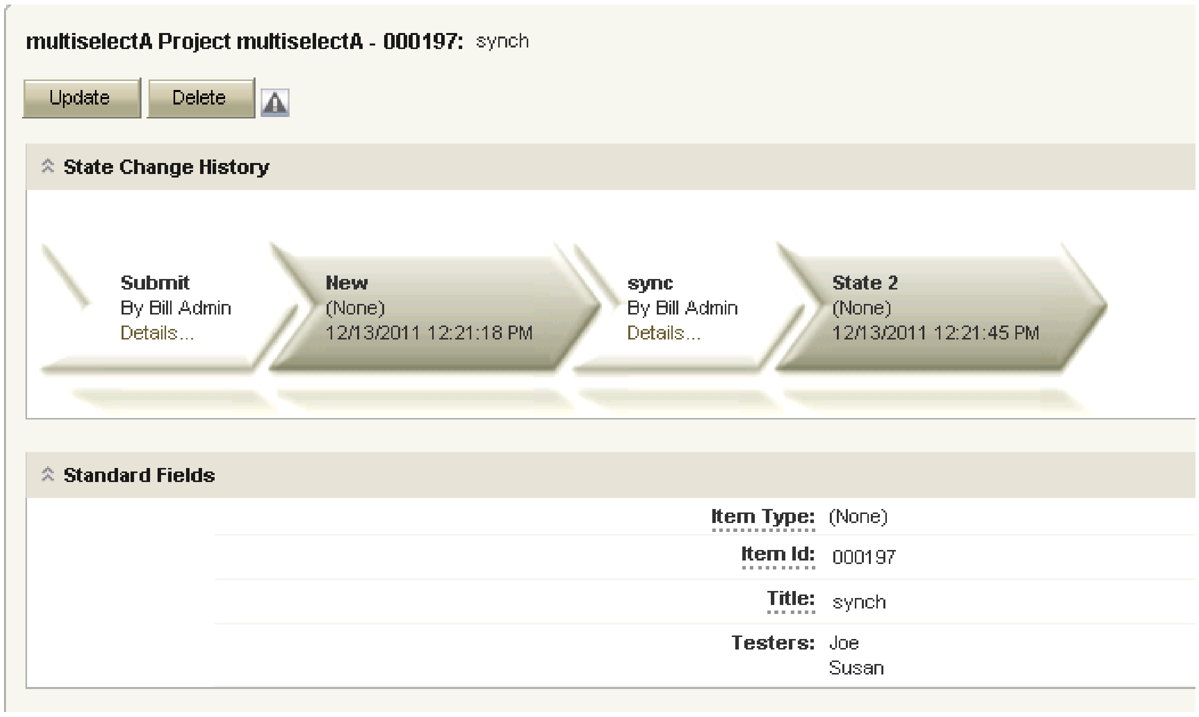
| Flow outputs | Source elements | Default value |
|---|---|---|
| EventNoticeWithReplyResponse | | |
| Extension | | |
| Testers[] (2 elements) | | |
| Testers[1] | | Susan |
| Testers[2] | | Joe |

After the user selects "Amy" and transitions the item, the *Testers* field contains only "Joe" and "Susan."



**sync multiselectA Project multiselectA - 000197:** synch

| | |
|---|---|
| **Item Type:** | (None) |
| **Item Id:** | 000197 |
| **Title:** | synch |
| **Testers:** | ☑ Amy<br>☐ Joe<br>☐ Robert<br>☐ Susan |


multiselectA Project multiselectA - 000197: synch

Submit
By Bill Admin
Details...

New
(None)
12/13/2011 12:21:18 PM

sync
By Bill Admin
Details...

State 2
(None)
12/13/2011 12:21:45 PM

| | |
|---|---|
| Item Type: | (None) |
| Item Id: | 000197 |
| Title: | synch |
| Testers: | Joe<br>Susan |

## Use Case: Updating Subtask Items

In this use case, you get the subtasks from an item and then automatically update fields in the subtasks when those fields are changed in the principal item.

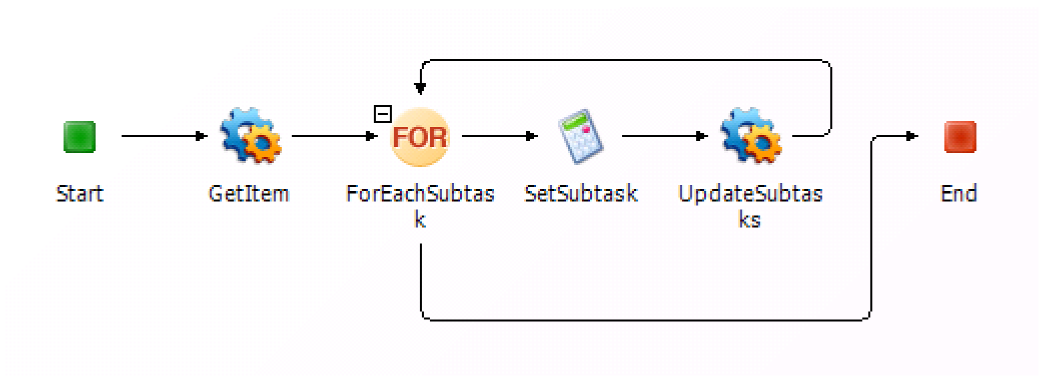
**Note:** For information about creating subtasks, see the "Defining Subtask-Driven Actions" tutorial in the *SBM Composer Guide*.

The orchestration workflow loops through each subtask and processes it as follows:

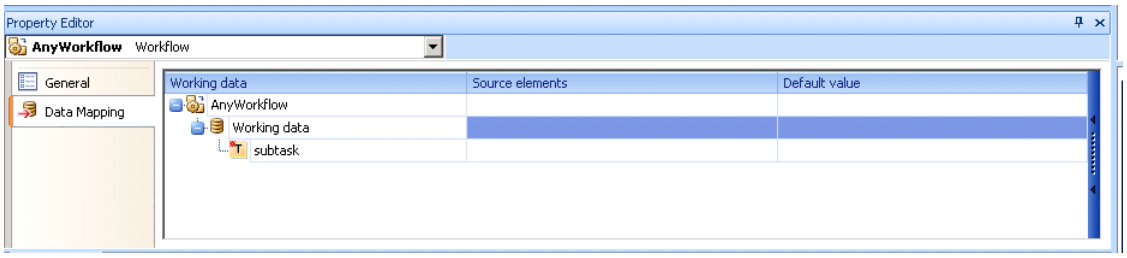
1. Gets the principal item, including its subtasks.
2. Stores the first subtask in a working data variable.
3. Updates the *Regression* and *Info* fields in the subtask with information in the extended field list. The values in the extended field list are mapped to the associated values in the principal item.
4. Repeats step 2 and step 3 for each remaining subtask.

**To update subtask values based on principal item values:**

1. Add steps to the orchestration workflow as shown below.

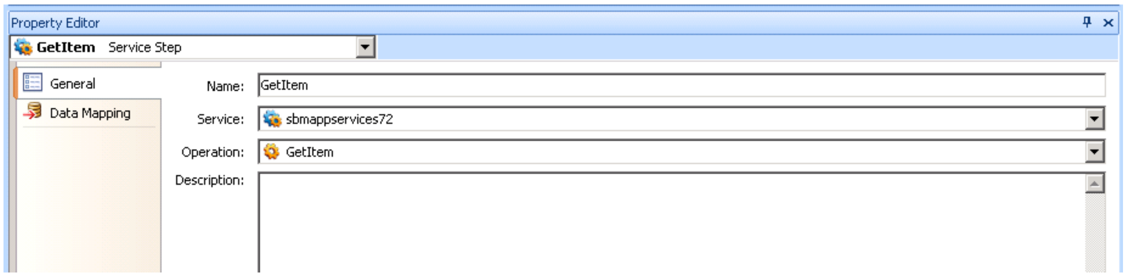


2. Define the working data for the orchestration workflow by adding a variable to store the subtask items.



| Working data | Source elements | Default value |
|---|---|---|
| AnyWorkflow | | |
| Working data | | |
| subtask | | |

3. Configure the **GetItem** step to get the principal item.

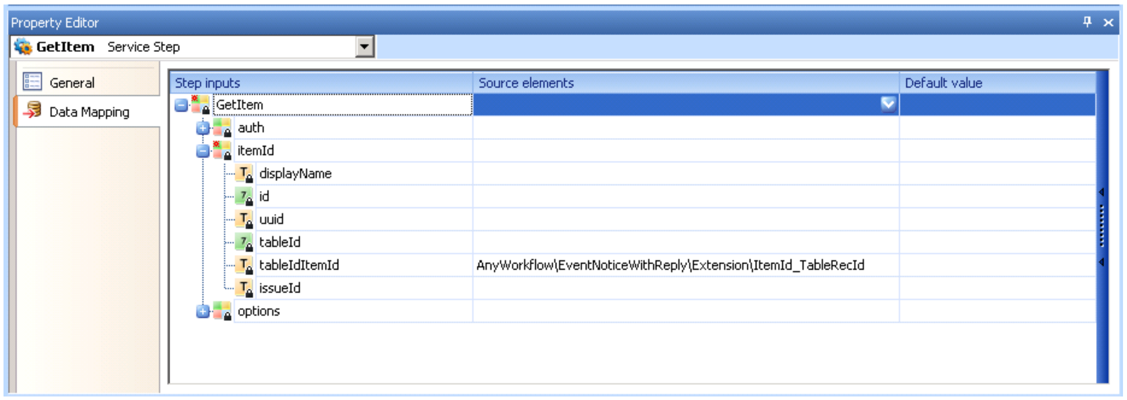


Name: GetItem

Service: sbmappservices72

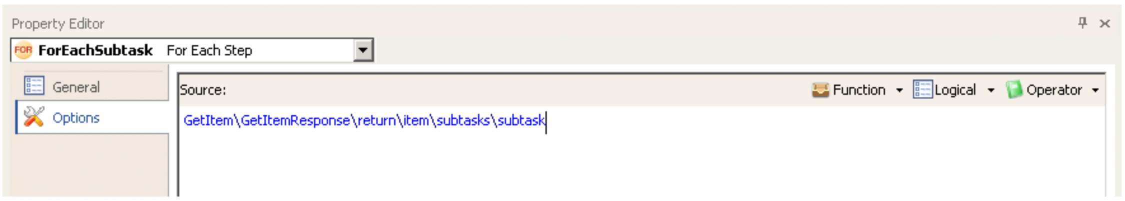
Operation: GetItem

Description:



| Step inputs | Source elements | Default value |
|---|---|---|
| GetItem | | |
| auth | | |
| itemId | | |
| displayName | | |
| id | | |
| uuid | | |
| tableId | | |
| tableIdItemId | AnyWorkflow\EventNoticeWithReply\Extension\ItemId_TableRecId | |
| issueId | | |
| options | | |

4. Configure the **ForEachSubtask** step to get each subtask from the principal item.


Property Editor

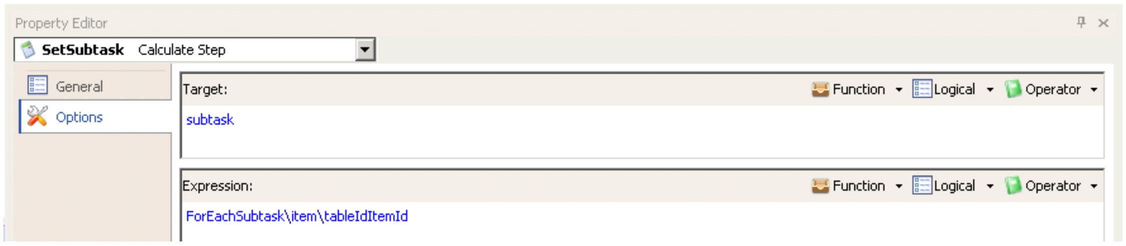
ForEachSubtask For Each Step

General
Options

Source:

GetItem\GetItemResponse\return\item\subtasks\subtask

5. Configure the **SetSubtask** step to store the subtask ID in the working data variable.


Property Editor

SetSubtask Calculate Step

General
Options

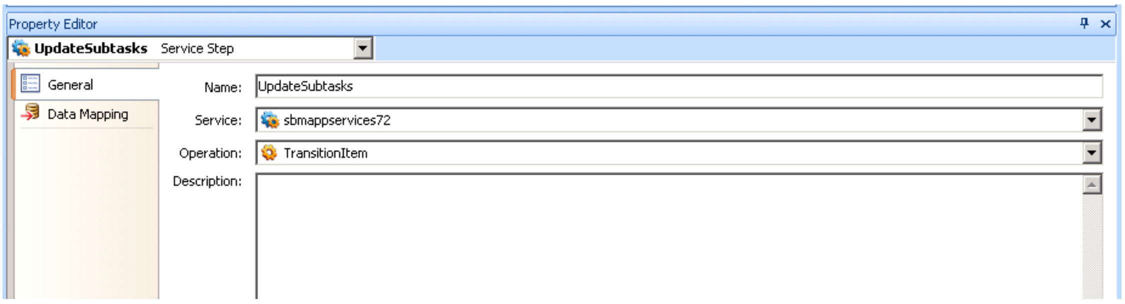
Target:

subtask

Expression:

ForEachSubtask\item\tableIdItemId

6. Configure the **Update Subtasks** step to update each subtask with the extended field values from the principal item.


Property Editor

UpdateSubtasks Service Step

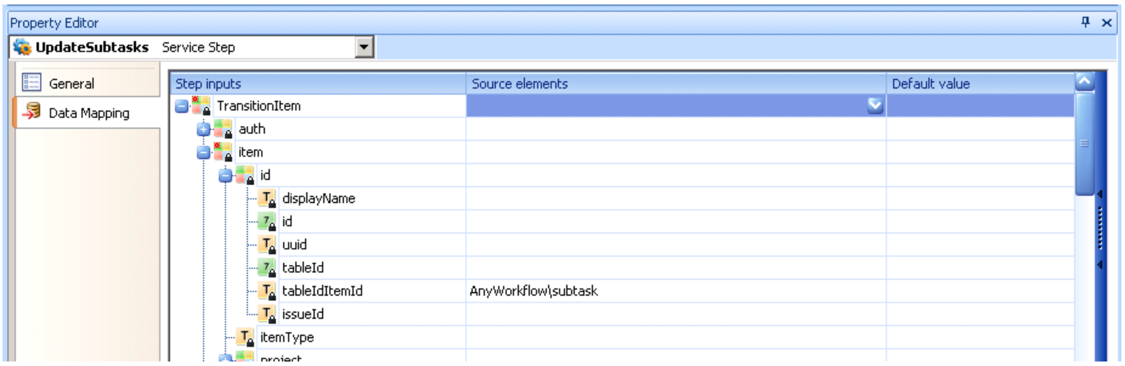
General
Data Mapping

Name: UpdateSubtasks

Service: sbmappservices72

Operation: TransitionItem

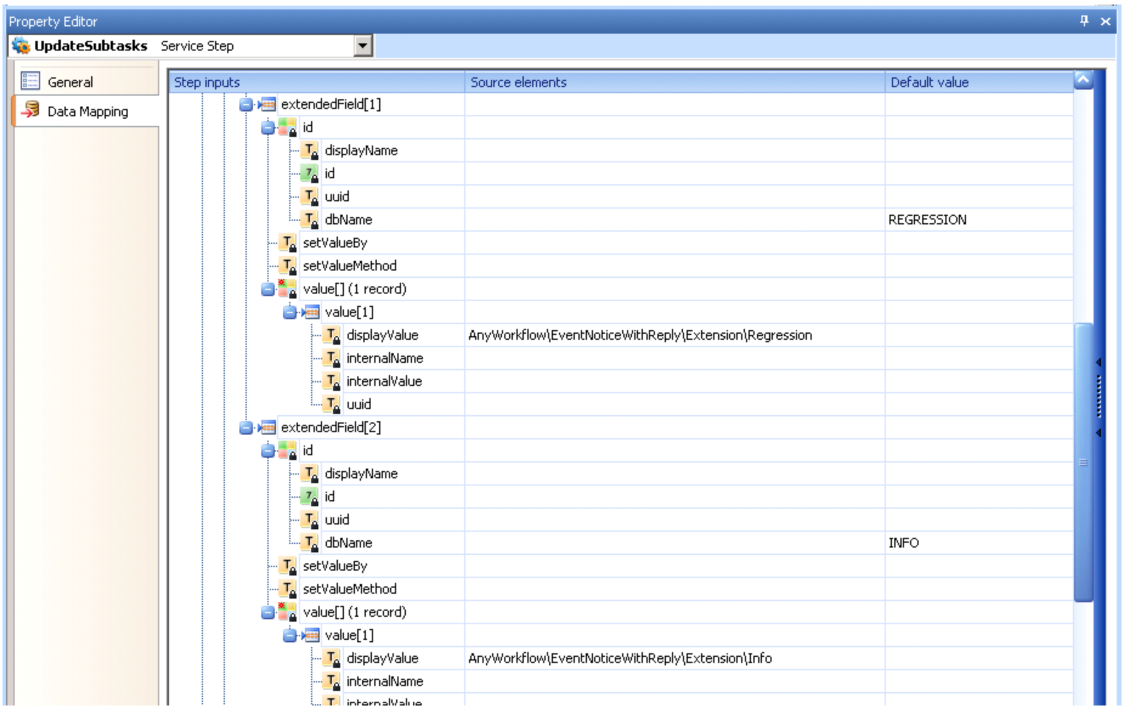
Description:


Property Editor

UpdateSubtasks Service Step

General
Data Mapping

| Step inputs | Source elements | Default value |
|---|---|---|
| TransitionItem | | |
| auth | | |
| item | | |
| id | | |
| displayName | | |
| id | | |
| uuid | | |
| tableId | | |
| tableIdItemId | AnyWorkflow\subtask | |
| issueId | | |
| itemType | | |



| Step inputs | Source elements | Default value |
|---|---|---|
| extendedField[1] | | |
| id | | |
| displayName | | |
| id | | |
| uuid | | |
| dbName | | REGRESSION |
| setValueBy | | |
| setValueMethod | | |
| value[] (1 record) | | |
| value[1] | | |
| displayValue | AnyWorkflow\EventNoticeWithReply\Extension\Regression | |
| internalName | | |
| internalValue | | |
| uuid | | |
| extendedField[2] | | |
| id | | |
| displayName | | |
| id | | |
| uuid | | |
| dbName | | INFO |
| setValueBy | | |
| setValueMethod | | |
| value[] (1 record) | | |
| value[1] | | |
| displayValue | AnyWorkflow\EventNoticeWithReply\Extension\Info | |
| internalName | | |

## Mapping Custom Endpoint Information in a Service Call

Custom endpoints have data that you can map as input for any of the SOAP services you have. SOAP services have their own endpoints that will be resolved individually when you deploy, but if you are calling another service from a service as part of an orchestration, you can use information from custom endpoints as inputs to the calling service so it can use that information to make whatever calls it needs to make to the other service.

To map custom endpoint parameters in your orchestrated SOAP service call:

1. Ensure that the custom endpoint you want to use is in your current orchestration's Custom Endpoint Library.

2. Add a SOAP service call that calls another SOAP service to your orchestration workflow.

3. Click the **Source elements** column beside the called SOAP service call's URL and then under Custom Endpoints, select the custom endpoint URL parameter.

4. Map other custom endpoint parameters to your SOAP service call's parameters as desired, such as user name and password.

You can also map custom endpoint information in RESTCaller, which includes special handling for calling a REST service from a SOAP service. For more information, refer to Chapter 10: Calling RESTful Web Services from an Orchestration Workflow [page 237].

## Using Custom Endpoints with RESTCaller

Using custom endpoints in orchestrations enables you to externalize the location of a REST resource from the orchestration design. If you have defined a custom endpoint to use with your REST service calls, the custom endpoint implementation enables you to access endpoint data and dynamically extend the URL. Following are some ways you can use custom endpoints with RESTCaller:

- Use Case #1: Setting Authentication Based on the Custom Endpoint [page 167]
- Use Case #2: Using Different Servers for Different Environments [page 167]
- Use Case #3: Using the Same Server with Different Resources Using a Resource Path [page 167]
- Use Case #4: Using the Same Server with Different Resources Using Query Strings [page 168]

### Use Case #1: Setting Authentication Based on the Custom Endpoint

You can use the authentication set in a custom endpoint as the authentication for the REST service call that will be made by your RESTCaller step. To configure this, do the following:

1. Use the Custom Endpoint Url field as the resource URL or a base to calculate the resource URL and map the result to the RESTCaller restUrl parameter.
2. Set the RESTCaller authorizationType to ENDPOINT.
3. Map the custom endpoint EndpointID field to httpAuthorization > endpoint > endpointID.

**Note:** For some authorization types you can map the custom endpoint information to the appropriate RESTCaller fields rather than use the ENDPOINT authorization type.

### Use Case #2: Using Different Servers for Different Environments

The typical use case is where you may have a resource server for the QA environment and a separate resource server for the Production environment. For example:

QA:

```
http://qaserver:port/resources/resource
```

Production:

```
http://productionserver:port/resources/resource
```

While the resource path is constructed the same way in both cases, the server address is different. By creating a custom endpoint, you can set the service address to the correct value for the particular environment at the point of deployment without having to change the orchestration or create some custom mechanism.

In this case, you would simply map the custom endpoint Url parameter to the RESTCaller restUrl argument, since the value you want for the server is set at the time of deployment.

### Use Case #3: Using the Same Server with Different Resources Using a Resource Path

Another use case is where a REST service provides access to a number of different resources. For example, these can be collections of resources, such as an initial resource that returns a list of states as follows:

```
http://server:port/states
```

With additional resources that return a specific set of data values concerning San Francisco, such as the following:

```
http://server:port/states/CA/cities/SanFrancisco
```

You could use an orchestration step to calculate a particular resource path depending on some separately provided value. For example you might pass the values, `state = CA` and `city = SanFrancisco` to the orchestration through its event structure or some other means. You could add an orchestration step to calculate the URL to pass to the REST service, taking the base URL from the custom endpoint as follows:

```
http://server:port/resources/states
```

and constructing the resource path from the data given and appending it to the URL provided by the custom endpoint as follows:

```
CA/cities/SanFrancisco
```

Finally, you would pass the following constructed URL to the RESTCaller restUrl parameter.

```
http://server:port/resources/states/CA/cities/SanFrancisco
```

### Use Case #4: Using the Same Server with Different Resources Using Query Strings

Some REST services use the HTTP query string to provide additional ways of selecting the resource. You can append these query parameters to the URL as a properly-formatted HTTP query and map them to the RESTCaller restUrl parameter or you can provide them as an array of key-value structures using the RESTCaller params argument, leaving RESTCaller to append the query string.

For more information on RESTCaller, refer to Chapter 10: Calling RESTful Web Services from an Orchestration Workflow [page 237].

## Running SBM ModScript from an Orchestration

You can use the RunModScript Application Engine Web service call from a **Service** step in an orchestration to execute SBM ModScript.

For example, in an orchestration, you might want to invoke a script from the command-line, read/write data to a file, or update an item with text. You can have an orchestration call SBM ModScript for any of these tasks and more.

The sbmappservices72 WSDL exports a RunModScript function that the Orchestration Engine can invoke. The interface allows the caller (Orchestration Engine) to provide input to the ModScript via an "inputs" data array; and allows ModScript to send output to the caller via an "output" data array. As such, you can fully incorporate a ModScript as a Service step in an orchestration.

Note that even though the orchestration may be initiated by an item transition, ModScript will not have a Shell.Item(). Instead, the Orchestration Engine can send the item's tableID:itemID as a parameter to the ModScript. The script could reference this input to read the item as a custom variable.

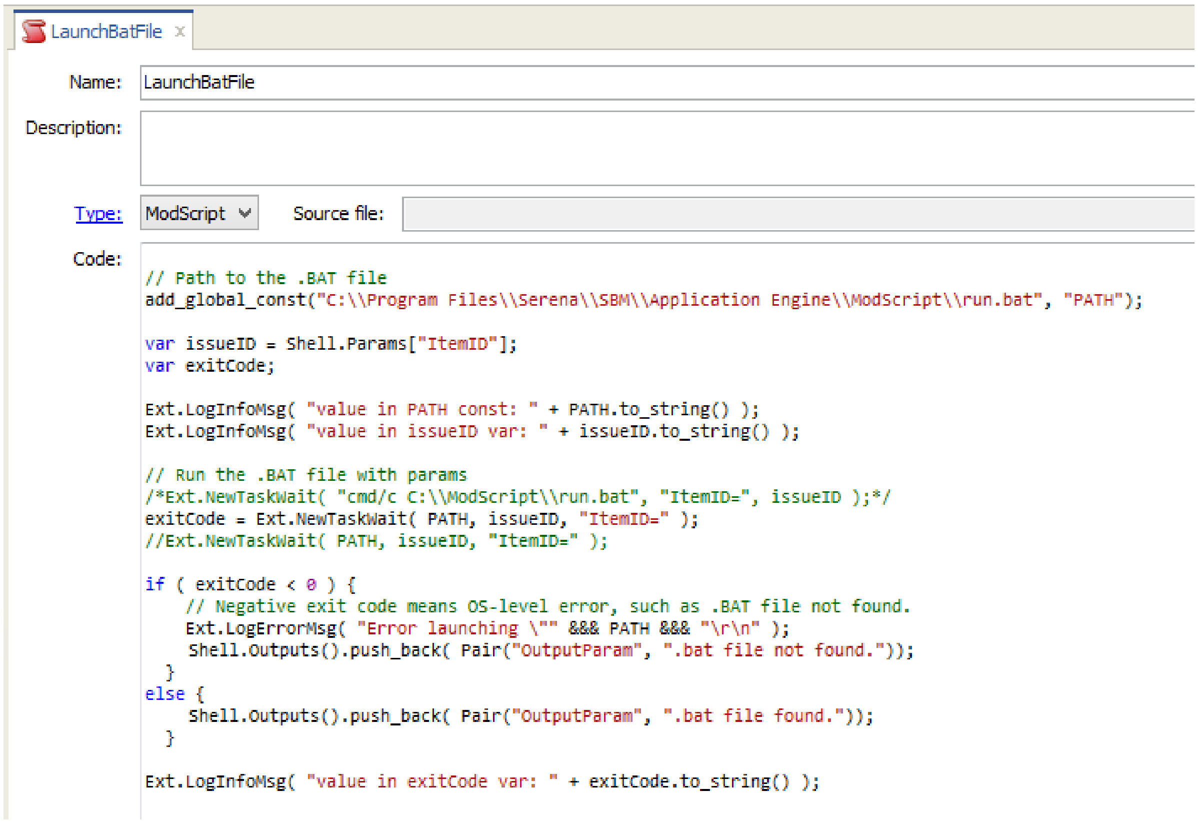
For details on the RunModScript Web service call, refer to the *SBM Web Services Developer's Guide* or SBM Composer help.

For details on programming with SBM ModScript, refer to the *SBM ModScript Reference Guide* or SBM Composer help.

## Example: Running a Batch File

In the following example, the "LaunchBat" orchestration contains a **Service** step that calls the RunModScript Web service, which runs a ModScript called "LaunchBatFile" that executes a simple batch file.

"LaunchBatFile" ModScript added to the list of **Scripts** in SBM Composer:



Name: LaunchBatFile

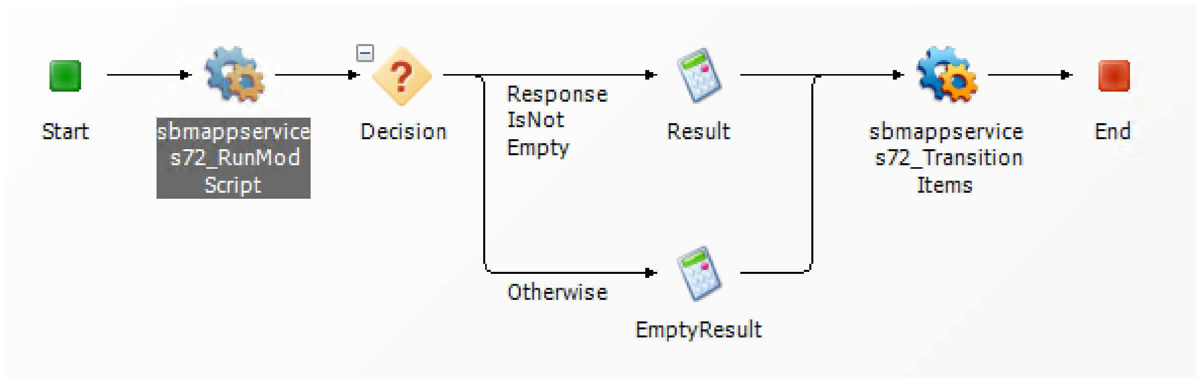
Type: ModScript

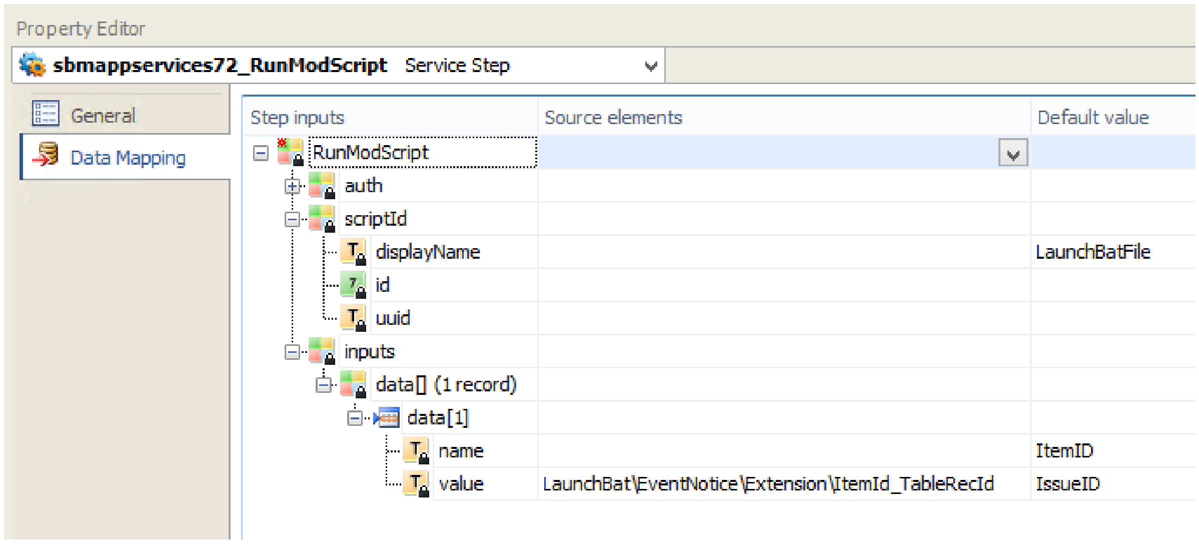
```
// Path to the .BAT file
add_global_const("C:\\Program Files\\Serena\\SBM\\Application Engine\\ModScript\\run.bat", "PATH");

var issueID = Shell.Params["ItemID"];
var exitCode;

Ext.LogInfoMsg( "value in PATH const: " + PATH.to_string() );
Ext.LogInfoMsg( "value in issueID var: " + issueID.to_string() );

// Run the .BAT file with params
/*Ext.NewTaskWait( "cmd/c C:\\ModScript\\run.bat", "ItemID=", issueID );*/
exitCode = Ext.NewTaskWait( PATH, issueID, "ItemID=" );
//Ext.NewTaskWait( PATH, issueID, "ItemID=" );

if ( exitCode < 0 ) {
    // Negative exit code means OS-level error, such as .BAT file not found.
    Ext.LogErrorMsg( "Error launching \"" &&& PATH &&& "\r\n" );
     Shell.Outputs().push_back( Pair("OutputParam", ".bat file not found."));
  }
else {
     Shell.Outputs().push_back( Pair("OutputParam", ".bat file found."));
  }

Ext.LogInfoMsg( "value in exitCode var: " + exitCode.to_string() );
```

Contents of `run.bat`, which is saved in `C:\Program Files\Serena\SBM\Application Engine\ModScript`:

```
@ECHO off
REM --- run.bat
REM --- This is an example batch program to be called
REM --- from the SBM ModScript example "Launch.tsc"
REM --- It appends a new line to C:\out.txt each time it runs
ECHO Launched run.bat, params (if any): %* >> "C:\Program Files\Serena\SBM\
Application Engine\ModScript\out.txt"
```

RunModScript **Service** step in the "LaunchBat" orchestration workflow:



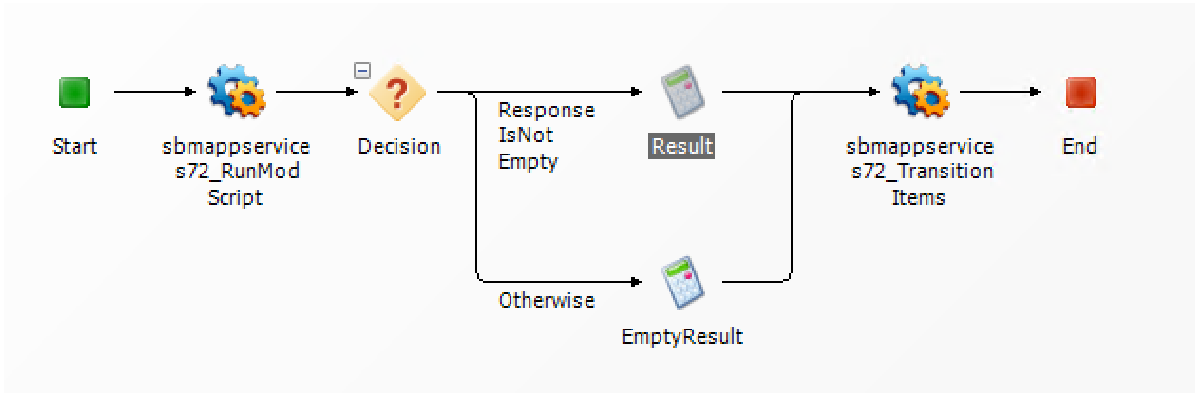


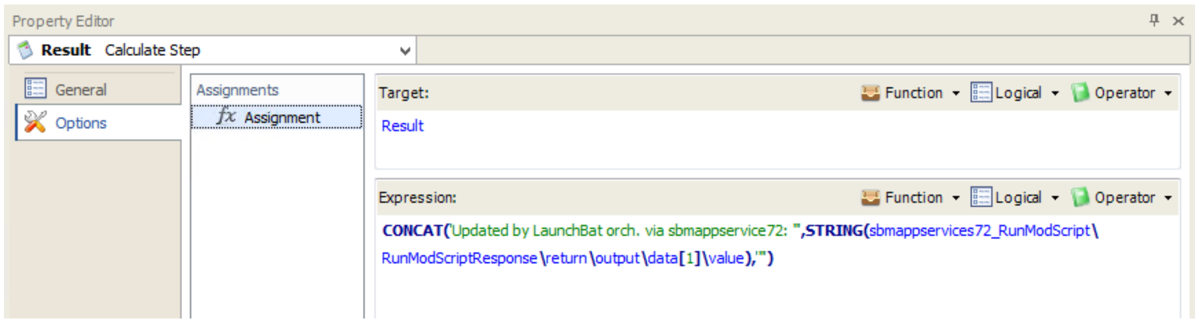
**sbmappservices72_RunModScript** Service Step

| Step inputs | Source elements | Default value |
|---|---|---|
| RunModScript | | |
| auth | | |
| scriptId | | |
| displayName | | LaunchBatFile |
| id | | |
| uuid | | |
| inputs | | |
| data[] (1 record) | | |
| data[1] | | |
| name | | ItemID |
| value | LaunchBat\EventNotice\Extension\ItemId_TableRecId | IssueID |

**Note:** In the RunModScript call, you can send any one of the elements in scriptId—you do not need to provide values for every element. You only need to provide more than one element in the event that the first element does not uniquely identify the script.

In the **Calculate** step, the RunModScript response is concatenated to hard-coded text in the **Result** target:




Property Editor

**Result** Calculate Step

General

Options

Assignments

Assignment

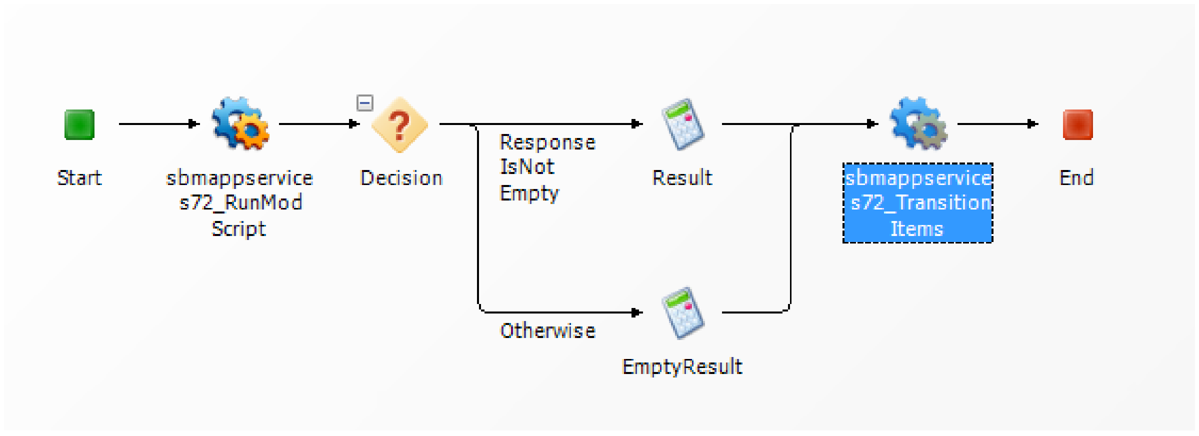
Target:

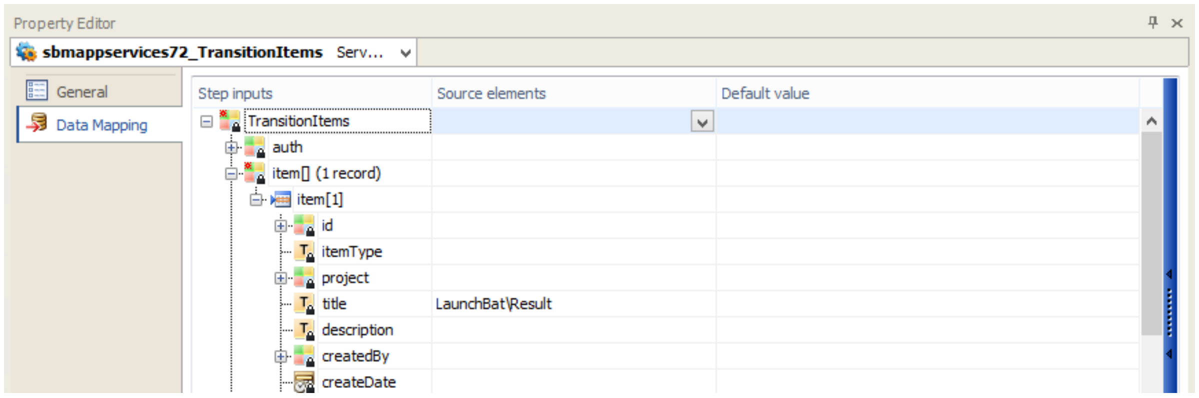
Result

Expression:

CONCAT("Updated by LaunchBat orch. via sbmappservice72: ",STRING(sbmappservices72_RunModScript\RunModScriptResponse\return\output\data[1]\value),"")

The item's *Title* field is updated via the TransitionItems Web service call:




Property Editor

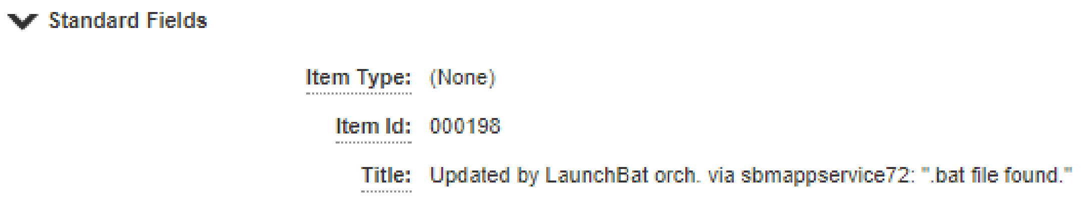
**sbmappservices72_TransitionItems** Serv...

General

Data Mapping

| Step inputs | Source elements | Default value |
|---|---|---|
| TransitionItems | | |
| auth | | |
| item[] (1 record) | | |
| item[1] | | |
| id | | |
| itemType | | |
| project | | |
| title | LaunchBat\Result | |
| description | | |
| createdBy | | |
| createDate | | |

SBM item showing the result in the *Title* field:


**Standard Fields**

| | |
|---|---|
| Item Type: | (None) |
| Item Id: | 000198 |
| Title: | Updated by LaunchBat orch. via sbmappservice72: ".bat file found." |

The `output.txt` file results:

```
Launched run.bat, params (if any): 1007:3 ItemID=
```

# Chapter 5: Orchestration Best Practices

This section provides standards to help you successfully build orchestration workflows that are easy to maintain and scale.

This section contains the following topics:

## Interaction with Application Workflows

Orchestration workflows often manipulate items in application workflows using **sbmappservices72** operations. This section discusses what you should and should not do when manipulating items in this manner.

- It is not necessary to provide auth with **sbmappservices72** Web service calls because the user that invokes the orchestration workflow is automatically granted a Security Token on successful log in to SBM. This means that you can ignore the auth structure in SBM Application Engine Web service calls. However, if you provide credentials in the auth element, they will override the Security Token auth. For example, this might be useful if the calling user does not have privileges to update the item.
- Never directly modify the *State* field of an SBM item. The application workflow should always control the state of items as they progress through the process flow.
- Use data fields and decisions in application workflows to provide process control. Decisions can respond to changes in data made by the orchestration workflow. Asynchronous orchestration workflows can use Web services to set data in tables; synchronous orchestration workflows return data that can change the data in the item.

  Do not make decisions in an orchestration workflow concerning what happens in an application workflow. For example, to decide which outgoing transition from a state should be executed, in the application workflow, change data in the item, and use a decision on the outgoing transition in the application workflow to route the item correctly. This has two advantages: it decouples the application and the orchestration, so that changing the application logic is less likely to affect the orchestration; and it makes the application workflow more clearly describe the process flow.
- Never directly modify the *Owner* field of an SBM item. The application workflow should always control the ownership of items as they progress through the process

flow. Use a *User* field as the owner of a state, and change the user as needed to control the ownership of an SBM item.

- You can use custom **Any**-to-**Any** step transitions to raise orchestration workflow events. Instead of raising events on every application workflow transition to invoke a specific orchestration workflow (for example, "update" or "create" to external tool), create custom transitions from the **Any** state back to itself. Then use transition actions on each application workflow transition to execute the appropriate transition from the **Any** state. This results in an event definition (Application Link) that does not change whenever a new or existing application workflow transition needs to raise a specify type of event.

  **Note:** To hide the transition button on the form, select the "hide button" option on the **Options** tab of the transition Property Editor. To hide the form from the user, select the "quick transition" option.

- When there are multiple asynchronous orchestration actions on the same transition, only one event of each event type is raised. Events start orchestration workflows that will run simultaneously depending on available resources, so the ordering of event actions on the **Actions** tab has no effect on the execution order of the orchestration workflows.

  **Note:** See the "Considerations for Using Actions" topic in the *SBM Composer Guide* for details.

- To optimize performance, the best practice is to send only necessary data in an event and avoid passing large amounts of data. Passing too much data in the event may result in a stack overflow error in the event manager (such as `java.lang.StackOverflowError`). If you encounter this problem with your event design, contact Support and reference solution S138101 to receive help with increasing the JVM stack size . However, to avoid modifying the stack size, or accommodate larger sizes of "multi-" data elements, use the sbmappservices72 GetItem call within the orchestration to retrieve that data.

## Naming Standards

Adhering to naming standards makes it easier to maintain and understand an orchestration workflow.

### Step Names

Best practices for orchestration workflow step names follow:

- Use camel case (CamelCase) to name steps. Do not use all uppercase or all lowercase letters.

- Name the steps to describe what they are doing:
  - A **Calculate** step assigns a value to something. It has two parts: the *Target* and the *Expression*. Name the step to describe what is being set in the *Target*. For example, use **SetTitleField** instead of **Calculate**.
  - Name **ForEach** and **While** steps to describe what they are processing. For example, use **ForEachConfigurationRecord** instead of **ForEach**.

    - Name **Service** steps to describe what the Web service operation is doing. For example, use **CreateAddress** instead of **CreateAuxItem**.

- Name **Service** steps using a prefix for the Web services. For example, use **SBM_GetTestCases** for the **sbmappservices72** "GetItems" operation to distinguish it from non-SBM Web services.

- Do not use a numeric suffix to indicate a different invocation of the same Web service. For example, do not use **SBM_GetItemsByQuery** and **SBM_GetItemsByQuery1**. Instead, use **SBM_RetrieveConfigurationRecords** and **SBM_RetrieveMatchingDefects**.

- Name each **Decision** step as a question and name each branch from the step as the answer. For example, use **ValidName** for the step name and **No** for the branch name.

  **Note:** In newer versions of SBM, spaces are allowed in branch names. However, in earlier versions spaces are not allowed, so those branches use camel case.

### Working Data Element Names

You use working data elements (variables) to hold intermediate data and other data needed by orchestration workflow steps. It is important to name these data elements in a way that makes it clear what role they play and how they are used.

Best practices for working data element names follow:

- Use camel case (CamelCase) to name the data elements. Do not use all uppercase or all lowercase letters.

- Use a name that makes it clear what kind of data is being stored. For example, do not use `DataElement` or `Temp`. Use descriptive names such as `SBMAuth` or `ItemName` instead.

- Use plurals or `List` to name working data array elements. For example, `WorkItems` or `WorkItemList`.

## Usage

The following sections provide guidelines for good usage.

### Orchestration Workflows

Best practices for orchestration workflow usage follow:

- Limit the data that is passed into the workflow to improve the performance of both asynchronous and synchronous workflows. By default, all fields in the primary table are passed to an aynchronous orchestration workflow; you can reduce this by clearing the check boxes for unnecessary fields in the Orchestration Link in the application. For synchronous workflows, you can choose both inputs and outputs in the Orchestration Link; select only those fields that are needed. (See About Orchestration Links [page 29] for information about orchestration links.)

**CAUTION:**

Do not limit the fields that are passed to the workflow if it means that you will later have to retrieve the field data with a Web service call in a **Service** step. The overhead of the Web service call is greater than any benefit gained by limiting the fields.

- If a Web service returns a large amount of data in the response, the entire body of data is loaded into memory, which can cause performance problems. Instead of requesting the entire body of data in a single call, consider breaking it into smaller pieces. For example, instead of using a "get" operation on 100 items, create a loop that gets information for twenty items at a time. You also throttle on payload size. For information about this, see the "Scaling Orchestrations" white paper. To access this white paper, visit the Knowledgebase, and then search for solution S136965.
- Use asynchronous (not synchronous) orchestration workflows to:
  - Update or insert data, especially on external systems
  - Call Web services that can take a long time, because synchronous orchestration workflows can time out.

  Synchronous orchestration workflows should only be used if the user must see the result of a transition immediately, and only to validate data the user entered or populate fields on a form.
- If a Web service throws a number of SOAP faults, use separate fault handlers for each fault. This reduces the complexity of the fault handlers; makes the workflow design clean and concise; and reduces the size of the workflow, which makes it more scalable.
- Check for an existing item in an "update" workflow and end the workflow if the item does not exist. Similarly, check for an existing item in a "create" workflow and end the workflow if the the item exists. This makes your orchestration handle error cases gracefully.
- When an orchestration workflow or Web service call updates an external tool, that tool could, in response to the update, raise an event in SBM to synchronize the change. Because the change originated in SBM, the event can be ignored. For example, suppose there are two users: "SBMUser" and "ExternalUser." "SBMUser" updates an SBM item, and the *Last Modifier* field value is "SBMUser." The connector replicates the item in the external tool using "ExternalUser," which automatically raises an event that is sent to SBM. In this case, the event should be ignored and the workflow should end.
- When implementing integrations between SBM and an external tool, use a dedicated SBM user and a dedicated external system user for updating items in the external system. This can prevent unintended permission errors during the execution of the orchestration workflow.
- Use references between items in SBM and an external tool. For example, external tools should contain a field to hold the SBM item ID. This allows workflows to easily identify the target SBM item and know if one has been created yet. On the other hand, the SBM item should have a unique identifier field to hold a reference to an item in the external tool. This could be two fields, such as *Database Name* and *Item ID*.

- If data is the same among SBM environments, create a working data element to hold the data. If the data is different among environments, store the data in auxiliary tables and retrieve the information dynamically depending on the situation. The following items describe this principle in more detail, using authentication data as an example.
  - In a connector scenario, authentication data is likely to be the same for different environments (such as Development, Testing, and Production). Normally, if you are using SBM Web services, no authentication credentials are needed; however, you can hard-code a designated user in the SBM Web services auth element to override the calling user if the permissions for the calling user are not adequate. Make sure that the overriding user has all of the permissions the workflow needs (for example, permission to read table data, update items, and so on). If you need to override the calling user, create working data elements to hold and store the authentication data for the SBM user.
  - In a direct SBM-to-external tool scenario, the authentication data for users can be different among environments. By default, the workflow runs in the context of who invoked it. It is a good practice to store the authentication data for each user in an auxiliary table. The orchestration workflow queries the data based on the user and then stores it in working data elements.

  Whether the user name and password are stored in working data or in an auxiliary table depends on the following:
  - Whether the authentication data might need to be modified by an administrator in the auxiliary table. Modifications could be needed if the authentication data is different among environments or if the company policy is to change the data regularly. If this data is stored in working data, it is more difficult to change and requires redeploying the process app.
  - Whether there are policies concerning where sensitive data like user names and passwords are stored. For example, you might not want developers to have access to the user name and password for the payroll system.

  Because access to auxiliary tables require a set of permissions, you might need to put the SBM user name and password in the working data. To keep this information private, you can use HTTPS for your Web service calls.
- If you are calling a Web service from an orchestration, and the call requires a certificate for authentication, ensure that your certificate contains the `clientAuth` property if it is required. For example:

  ```
  #8: ObjectId: 1.2.29.37 Criticality=false

  ExtendedKeyUsages [

    1.2.3.1.5.5.8.2.2

    clientAuth

    serverAuth

  ]
  ```

## Steps

Best practices for step usage follow:

- Use **ForEach** steps to always process *every* item in an array element.
- Use **While** steps to process an item only if a certain condition exists before entering the loop and while in the loop.
- Use **Calculate** steps to perform manipulations such as string concatenation and data conversion. You can also use **Calculate** steps to store intermediate data that needs to be extracted from previous **Service** steps or events in the workflow, as described below.
- Use **Calculate** steps to extract a specific element from an array of elements. The most common use is to extract a specific field from the `extendedField` of a response from a "GetItem" operation. For example:

  ```
  SBM_GetItemResponse:SBM_GetItem.GetItemResponse.return.item.extendedField
  →[id.dbName="PRODUCT_BACKLOG"].value[internalValue]
  ```

- Do not use **Calculate** steps to transform large amounts of data because it requires a large number of **Calculate** steps. It is better to use a Web service to do this.
- Use only the number of **Calculate** steps that you need. Excessive **Calculate** steps can slow orchestration workflow processing.
- If you use several **Calculate** steps to achieve a single result (such as transforming one data structure into another), group those steps into a **Group** step and give it a name that describes the transformation.
- Do not use working data elements to store data that can be mapped directly from the outputs of a previous **Service** step to the inputs of a subsequent **Service** step. For example, the `ItemId` of an **SBM_GetItem** Service step can be mapped directly as an input to an **SBM_UpdateItem** Service step.
- Consider using the main branches from a **Decision** step to handle expected use cases, and reserving the **Otherwise** branch for errors where none of the expected cases occur.
- Use **Scope** steps to create a structure that handles faults that occur during the execution of a Web service.
- Use **Group** steps to organize steps that together perform a larger set of logic.

## Step Functions

The expressions used in **Calculate**, **ForEach**, and **While** steps can contain a variety of Supported XPath Functions [page 33]. Best practices for step function usage follow:

- Use the `NUMBER` function to convert text to integers. Use the `STRING` function to convert integers to text.
- The `CONCAT` function supports the concatenation of any number of arguments. When you use the `CONCAT` function, provide all arguments in a single call to build a text string; do not use nested `CONCAT` calls.

- Use the `STRINGLENGTH` function to determine if a text element is empty or has data. For example, use `STRINGLENGTH(SomeElement) = 0` instead of `SomeElement = ""` and `STRINGLENGTH(SomeElement) > 0` to determine whether a text element contains data.

- When you test for a negative (does not contain), use the `NOT` function to turn a branch into a positive. For example, use `NOT(CONTAINS(SomeText, "some string"))` for a branch that is taken when the `SomeText` element does not contain "some string."

## Working Data Elements

Best practices for working data element usage follow:

- If you have a common set of data that is used in multiple places in the workflow (for example, authentication credentials), create working data elements that store the data in one place, instead of setting the data as default values in every **Service** step. You can then map the working data to the authentication inputs for the steps. If you need to change the credentials later, you can do it in working data one time, instead of in each step.

  **Important:** If the values differ among environments, store the data in an auxiliary table and query the table to populate the working data at the beginning of the orchestration workflow.

- Use library types for elements where applicable. For example, use the the **sbmappservices72** `auth` type instead of separate data elements for `userId` and `password`. This lets you map one working data element to one **Service** step data element instead of two.

  **Note:** To use library types, right-click a new working data element, select **Type**, select **Select from Type Library**, and then select the type from the Select Library Type Dialog Box [page 55].

- Store SBM `itemID` inputs as text, because the **sbmappservices72** operations all expect text values.

- Provide working data elements to store intermediate data extracted from array elements. For example, use `ItemProductBacklog` to store:

```
SBM_GetItem.GetItemResponse.return.item.extendedField
→[id.dbName="PRODUCT_BACKLOG"].value[internalValue]
```

- Generally, you will use quotes to surround string constants in working data variables. When working with string constants that contain quotation marks, use either single or double quotes to surround string constants, depending on the type of quotation mark in the string. For example, to insert a single quote, surround the string constant with double quotes. To insert a double quote, surround the string constant with single quotes. If you do not need to specify either type of quote in the content, you can use either single or double quotes to surround the string constant.

## Web Services

SBM includes two Web services that provide numerous operations: **sbmappservices72** and **sbmadminservices72**. Best practices for Web service usage follow:

- Never use both display names and interval values. The display name is what the user sees and the internal value is what is stored in the database. For text fields these are the same, but for many field types they are completely different. Setting the display name and internal value to the same value will prevent the field from being updated. For example:

- A *Selection* field's display name could be **High**, but its internal value could be `13` (the ID of the record in the *TS_SELECTIONS* table.)
- A transition's display value could be **Delegate**, but its internal value could be `WorkflowName.Delegate`.

- Make sure the values for *Selection*, *Relational*, *User*, and *Group* fields (*Single* and *Multi-*) in SBM and the external tool match exactly. For example, if the display value is used in one, make sure the display value, not the internal value, is used in the other. In *Multi-Relational* fields, use arrays to use display values or internal values consistently.
- When you need to create and assign values to extended fields, create the array elements directly in the **Service** steps and assign values to them directly, instead of trying to dynamically create them.
- The `createDate` and `createdBy` data elements represent the *Submit Date* and *Submitter* fields in the item. (The SBM Web services use these data element names instead of the database field names.)
- Orchestrations are not intended to be used to process very large blocks of data or file attachments. For example, the **sbmappservices72** GetFileAttachment call enables you to retrieve file attachments for a given item; however, you should avoid using this call in an orchestration because orchestrations are not designed to handle this kind of data efficiently. If this is unavoidable in your application, set the attachment element in the response message to an empty string as soon as the reason for bringing the attachment into the orchestration has been satisfied before the orchestration finishes. This frees the attachment data block and should allow the system to recycle the memory. Otherwise, the Orchestration Engine will persist the attachment to the Orchestration Engine database as part of the response message. This can cause orchestration processing to take a long time and cause unnecessary space to be allocated in the Orchestration Engine database.

**Note:** See the *SBM Web Services Developer's Guide* for detailed usage information and best practices for each operation.

## Event Handling

Each orchestration workflow should handle only one type of event. This can be achieved even if an event needs to either create a new item if one does not yet exist, or update an item if one does exist. The same event can be consumed by two workflows, one for "create" and one for "update." For example, the "create" workflow would check whether the item exists and end if it does, or process it if it does not. The "update" workflow would check whether an item exists and end if it does not exist, or process it if it does exist.

In most connector orchestration process apps, there are six orchestration workflows to handle events:

- Create an SBM item from an external item.
- Update an SBM item from an external item.

- Delete an SBM item from an external item. (This rarely happens because the item should instead be closed.)
- Create an external item from an SBM item.
- Update an external item from an SBM item.
- Delete an external item from an SBM item. (This is rarely used because most process apps do not allow users to delete items.)

If you want to raise an event from an external tool that will invoke an orchestration workflow, define the interface in a custom event definition and then export a WSDL file from the event definition. The external tool uses the WSDL file to call Web services that raise events. This is described in About Application Links and Event Definitions [page 28].

You can design SBM and external tools to provide operations that expose high-level business granularity. For example, you could have a set of orchestration workflows that include "create," "transition" (update), and "delete" operations that are invoked by transitions in the application workflow. You could have similar orchestration workflows that are invoked by the external tool.

**Note:** The custom event definition is simply a description of the events. It does not automatically implement an event generator from the external tool.

When SBM Application Engine calls the Orchestration Engine via the ALF event mechanism, it sends any empty numeric primary table fields using the following max values: `4294967295` for floating point or fixed precision number and `2147483647` for an integer field. These numbers are defined constants in the SBM Application Engine. You can avoid this by: making any numeric fields that you send on an event required, giving them a default value, or by ensuring the user provides a value by some other means. If that is not possible, test for these max values in any orchestrations that might receive them and handle the case as appropriate for your application (for example, by changing the value to 0).

## Scalability

This section provides best practices for designing orchestration workflows to maximize performance and scalability.

- An application that performs large-scale data transformations will not scale well if it is implemented using an orchestration. Instead of using a single orchestration to process a large number of items, break the work into smaller chunks or consider using another method to process the data. In general, use orchestrations only to enable collaboration between systems and users.
- Use orchestration workflows to solve the following business problems:
  - **Collaboration with existing legacy systems:** If a legacy system can receive Web service calls, it can be called from asynchronous orchestration workflows that are executed during transitions in an application workflow.

    **Important:** It is recommended that you use asynchronous (not synchronous) orchestration workflows to do this. See Usage [page 175] for synchronous orchestration workflow limitations.

   - **Collaboration with SOA capable systems:** Application workflows (also known as "human workflows") can easily be integrated with other SOA (service-oriented architecture) systems. These integrations use orchestration workflows initiated from the human workflow to call Web services for passing data to and receiving responses from the external systems. In addition, because the Event Manager can be called as a Web service, asynchronous orchestrations can be initiated from events occurring in the external systems to communicate back to the SBM application.
   - **Intelligent data enrichment:** Users make decisions and take actions based on information available within the system. Often they can make better decisions if they have access to related information stored outside of the system. A synchronous orchestration workflow can implement business logic to collect and coordinate data from several sources to present in a form. For example, a form in a credit approval application can include a customer's credit information.
   - **Validation:** Synchronous orchestration workflows that use programming constructs such as comparisons, string and number manipulation, and loops can be used to validate data that users entered. Validation can also occur through the use of JavaScript in custom forms.
   - **Process-level synchronization:** In this scenario, the data between two or more systems is continuously synchronized as items flow through various business processes. Another synchronization scenario is not suitable for orchestration workflows and is described below.
- Do not use orchestration workflows to perform the following tasks:
   - **Visual programming:** SBM Composer provides the ability to visually create orchestration workflows using constructs such as loops, decisions, data manipulation, and fault handling. While it is possible to use these constructs as a general purpose programming language to create complex procedural programs, it is discouraged. It is better to use these constructs to apply business logic while coordinating or orchestrating the Web services called in the workflow.

     Drawbacks of adding long modules within orchestration workflows include: orchestration workflows become too long, so they are difficult to view, debug, and maintain; designers lose the benefit of compile time checks that other languages such as Java offer; and performance of these constructs is nowhere near as fast as that of traditional programming languages. Complex logic or data manipulation should be moved out of orchestration workflows and into Web services that can be invoked as a part of the orchestration workflow.
   - **ETL processing:** Data warehousing Extract, Transform, and Load (ETL) processing is data intensive and requires minimum overhead for each record and very fast data transformation processing. Implementing ETL with an orchestration workflow entails at least two Web service calls for each record that is processed, as well as the overhead for each record for data transformation. Using an orchestration workflow for ETL processing results in poor performance and unacceptably complex visual programming.
   - **Batch process synchronization:** In this scenario, synchronization takes place after the fact in a batch process. It involves processing a large number of records in multiple systems to ensure that they remain synchronized after changes to one or more system. An example of this is synchronizing a customer or product

master database with data stored in distributed systems. To achieve this using an orchestration workflow, you would need to iterate through each record on each system, making multiple Web service calls for each iteration; comparing the data between systems, and if necessary, update the data by making additional Web service calls.

## Security

This section discusses security considerations for orchestrations.

To call a service using HTTPS from an orchestration workflow, you must establish a trust by importing the SSL certificate that the service uses. This applies to REST services called via the RESTCaller as well.

To execute an external Web service call from SBM using SSL, the SBM certificate truststore must contain the external service's public certificate (in the event that the certificate does not already exist in the truststore). Therefore, you must import the service's public certificate into either the Windows or Tomcat truststore—depending on which SBM component performs the call.

For example, if the external Web service call is invoked from a workflow transition, you must add the public certificate to the Windows truststore in the IIS tab on the IIS server. This ensures that SBM Application Engine calls are trusted by the external service. Similarly, you must add the public certificate to the Tomcat truststore to ensure that SBM Orchestration Engine calls are trusted by the external service. For example, if you create an SBM orchestration that contains an external Web service call that is secured by SSL, the public certificate for that service must be added to the Tomcat truststore.

Consult your SBM administrator, and use the **Manage Trusted Certificates** option in SBM Configurator (*SBM On-Premise only*) to import the service's public certificate into the JVM truststore. The truststore may already contain some public certificates, but if you create your own certificates or use certificates that are newer than those the truststore, the truststore must be updated to successfully complete calls over HTTPS.

# Chapter 6: Orchestration Tutorial

The exercises in this tutorial demonstrate how to use SBM Composer to create orchestrations. This tutorial takes approximately 1/2 hour to complete, and requires that you first complete the process app tutorial, which also takes approximately 1/2 hour. See the *SBM Composer Guide* for the process app tutorial.

In the orchestration tutorial, you create two synchronous orchestration workflows, and one asynchronous orchestration workflow with a Web service. All three workflows are necessary for the process app to work correctly.

This tutorial contains the following steps:

## Step 1: Create an Orchestration

**To create an orchestration:**

1. In App Explorer, right-click **MyProcessApp**, point to **Add New**, and then select **Orchestration**.
2. In the **New Orchestration** dialog box, enter `MyOrch` in the **Name** box.
3. Click **OK** to close the **New Orchestration** dialog box.

   **Application Links**, **Orchestration Workflows**, **Type Library**, and **Web Services** appear under the new **MyOrch** heading in App Explorer.
4. Continue to Step 2: Create a Synchronous Pre-Transition Orchestration Workflow [page 185].

## Step 2: Create a Synchronous Pre-Transition Orchestration Workflow

In this exercise, you use the **Action Wizard** to create the *SynchBefore* orchestration. SynchBefore is a synchronous pre-transition orchestration workflow. When SynchBefore is invoked from Submit transition, it updates (adds text to) the **Title** box of the transition page before the page loads.

**To create a pre-transition orchestration workflow with reply:**

1. In App Explorer, under the **Application Workflows** heading, select **MyAppWorkflow**.
2. In the application workflow editor, right-click the **Submit** transition, and select **Show Actions**.
3. On the **Actions** tab of the Property Editor, click **New.**

   The **Action Wizard** asks, "Which type of action do you want to execute?"
4. From the list of action types in the **Step 1** box, select **Orchestration Workflow**.
5. In the **Step 2** box, click the **and continue executing (asynchronous)** link, select **and wait for reply (synchronous)** from the list, and then click **Next**.

   The **Action Wizard** asks, "When do you want to call the orchestration workflow?"
6. In the **Step 1** box, select **Before**, and then click **Next**.

   You do not do anything in the **Step 2** box, which reads "Invoke an orchestration workflow and wait for reply (synchronous), before this transaction occurs".

   The **Action Wizard** asks, "Which orchestration workflow do you want to call?"
7. On the menu under **Step 1**, select **(Add new workflow...)**.

   You do not do anything in the **Step 2** box, which still reads "Invoke an orchestration workflow and wait for reply (synchronous), before this transaction occurs".
8. In the **Event with Reply** dialog box, do the following:
   a. Change the **Name** to `MyOrchWFWR` (which is short for MyOrch Workflow with Reply).

   b. From the **Orchestration** list, select **MyOrch.**

   c. In the **Workflow** box, change the name to `SynchBefore`.

   d. In the **Fields used by event** column, select the **Title** check box.

   e. In the **Fields returned by event** column, select the **Title** check box.

   f. Click **OK.**

   You do not do anything in the **Step 2** box, which now reads "Invoke an orchestration workflow and wait for reply (synchronous), before this transaction occurs, Call MyOrchWFWR | If form is invalid, don't rerun this service."
9. In the **Action Wizard**, without changing anything in the **Step 2** box, click **Finish**.

   A new icon appears next to the transition name in the application workflow editor indicating that an action is associated with the transition. The action is also listed on the **Actions** tab of the **Property Editor**.

   In addition, in App Explorer, under the **MyOrch** heading, **SynchBefore** appears under **Orchestration Workflows**, and **MyOrchWFWR** appears under the **MyApp** heading, under **Orchestration Links**.

10. In App Explorer, under **MyOrch**, under **Orchestration Workflows**, select **SynchBefore**.

11. In the **New Items** section of the **Step Palette**, drag and drop a **Calculate** step onto the line between the **Start** and **End** steps.

12. On the **Options** tab of the Property Editor, in the **Target** section, enter the following expression: `EventNoticeWithReply\Extension\Title`

13. In the **Expression** section, enter the following function: `STRING("Replace this text with something unique, 25 characters or less.")`

    For more information about creating expressions, see About the Expression Editor [page 32].

14. In the orchestration workflow editor, select the **End** step.

15. On the **Data Mapping** tab of the Property Editor, expand the **Extension** data element, locate the **Title** data element, select the corresponding cell in the **Source elements** column, and then click the down arrow.

16. In the **Select a Source** popup, expand **SynchBefore**, **Inputs**, **EventNoticeWithReply**, and **Extension**; select **Title**; and then click **OK.**

    **SynchBefore\EventNoticeWithReply\Extension\Title** appears in the **Source elements** column.

17. Continue to Step 3: Create a Synchronous Post-Transition Orchestration Workflow [page 187].

## Step 3: Create a Synchronous Post-Transition Orchestration Workflow

In this exercise, you use the **Action Wizard** to create the *SynchAfter* orchestration workflow. SynchAfter is a synchronous post-transition orchestration workflow. When SynchAfter is invoked from the Submit transition, it appends the text in the **Title** box of the transition page before the state form loads. The new appended text appears in the **Title** box on the state form.

**To create a synchronous post-transition orchestration workflow:**

1. In App Explorer, under the **Application Workflows** heading, select **MyAppWorkflow**.

2. In the application workflow editor, right-click the **Submit** transition, and then select **Show Actions**.

3. On the **Actions** tab of the Property Editor, click **New.**

   The **Action Wizard** asks, "Which type of action do you want to execute?"

4. From the list of action types in the **Step 1** box, select **Orchestration Workflow**.

5. In the **Step 2** box, click the **and continue executing (asynchronous)** link, select **and wait for reply (synchronous)** from the list, and then click **Next**.

   The **Action Wizard** asks, "When do you want to call the orchestration workflow?"

6. In the **Step 1** box, select **After**, do not change anything in the **Step 2** box, and then click **Next**.

   The **Action Wizard** asks, "Which orchestration workflow do you want to call?"

7. In the list under **Step 1**, select **(Add new workflow...)**.

8. In the **Event with Reply** dialog box that opens:

   a. Change the **Name** to "MyOrchWFWR2" (which is short for MyOrch Workflow With Reply #2).

   b. From the **Orchestration** list, select **MyOrch.**

   c. In the **Workflow** box, change the name to `SynchAfter`.

   d. In the **Fields used by event** column, select the **Title** check box.

   e. In the **Fields returned by event** column, select the **Title** check box.

   f. Click **OK.**

9. In the **Action Wizard**, click **Finish**.

   The action is listed on the **Actions** tab of the Property Editor.

   In addition, in App Explorer, under the **MyOrch** heading, **SynchAfter** appears under **Orchestration Workflows**, and **MyOrchWFWR2** appears under the **MyApp** heading, under **Orchestration Links**.

10. In App Explorer, under **MyOrch**, under **Orchestration Workflows**, select **SynchAfter**.

11. In the **New Items** section of the **Step Palette**, drag a **Calculate** step onto the line between the **Start** and **End** steps.

12. On the **Options** tab of the **Property Editor**, in the **Target** section, enter the following expression: **`EventNoticeWithReply\Extension\Title`**.

13. In the **Expression** section, enter the following function:

    ```
    CONCAT(EventNoticeWithReply\Extension\Title, " Appended by
    SynchAfter.")
    ```

    **Note:** Be sure to include the space after the comma.

    For more information about creating expressions, see About the Expression Editor [page 32].

14. In the orchestration workflow editor, select the **End** step.

15. On the **Data Mapping** tab of the **Property Editor**, expand the **Extension** data element, locate the **Title** data element, select the corresponding cell in the **Source elements** column, and then click the down arrow.

16. In the **Select a source** popup, expand **SynchAfter**, **Inputs**, **EventNoticeWithReply**, and **Extension**; select **Title**; and then click **OK.**

**SynchAfter\EventNoticeWithReply\Extension\Title** appears in the **Source elements** column.

17. In App Explorer, under the **MyApp** heading, under **Orchestration Links**, select **MyOrchWFWR2**.
18. On the **MyOrchWFWR2** tab, the **Title** check boxes of the **Fields used by event** and **Fields returned by event** columns should be selected.
19. Continue to Step 4: Create an Asynchronous Orchestration Workflow [page 189].

## Step 4: Create an Asynchronous Orchestration Workflow

In this exercise, you add an event without reply as a transition action on the **Close** transition of **MyAppWorkflow**.

**Asynch** is an asynchronous orchestration workflow. When **Asynch** is invoked from the **Close** transition, it creates a new item in MyOtherAppProject. The **Title** box of the new item contains the text that was added by the **SynchBefore** and **SynchAfter** orchestration workflows and some additional text.

**To create an event without reply:**

1. Add a second application workflow called **MyOtherAppWorkflow**:
   a. In App Explorer, right-click the **Application Workflows** heading, and then select **Add New Workflow**.
   b. On the **General** tab of the Property Editor, change the **Name** field to `MyOtherAppWorkflow`, and then press TAB.
2. In App Explorer, under the **Application Workflows** heading, select **MyAppWorkflow**.
3. In the application workflow editor, right-click the **Close** transition, and select **Show Actions**.
4. On the **Actions** tab of the Property Editor, click **New.**

   The **Action Wizard** asks, "Which type of action do you want to execute?"
5. From the list of action types in the **Step 1** box, select **Orchestration Workflow**.
6. Without changing anything, click **Next.**

   The **Action Wizard** asks, "What do you want to affect?"
7. Without changing anything, click **Next.**

   The **Action Wizard** asks, "Which condition do you want to check?"
8. Without changing anything, click **Next.**

   The **Action Wizard** asks, "Which orchestration workflow do you want to invoke?"
9. In Step 1, select **(Add new workflow...)**.

   **Close** appears and is selected in the **Step 1** list.

10. Without changing anything in the **Step 2** box, click **Finish**.

    A new icon appears next to the transition name in the application workflow editor indicating that an action is associated with the transition. The action is also listed on the **Actions** tab of the Property Editor.

    In addition, in App Explorer, under the **MyOrch** heading, **Close** appears under **Orchestration Workflows** and **MyAppEventDefinition** appears under **Application Links**.

11. In App Explorer, under **Application Links**, select **MyAppEventDefinition**.

12. On the **General** tab in the Property Editor, change the **Name** to `MyOrchWFNR` (which is short for MyOrch Workflow No Reply), and then press the Tab key.

13. Under **MyOrch**, under **Orchestration Workflows**, select **Close**.

14. On the **General** tab of the Property Editor, change the **Name** to `Asynch`, and then press the Tab key.

15. In the **New Items** section of the **Step Palette**, drag and drop a **Calculate** step onto the line between the **Start** and **End** steps.

16. On the **Options** tab in the Property Editor, in the **Target** section, enter the following expression: `EventNotice\Extension\Title`.

17. In the **Expression** section, enter the following string function:
    ```
    CONCAT(EventNotice\Extension\Title," This issue was created by
    Asynch.")
    ```

    **Note:** Be sure to include the space after the comma.

    For more information about creating expressions, see About the Expression Editor [page 32].

18. In the **New Items** section of the **Step Palette**, drag a **Service** step onto the line between the **Calculate** and **End** steps.

    **sbmappservices72** appears under **Web Services**, under the **MyOrch** heading.

    SBM Composer automatically adds the SBM Web Service (sbmappservices72) to new orchestrations. (For more information, see the *SBM Web Services Developer's Guide*.)

19. On the **General** tab of the Property Editor, from the **Service** list, select **sbmappservices72**.

    **Note:** You can also add another Web service by selecting **(add new service...)** on the **Service** menu. In the **Web Service Configuration** dialog box that opens, select the Web service's WSDL file on the **WSDL** menu or click the browse button.

20. From the **Operation** menu, select the **CreatePrimaryItem** Web service operation.

21. On the **Data Mapping** tab, expand the **project** data element, locate the **internalName** data element, and then select the corresponding cell in the **Default value** column.

22. In the **Default value** column, enter the following text: `USR_MYOTHERAPP.MYOTHERAPPPROJECT`. This is the name of the project into which the new item will be submitted.

23. Expand the **item** data element, locate the **title** data element, select the corresponding cell in the **Source elements** column, and then click the down arrow.

24. In the **Select a source** popup, expand **Asynch**, **Inputs**, **EventNotice**, and **Extension**; select **Title**; and then click **OK.**

25. Continue to Step 5: Validate the Process App [page 191].

## Step 5: Validate the Process App

SBM Composer validates a process app before publishing it, but you can also validate a process app at any time by performing steps 1 and 2 in the following exercise.

**Tip:** While you are creating orchestration workflows, you should validate often to catch potential problems before you attempt to publish and deploy a process app.

**To validate a process app:**

1. On the **Deployment** tab of the Ribbon, click the **Validate** button.

   If the process app is ready to be published, the Validation Results advises you that the validation succeeded.

2. To see what a validation error looks like:

   a. In App Explorer, under the **MyOrch** heading, under **Orchestration Workflows**, select any of the orchestration workflows.

   b. In the **New Items** section of the **Step Palette**, drag a **Calculate** step onto the orchestration workflow editor, and drop it anywhere on the line between the **Start** and **End** steps.

   c. Click **Validate** again.

      Details about the errors, warnings, and messages appear in the Validation Results. If the Validation Results is not available, on the **Home** tab of the Ribbon, in the **Common Views** area, select the **Validation Results** check box. If this check box is already selected and the details are still not visible, select the **Validation Results** tab in the area under the editor pane.

      **Note:** For more information about the Validation Results, see the *SBM Composer Guide*. For information about validation errors, see Troubleshooting Orchestrations Using the Validation Results [page 195].

3. To return the orchestration workflow to its valid condition, delete the **Calculate** step that you just added.

4. On the **Deployment** tab of the Ribbon, click the **Validate** button again.

   The Validation Results should show that the validation succeeded.

5. Continue to Step 6: Publish the Process App [page 192].

## Step 6: Publish the Process App

**To publish a process app:**

1. Click the Composer button, and then click **Publish**. If the process app was not saved yet, or if any part of the process app changed since the last time it was saved, a message box opens reminding you to save the changes. Click **OK**.

2. When the **Check In Design Elements** dialog box opens, you can enter an optional comment in the **Comment** box.

3. Click **OK**.

   The **Publish Process App** dialog box opens.

4. You can enter an optional comment in the **Comment** box.

5. In the **Label** box, you can modify the text that identifies this version of the process app.

6. Under **Visibility**, select the **Allow others to deploy this version of the process app** check box.

7. Click **Publish**.

   A message box opens indicating whether the process app published successfully or the operation failed. (See Troubleshooting Orchestrations Using the Validation Results [page 195] for information about publish operation errors.)

8. Click **OK**.

9. Continue to Step 7: Deploy the Process App [page 192].

## Step 7: Deploy the Process App

**To deploy a process app:**

1. On the **Deployment** tab of the Ribbon, click the **Deploy** button.

   The **Deploy Process App** dialog box opens.

2. From the **Environment** list, select a runtime environment, and then click **Deploy**.

   In the Activity Log, a message appears notifying you that the deployment started successfully.

3. Wait for the deployment to complete.

   - If the deployment succeeds, the following message appears in the Activity Log: "Deployment of 'MyProcessApp' has completed."

   - If the deployment fails, the following message appears in the Activity Log: "Deployment of 'MyProcessApp' has aborted or failed."

   If the Activity Log is not available, on the **Home** tab of the Ribbon, in the **Common Views** area, select the **Activity Log** check box. If this check box is already selected and the details are still not visible, select the **Activity Log** tab in the area under the editor pane.

For more information about the Activity Log, see the *SBM Composer Guide*. For information about deployment errors, see Troubleshooting Orchestrations Using the Validation Results [page 195].

4. Continue to Step 8: Run the Process App [page 193].

## Step 8: Run the Process App

In this exercise, you test **MyProcessApp** by running it in SBM Work Center.

**To run a process app in SBM Work Center:**

1. Log on to the SBM Work Center.

   **Tip:** In SBM Composer, you can click the **Work Center** button on the **Launch** tab of the Ribbon to do this. You can also navigate directly to `http://`*`serverName`*`/workcenter`.

2. Click the **MyApp** icon.

3. Click the **+New** button.

4. On the **Browse** tab, click the **MyAppProject** link.

   The first transition page opens, and the title contains the text that you specified in the **Calculate** step of the **SynchBefore** orchestration workflow: `Replace this text with something unique, 25 characters or less.`

5. Delete the text in the **Title** box, enter some unique text (fewer than 25 characters).

6. Select a user from the **Manager** list, and then click **OK**.

   The first state form opens, and the title now contains the text you entered in the previous step followed by the text that you specified in the **Calculate** step of the **SynchAfter** orchestration workflow: `Appended by SynchAfter.`

7. Click the **Assign** button.

   The second transition form opens.

8. Select a user from the **Employee** list, and then click **OK**.

9. Click **Close** and then click **OK** on the page that opens. The **Asynch** orchestration workflow is invoked, and it creates a new item in MyOtherAppProject.

10. Perform a search for the item.

    In the search results list, the item's **Title** column contains the following: `[Your unique text] Appended by SynchAfter. This issue was created by Asynch.`

11. To view the item, click item row.

## Orchestration Reminder List

As you begin creating your own orchestration workflows, keep these points in mind:

- Make sure you have access to SBM Composer, SBM Work Center, and SBM Application Administrator.

- If you plan to use Web services, make sure you can specify the location for each Web service's WSDL file.
- Make sure that the process app you created or selected contains an application, that it has a workflow defined, and that the workflow contains at least one transition between states.
- Before creating an orchestration workflow, make sure you know what data must be passed from the application to the orchestration workflow.

# Chapter 7: Troubleshooting Orchestration Workflows

This section contains the following topics about the resources and tools that you can use to troubleshoot orchestration workflows. You can also use the **Scope**, **Throw**, and **Compensate** steps to help you troubleshoot orchestration workflows (see Using the Scope, Throw, and Compensate Steps to Handle Faults From Web Services [page 96].

## Troubleshooting Orchestrations Using the Validation Results

You can use the Validation Results to troubleshoot orchestration workflows before you publish and deploy them. To learn how to use the Validation Results, see the *SBM Composer Guide*.

Following are some typical situations that generate error and warning validation messages for orchestration workflows:

- You did not map a required data element or provide a default value for it in the **Select a Source** popup. (Error)
- You failed to enter a required expression or you entered an invalid expression, for example, in the **Target** section of a **Calculate** step or in the **Rule** section of a **While** step. (Error)
- You did not provide a compensation handler for a scope. (Warning)
- You did not specify a default value for a working data element. (Warning)

**Important:**

SBM 10.1.3 introduced improved support for multi-relational, multi-user, multi-group, and multi-select fields. These changes may cause validation errors in some orchestrations that use multi-type fields when you open the process app in SBM Composer.

Prior to 10.1.3, multi-relational fields were passed as an array of strings in the EventNotice definition. In 10.1.3 and higher, the EventDefinition passes a new Complex Type—`Multi_Type`—which consists of an array of "Items", where "Item" is a string.

For example, the statement:

```
STRINGLENGTH(EventNotice\Extension\MyRelationalField[1])>0
```

Should be expressed as follows after 10.1.3:

```
STRINGLENGTH(EventNotice\Extension\MyRelationalField\Item[1])>0
```

If you receive validation errors, review the EventNotice and add `\Item` manually if it has not been automatically upgraded by SBM Composer.

## Troubleshooting Orchestrations Using the Common Log Viewer

This section demonstrates how to work with Common Log Viewer messages to troubleshoot, or debug, orchestration workflows. (The Common Log Viewer is described in the *SBM Composer Guide*.)

**Note:** Unhandled Web service faults are the most common causes of orchestration workflow failures. (See Using the Scope, Throw, and Compensate Steps to Handle Faults From Web Services [page 96].)

If a process app that contains an orchestration workflow does not perform properly, there might be a problem with the orchestration workflow. You can use the Common Log Viewer to troubleshoot orchestration workflow-related runtime errors. The entries in the Common Log Viewer provide an audit trail that you can use to diagnose these problems.

For example, you can determine why incorrect values were copied to working data elements or **Service** step source elements during the execution of **Calculate** steps. To do this, turn Debug Logging on, and select **User messages only** in the **Message Filter** dialog box. The messages, which begin with "copying value," will show the actual values that the **Calculate** steps copied.

**Note:** If a **Calculate** step is copying an XML element, the message will show the value in XML format.

## Web Service Faults

External Web services are called by **Service** steps in orchestration workflows by sending request SOAP messages. (To the SBM Orchestration Engine, the SBM Application Engine is an external Web service.) Web services also reply with SOAP messages. If an error occurs during the processing of a request, a response SOAP message is returned that contains a SOAP fault. Because orchestration workflows are actually Web services that are invoked by other SBM components, they send and receive SOAP messages as well. (See About SOAP Messages [page 36].)

In SBM, the SBM Application Engine invokes a synchronous orchestration workflow with a SOAP message that is sent to the SBM Orchestration Engine. The SBM Orchestration Engine returns a response SOAP message to the SBM Application Engine at the **End** step.

The SBM Application Engine or an external event invokes an asynchronous orchestration workflow with a request SOAP message that it sends to the Event Manager. The Event Manager relays the message to the SBM Orchestration Engine. For asynchronous orchestration workflows, no response is returned.

During the execution of an orchestration workflow, SOAP messages returned from an external Web service might contain a SOAP fault. The first step in debugging an orchestration workflow using the Common Log Viewer is to look for errors that indicate a SOAP fault. The last error that occurred in the orchestration workflow sequence usually contains the most useful information.

Following are some typical situations that generate error messages for orchestration workflows in the Common Log Viewer:

- You tried to pass an invalid value to the Web service.
- You did not specify the correct path for the WSDL.
- You failed to enter a required expression or you entered an invalid expression, for example, in the **Target** section of a **Calculate** step or in the **Rule** section of a **While** step.

## Debugging Orchestration Workflows

This section presents the steps that you should take to debug the **Service** step. The example demonstrates one of the most common errors that cause the execution of an orchestration workflow to stop. This problem occurs when a Web service returns a response SOAP message that contains a SOAP fault during execution of the workflow.

**To debug orchestration workflows using the Log Viewer:**

1. In the Common Log Viewer, click **Refresh** to see the latest messages.

2. On the **Overview** tab, look for errors in any of the orchestration workflows invoked by the application workflow (project) that you submitted to.

   Errors are indicated by numbers in the first position within the brackets next to the name of the application workflow. For example, `UserVerificationOWF [1/0/0/7]` indicates that there is one error messages, no warning messages, no info messages, and seven debug messages in the Common Log Viewer for this orchestration workflow.

   **Note:** If there are no error messages, you should also look at any other messages. Find the message with the most severe logging level that is the farthest in the orchestration workflow sequence.

3. Select the orchestration workflow in the list, and then click the **Details** tab.

   In the example, you would click `UserVerificationOWF`.

4. Locate any error messages on the list.

   You would usually look for the error that occurred during the latest run, that is, the one with the highest number.

   For example, the following series of messages can be displayed when an orchestration workflow is executed. (By default, all Common Log Viewer messages appear in this format.) From the messages, you can see that something went wrong with **Service** step `VerifyUser`.

| Date | Run | Associated element | Text |
|---|---|---|---|
| D 2/20/ 2… | 1 | Asynch | The Orchestration Engine received the following message to invoke... |
| D 2/20/ 2… | 1 | VerifyUser | A Web service is being invoked at Service step VerifyUser, and the… |
| D 2/20/ 2… | 1 | VerifyUser | A Web service was invoked at Service step VerifyUser, and now the… |
| ✖ 2/20/ 2… | 1 | VerifyUser | A fault occurred during the execution of the orchestration workflo… |

5. Right-click the error message row, and then select **Show Message** on the menu.

6. The **Message Detail** dialog box opens and displays the entire error message. (Steps 7 and 8 relate to Service steps only.)

   The following is part of the example error message. Note that the message (in bold text) indicates an invalid User ID or password.

```
The fault is:  <?xml version="1.0" encoding="UTF-16"?><AEWebservicesFault
xmlns="urn:sbmappservices72"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ae="urn:sbmappservices72" xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#"
xmlns:diag="urn:SerenaDiagnostics" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
Invalid User ID or Password</AEWebservicesFault>.
```

7. In the **Message Detail** dialog box, click the **Previous** button to display the `"A Web service was invoked"` message, and then locate the SOAP fault, shown below in bold text.

   ```
   A Web service was invoked at Service step VerifyUser, and now the
   Orchestration Engine is receiving the following message:

       <SOAP-ENV:Envelope
       xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'
       xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
       xmlns:ae='urn:sbmappservices72'
       xmlns:c14n='http://www.w3.org/2001/10/xml-exc-c14n#'
       xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
       xmlns:wsse='http://docs.oasis-open.org/wss/2004/01/oasis-2004
       01-wss-wssecurity-secext-1.0.xsd'
       xmlns:wsu='http://docs.oasis-open.org/wss/2004/01/oasis-20040
       1-wss-wssecurity-utility-1.0.xsd'
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'><SOAP-E
       NV:Header></SOAP-ENV:Header><SOAP-ENV:Body><SOAP-ENV:Fault
       xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'><f
       aultcode>SOAP-ENV:Server</faultcode><faultstring>Invalid User
       ID or Password</faultstring><detail><ae:AEWebservicesFault
       xmlns:ae='urn:sbmappservices72'>Invalid User ID or
       Password</ae:AEWebservicesFault></detail></SOAP-ENV:Fault></S
       OAP-ENV:Body></SOAP-ENV:Envelope>
   ```

8. Often a SOAP fault is returned because some invalid data was sent to the Web service. To see the invalid User ID or Password that was sent by the **VerifyUser** Service step call to the SBM Application Engine, click **Previous** to view the `"A Web service is being invoked"` message. The values that were sent with the Web service call are shown in bold text.

   ```
   A Web service is being invoked at Service step VerifyUser, and the
   Orchestration Engine is sending the following message:

       <env:Envelope
       xmlns:env='http://schemas.xmlsoap.org/soap/envelope/'
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'><env:Header><n:S
       ecurity SOAP-ENV:mustUnderstand='0'
       xmlns:n='http://docs.oasis-open.org/wss/2004/01/oasis-200401-
   ```

```
wss-wssecurity-secext-1.0.xsd'
xmlns:SOAP-ENV='null'></n:Security></env:Header><env:Body><de
faultNS1:IsUserValid
xmlns:bpws='http://schemas.xmlsoap.org/ws/2003/03/business-pr
ocess/' xmlns:defaultNS='urn:sbmappservices72'
xmlns:defaultNS1='urn:sbmappservices72'
xmlns:ns8='urn:sbmappservices72'><ns8:auth><ns8:userId>ZZZZ</ns
8:userId><ns8:password>XXXX</ns8:password></ns8:auth><ns8:log
inId xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'
xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
xmlns:ns='http://www.eclipse.org/alf/schema/EventBase/1'
xmlns:s='http://www.eclipse.org/alf/schema/EventBase/1'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:type='xsd:string'
xmlns:xsd='http://www.w3.org/2001/XMLSchema'>XYZ</ns8:loginId
></defaultNS1:IsUserValid></env:Body></env:Envelope>
```

9. To go to the source of the problem, right-click in the error message row, and then select **Show Associated Element** on the menu.

   SBM Composer opens the appropriate design element in the workflow editor and selects the design element associated with the error.

   In the example, SBM Composer opens **UserVerificationOWF** in the workflow editor and selects the **VerifyUser** Service step.

10. Correct the problem in the design element. (If the error was caused by sending invalid data, configure the Service step to provide the correct data.)

    In the example, you would enter a valid User ID and Password in the **userID** and **password** data elements under **auth**.

## Troubleshooting Orchestrations Using Error Messages

This section contains a list of error messages. Each entry in the list provides the actual text of the message, an explanation of its cause, and a suggested solution. These messages are likely to appear in the Common Log Viewer, but they might also be displayed in error messages boxes or in other areas of SBM Composer.

```
Execution of the orchestration workflow is stopped because the size of the Web
service response has exceeded the allowable limit of {0} at Service step {1}.
```

{0} is the maximum allowable size, and {1} contains the name of the associated **Service** step.

**Explanation:** A Web service response is limited to a certain size in Solutions Business Manager. If the response to a Web service that is invoked by means of a **Service** step in an orchestration workflow exceeds this limit, the orchestration workflow is stopped. In addition, an error message containing a SOAP fault appears in the Common Log Viewer for the associated **Service** step. This error message also indicates the maximum allowable size for the response.

**Solution:** To handle the fault, you should place the **Service** step within a **Scope** step and catch the fault in the **Catch All** branch of the **FaultHandler**. Also, you should consider reducing the size of the response SOAP message's payload by redesigning the way you invoke the **Service** step. For example, you could add filtering if the Web service supports it.

## Retrying Failed Asynchronous Events

The ability to manually retry or reprocess failed asynchronous orchestration events is critical in situations where the processing of these events must be guaranteed 24 hours a day, 7 days a week. An integration between SBM and an external system that uses an event emitter mechanism is an example of this scenario. In this type of integration, the external system is responsible for raising the event, while SBM is responsible for processing it. It is possible that the event will not be processed successfully in the first attempt. The manual reprocessing of failed events is a recovery mechanism for SBM data that gets out-of-sync.

For example, you might have an external system that emits events to Service Support Manager, with the expectation that SBM will process these events. For numerous reasons, Service Support Manager may fail to process these events on a first attempt; the user needs to know about the failures so he or she can reprocess them manually.

**To identify and retry events:**

1. Open the SBM Work Center error report.
2. Review the failure information to determine whether you want to retry the event.
3. Open SBM Application Repository.
4. In the navigation pane, from the Orchestration Engine view, select **Event Manager Log**. In the list of asynchronous events that is displayed, failed events have EVENT_FAILURE in the **Status** column. (You can create a filter to show only failed events. For instructions, see the *SBM Application Repository Guide* or online help.)
5. To ensure that an event is the same event you investigated in the report before you retry it:
   a. Select the event.
   b. Click **Event Summary** at the top of the list, or right-click the event and then select **Event Summary**.
   c. Check that the event ID from the report matches the event ID shown in the **Event Summary** window.
6. To retry one or more events:
   a. Select the event or events and then click **Retry Event(s)** at the top of the list. Alternately, right-click the events and then select **Retry Event(s)**. This command will reprocess all failed orchestration workflows that were invoked by each selected event. Orchestration workflows that succeeded or have not yet completed will not be reprocessed.
   b. Click **OK** in the confirmation message that appears.

## Limitations on WSDL Files

If you are having problems with the orchestration workflows in your process apps, it might be because the WSDL files you are using are not fully supported by SBM Composer.

Although SBM Composer supports most WSDL files, it has some limitations. Contact your Web service provider to determine if any of the following restrictions apply:

- Only SOAP encoding may be used (not REST).
- Multi-part WSDL files are not supported.
- The WSDL must be WSI-BP (WS-I Basic Profile) compliant.
- Document/Literal is fully supported. Experimental support is available for the following binding styles:
  - Document/Encoded
  - RPC/Literal
  - RPC/Encoded
- HTTP binding is partially supported:
  - HTTP/POST is supported (experimental)
  - HTTP/GET is not supported
- Attachments are not supported.
- Recursive elements or types cannot be imported into an application; however, you can import a WSDL that contains recursive elements into an orchestration workflow.
- Derived complex types are not completely supported in application workflows; however, they can be used in orchestration workflows.
- Operation overloading, where the same name can apply to different operations in different situations, is not supported.
- Attribute references are not completely supported.
- WSDLs might fail to import if they are not formatted correctly, or if they contain functionality that is not supported by SBM Composer.
- The `<choice>` element is not supported.
- Creating a base type in SBM Composer and then overriding it with any of its `Extension` types is not supported.

  For example, when you create "`SearchRecord`" in the following sample code, you get a child element named "`record`" of type "`sObject`". If this type is extended by other types, such as "`Employee`" or "`Hardware`", you will not be able to change the type of "`record`" to "`Employee`" type on the **Data Mapping** tab of the orchestration workflow Property Editor. You can, however, create a working data element of type "`Employee`".

  ```
  <complexType name="SearchRecord">
  <sequence>
  <element name="record" type="ens:sObject"/>
  </sequence>
  </complexType>
  ```

- Creating arrays of varying lengths as Web service inputs is not supported. For example, you can create an array on the **Data Mapping** tab of the orchestration workflow Property Editor if you know how many records to create. However, if you are unable to determine the number of records at design time, SBM cannot create such an array.
- All WSDL definitions (`wsdl:service`, `wsdl:binding`, `wsdl:portType`, `wsdl:message`, and `wsdl:types`) must be in the same namespace; splitting WSDL definitions into more than one namespace is not supported, even if `wsdl:import` is used. However, the XSD schemas that are declared or imported into the `wsdl:types` section can be in namespaces that are different from the WSDL definitions namespace.

## Debugging for Development and Support

Advanced options for debugging, such as setting the logging level to trace, are available in Application Repository. Refer to the "Logging" section of *SBM Application Repository Guide* for more information.

# Chapter 8: Renew Utility

**Restriction:** This information pertains only to SBM On-Premise installations.

This chapter describes the orchestration engine renew utility, which enables you to remove and restore orchestration workflow data from the database.

The renew utility is primarily used during an upgrade, but it can also be used if the orchestration engine portion of an undeploy operation fails. For example, if the Application Engine portion of the undeploy succeeds, but fails for the orchestration engine, invalid deployments may remain in the database and consume resources that could be used elsewhere at runtime. Or, if an orchestration requires several events to start and it then stops (for whatever reason), several events may be saved in the database. The renew utility can clear these from the database in that scenario. In prior releases, the utility was used as part of routine maintenance, but that is no longer applicable under ODE because ODE does not accumulate deployments and runtime data the same way.

- Running Renew [page 205]
- Extended Character Handling [page 207]
- Registry Access Restriction Handling [page 207]
- Problematic Server Configurations [page 207]
- Renew Commands [page 208]

## Running Renew

The `renew.jar` file is located in the *`installDir`*`\SBM\Common\Misc\renew` directory.

The following procedure describes how to prepare for and run the renew utility. For a description of the commands and their usage and output, see Renew Commands [page 208].

**Important:** Renew must be run locally from the Orchestration Engine server. It uses libraries from the Orchestration Engine server Tomcat installation and uses the Windows registry to find that installation.

Renew is a command-line Java application, and requires the JRE in the Windows "Path" system variable. You can use the JRE and create a small batch file that surrounds your particular usage of renew. The following is an example of the file content, and shows the path to the JRE in the default SBM directory:

```
Setlocal
set JAVA_HOME=c:\Program Files\Serena\SBM\Common\jre1.8
path=%path%;%JAVA_HOME%\bin
java -Dfile.encoding=UTF-8 -jar renew.jar
endlocal
```

Modify and use the provided batch files located here:

```
installDir\SBM\Common\Misc\renew\renew_templates
```

**To run renew:**

**Important:** Back up your Orchestration Engine database before you run renew. For production systems and complex situations, it is a best practice to perform a trial run on a test system using a backup of your target system.

1. Obtain the following information about your SBM system:
   - The number of environments you have
   - The number of Orchestration Engine servers you have
   - The environment that points to each Orchestration Engine server
   - The database that is used for each Orchestration Engine server
   - An SBM user name and password that can access the environments you need to renew
   - A database user name and password that can be used to log into the Orchestration Engine database you need to renew, with privileges to list and deploy process apps.
2. Determine which Orchestration Engine server you are going to renew.
3. Determine which environments use the Orchestration Engine server you are going to renew. (There is typically only one, but it is a good idea to check.)
4. Make sure the endpoints in the environment that uses the Orchestration Engine server are correct. Changes are only picked up on deployment, and renew will use the latest.
5. If extended characters were used in the names of environments, process apps, orchestrations, or orchestration workflows, follow the procedures in Extended Character Handling [page 207].
6. Run `renew -listenvs` to list the environments in your system and make sure you are working with the correct environment. This command can warn you about certain problematic server configurations.
7. Run `renew -report` to view information specific to that environment, such as the process apps, target servers, endpoints, and orchestrations in the environment. This command can warn you about certain problematic server configurations.

   **Important:** Check whether you have a problematic server configuration where SBM Application Engine or Orchestration Engine servers are shared across environments. Do not continue unless you are sure that this does not apply to your system or unless you understand the consequences. (For details, see Problematic Server Configurations [page 207].)
8. Run `renew -redeploy` for each of the environments associated with the Orchestration Engine server you are renewing.
9. Allow access to the SBM system.

## Extended Character Handling

SBM uses the UTF-8 character set; for correct operation it is always necessary to specify this character set when running renew. It is also necessary that the correct display language is used so you can specify renew parameters with the appropriate language symbols.

**To ensure that UTF-8 is specified as the default file encoding:**

1. Create a Windows batch file using UTF-8 encoding.
2. Call renew from this batch file.
3. Redirect the renew output to a file.
4. View the output file with a UTF-8 enabled program that can display the appropriate characters.

Example:

```
java -Dfile.encoding=UTF-8 -jar renew.jar -report -environment "Default Environment"
-username "user" -password "pwd" > report.out.txt
```

**To set the correct language:**

1. Open the Windows Control Panel.
2. Select **Region and Language** (this label could vary depending on your Windows version.)
3. Navigate to the language settings and select the language you are using with SBM.
4. Reboot your machine if necessary.

## Registry Access Restriction Handling

When a Windows user does not have access to the registry on the machine where the renew utility is run, the `-sbmdir` option must be used with each command.

**For example:**

```
java -Dfile.encoding=UTF-8 -jar renew.jar -report -environment "Default Environment"
-username "user" -password "pwd" -sbmDir "C:/Program Files/Serena/" > report.out.txt
```

## Problematic Server Configurations

SBM Application Repository allows you to create certain server configurations that are not recommended and that could make it difficult to use all of the renew features. In particular, you can create overlapping environments and multiple Orchestration Engine target servers. The following sections discuss these configurations.

### Overlapping Environments

In a correctly configured SBM system that has multiple environments, each environment will have its own distinct Application Engine server, and Event Manager server, and an Orchestration Engine server. The Orchestration Engine and Event Manager servers are known to Application Repository as "target servers." The Orchestration Engine installation and database is represented by **BPEL Server** on the **Target Servers** tab. The Event Manager server is represented by **System Event Manager**.

Application Engine and the target servers should be unique to a specific environment so environments do not overlap. However, you can create overlapping environments, because Application Repository lets you reuse the same server in a different environment. This is not the intent of environments, and in most cases it is undesirable and unintended. It means the server is not fully owned by a particular environment and cannot be managed in isolation. You could have created such environments accidentally by using the **Clone** operation, or could have created them for testing purposes. It is strongly recommended that you clean up such environments before you run renew.

Renew can be used with overlapping environments in a limited capacity, but it is necessary to understand your configuration thoroughly and understand the following limitations of renew:

- Renew clears the Orchestration Engine database. All environments that point to this database will require orchestrations to be redeployed.
- Renew attempts to redeploy all of the orchestrations from the Application Engine defined in the environment. If the Application Engine is shared, renew may deploy more orchestrations than you expect.
- Renew uses the specified environment to bind the endpoints. If the orchestration was not originally deployed through that environment, the endpoint might not be defined or might have the wrong value.

Using renew to redeploy when there are overlapping environments is problematic and is not recommended. You can use renew to clear the Orchestration Engine database but to ensure correct results, you should use Application Repository to manually redeploy each of the process apps from the overlapping environments from which they were originally deployed.

By default, renew tries to detect overlapping environments and will ask for confirmation before redeploying. (If you use the `-nowarning` option with `renew -redeploy`, no confirmation warning is presented.) The `-listenvs` command also detects overlapping environments.

**Note:** Overlapping can only be detected if the same server name is used across the environments. If different names are used (for example, "localhost," an IP address, and a DNS name), overlapping is not detected, even if these names point to the same physical server. Make sure you thoroughly understand your system configuration.

## Renew Commands

The following renew commands are available:

- -h or -help [page 209]
- -listenvs [page 210]
- -report [page 211]
- -cleanupScheduledReports [page 212]

- -clearCommonLog [page 213]
- -clearEventLog [page 214]
- -redeploy [page 215]
- -restartSSFIndexUpdate [page 218]

The success or failure of these commands can be tested in a Windows batch file using "not errorlevel 1," which returns true if the error level was 0. An error level of 0 means the command executed successfully; an error level of 1 or greater means the command failed. For example:

```
java -Dfile.encoding=UTF-8 -jar renew.jar -listenvs -username "admin" -
password "password" -arAddress "http://localhost:8085"
if not errorlevel 1 goto END
echo An error occurred running renew
:END
```

The following sections describe the commands and provide example output.

## -h or -help

Displays the command line arguments. This information is also displayed if no command or an invalid command is specified.

```
 Renew [-clearCommonLog [-nowarning] [-dbUrl <arg>]
[-dbUsername <arg>] [-dbPassword <arg>]]
 Renew [-clearEventLog [-nowarning] [-dbUrl <arg>]
[-dbUsername <arg>] [-dbPassword <arg>]]
 Renew [-orchStat -action <arg> [-startDate <arg>] [-endDate <arg>]
[-dbUrl <arg>] [-dbUsername <arg>] [-dbPassword <arg>]]
 Renew [-redeploy [-environment <arg>] [-nowarning] -username <arg>
-password <arg>
                [-arAddress <arg>] [-log [<file>]] [-sbmDir <arg>]
                [-redeployInFile <file>] [-failOutputFile <file>] ]
 Renew [-listenvs [-endpoints] -username <arg> -password <arg>
                [-arAddress <arg>] [-log [<file>]] [-sbmDir <arg>] ]
 Renew [-report -environment <arg>  [-targetservers] [-endpoints] [-processapps]

                [-arAddress <arg>] [-orchestrations] [-log [<file>]] [-sbmDir <a
rg>] ]
usage:
 -action <arg>                Action for -orchStat command(migrate or
                              purge).
 -arAddress <arg>             Application Repository Address. It can be
                              omitted and in this case it is built using
                              data from renew.properties. Example:
                              "http[s]://server:port"
 -cleanupScheduledReports     Delete orphaned data from TS_NOTIFICATIONS,
                              TS_NOTIFICATIONEVENTS, and TS_NOTIFICATIONSUBSCRIPTIONS tables.
 -clearCommonLog              Truncate all data from CL_LOG and
                              CL_CONTEXT_VALUE tables
 -clearEventLog               Truncate all data from EL_EVENT,
                              EL_EVENT_PROCSNG_DATA,
                           EL_EVENT_SRVC_FLW_PROCSNG_DATA,
                           EL_EVENT_SERVICE_FLOW and EL_EVENT_LOG_MESSAGE
                           tables
 -dbPassword <arg>         Database user password.
 -dbUsername <arg>         Database user name.
 -endDate <arg>            End date for orchStat action in format
                           MM/dd/yyyy(e.g. 01/21/2016)
 -endpoints                List of endpoints of particular environment.
 -environment <arg>        Environment name. If omitted all environments
                           are processed.
 -exportOnly <arg>         Path to export applications for manual
                           processing (export only "redeploy" mode)
 -failOutputFile <arg>     Output file for failed applications (users,
                           environments) list.
 -h                        help
 -help                     help
 -listenvs                 List of all environments.
 -log <file>               Enable logging to the console or to the file
                           named <file>.
 -nowarning                Don't warn.
 -orchestrations           List of orchestrations of particular
                           environment.
 -orchestrationsOnly       Used with redeploy command to exclude event map
                           deployment.
 -orchStat                 Migrate data from eventlog tables or purge
                           orch statistic logs(depends on action)
 -password <arg>           SBM user password.
 -processapps              List of processes applications of particular
                           environment.
 -redeploy                 Redeploy all orchestrations.
 -redeployInFile <arg>     Input file which contains list of apps(users,
                           environments) which need to be redeployed.
 -report                   Get information about environments.
 -restartSSFIndexUpdate    Restart Work Center Search Engine index update
                           process
 -sbmDir <arg>             SBM location. It is required when registry
                           access is restricted. Example "C:/Program
                           Files/Serena/"
 -startDate <arg>          Start date for orchStat action in format
                           MM/dd/yyyy(e.g. 01/21/2016)
 -targetservers            List target servers of particular environment.
 -username <arg>           SBM user name.
```

## -listenvs

Lists all environments in Application Repository.

Example output with no overlapping environments (see Overlapping Environments [page 207] for details):

```
java -Dfile.encoding=UTF-8 -jar renew.jar -listenvs -username "admin" -
password "password"
-arAddress "http://localhost:8085"
```

```
          Environments :
               Default Environment
          Done.
```

Example output with overlapping environments:

```
java -Dfile.encoding=UTF-8 -
jar renew.jar -listenvs -username "admin" -password "password" -arAddress
"http://localhost:8085"

Environments :
     Default environment,
     clone1,
     clone 2

WARNING: Target servers shared across environments.
         Using renew with this configuration requires special care !!!
         DO NOT PROCEED WITHOUT READING THE DOCUMENTATION.

DS, [JBPM_BPEL->http://localhost:8085] used by : [Default Environment][clone1][clone 2]
DS, [ALF->http://localhost:8085] used by : [Default Environment][clone1][clone 2]
AE, [AE->http://localhost:80/gsoap/gsoap_ssl.dll?sbminternalservices72] used by : [Default
Environment][clone1][clone 2]
Done.
```

### -report

Gets information about a particular environment. If you specify no options, only orchestrations are displayed.

Example output with no options:

```
java -Dfile.encoding=UTF-8 -jar renew.jar -report -environment "Default Environment"
-username "admin" -password "password" -arAddress "http://localhost:8085"

Orchestrations:
   Issue Defect Management, IDMOrchestrations, AddSCMAssociations
   OrchestrationTest, SynchOrch, Synch
   OrchestrationTest, SynchOrch, Asynch
   OrchestrationTest, SynchOrch, SynchTimeout
   OrchestrationTest, SynchOrch, SynchFault
   OrchestrationTest, SynchOrch, AsynchFault
   OrchestrationTest, SynchOrch, SynchError
   OrchestrationTest, TestOrch, NewTestOrchWorkflow
   OrchestrationTest, TestOrch, NewTestOrch2Workflow
   Incident Management, IncidentOrchestration, CloseChildIncidents
   Incident Management, IncidentOrchestration, ReOpenChildIncidents
   Incident Management, IncidentOrchestration, IssueDevelopmentComplete
   OeOnly, OeOnlyOrch, OrchestrationWorkflow
   AAS_TEST_1, AAS_One, AAS_OneWorkflow
   AAS_TEST_1, AAS_SYNC_1, AAS_SYNC_1Workflow2
   Change Approval Requests, CAR, IssueDevelopmentComplete
   Change Approval Requests, CAR, InCABReviewAddReviewerWorkflow
Done.
```

Example output with selected options:

```
java -Dfile.encoding=UTF-8 -jar renew.jar -report "Default Environment"
-targetservers -endpoints -processapps -username "admin" -password "password"
-arAddress "http://localhost:8085"

Target Servers :
   AE, [Default Application Engine] : http://TEST/gsoap/gsoap_ssl.dll?sbminternalservices72
   DS, [Default Event Manager Server] : http://TEST:8085
   DS, [Default BPEL Server] : http:// TEST:8085
   DS, [test] : http://localhost:8085

Endpoints :
   RestWrapperService, http:// TEST:8098/restwrapper
   sbmappservices72, http://TEST:80/gsoap/gsoap_ssl.dll?sbmappservices72
   sbmadminservices72, http://TEST:80/gsoap/gsoap_ssl.dllsbmadminservices72

Process Applications :
   Global Process App (eval)
   Issue Defect Management
   Incident Managment
   Change Approval Requests
   OrchestrationTest
   AAS_TEST_1
   AEOnly
   OeOnly
Done.
```

## -cleanupScheduledReports

Deletes orphaned scheduled report records in TS_NOTIFICATIONEVENTS, TS_NOTIFICATIONS, and TS_NOTIFICATIONSUBSCRIPTIONS. For example, if an entry for a scheduled report is deleted from TS_NOTIFICATIONS, but a related record exists in TS_NOTIFICATIONEVENTS, you can use this command to clean up both tables.

The command performs the following actions:

- Deletes scheduled reports records (with TS_SENDTYPE=256) from the TS_NOTIFICATIONS table that reference a report that no longer exists. In other words, delete records in which TS_NOTIFICATIONS.TS_REPORTID references a TS_REPORTS.TS_ID that does not exist.

- Deletes scheduled reports records (with TS_SENDTYPE=256) from the TS_NOTIFICATIONS table that do not have a related record in the TS_NOTIFICATIONEVENTS table (TS_NOTIFICATIONID column).

- Deletes scheduled reports records (with TS_SENDTYPE=256) from the TS_NOTIFICATIONEVENTS table in which TS_NOTIFICATIONEVENTS.TS_NOTIFICATIONID references a TS_NOTIFICATIONS.TS_ID that does not exist.

- Deletes scheduled reports records from the TS_NOTIFICATIONSUBSCRIPTIONS table in which TS_NOTIFICATIONSUBSCRIPTIONS.TS_NOTIFICATIONID references a TS_NOTIFICATIONS.TS_ID that does not exist.

There are two ways to execute this command using the provided batch template files:

- Run `cleanupSR.bat`:

  ```
  installDir\SBM\Common\Misc\renew\renew_templates\cleanupSR.bat
  ```

  This command deletes orphaned records related to scheduled reports from TS_NOTIFICATIONEVENTS and TS_NOTIFICATIONS.

  In this case, the `-dbUsername` and `-dbPassword` options are omitted. In their place, the credentials are taken from the `renew.properties` file located here:

  ```
  installDir\SBM\Common\tomcat\server\default\conf
  ```

- Run `cleanupSR_dbCredentials.bat`:

  ```
  installDir\SBM\Common\Misc\renew\renew_templates\cleanupSR_dbCredentials.bat
  ```

  This command deletes orphaned records related to scheduled reports from the TS_NOTIFICATIONEVENTS, TS_NOTIFICATIONS, and TS_NOTIFICATIONSUBSCRIPTIONS tables using the database credentials that are specified in the batch file.

Example output:

```
java -Dfile.encoding=UTF-8 -jar renew.jar -cleanupScheduledReports
-dbUsername "sa" -dbPassword "password"

WARNING: This will delete some data from TS_NOTIFICATIONS and TS_NOTIFICATIONEVENTS tables.
Are you sure? [Y/N]:Y
Done.
```

## -clearCommonLog

Deletes all records in the CL_CONTEXT_VALUE and CL_LOG common log tables. If you enabled debug level logging for a process app and forgot to disable it, the common log could grow significantly and you may not be able to execute any common log operations in Application Repository or SBM Composer. You can use this command to clear the common log tables in this scenario.

**Important:** Before you run the `renew -clearCommonLog` command, it is highly recommended that you stop the SBM Tomcat service first.

There are two ways to execute this command using the provided batch template files:

- Run `clearCL.bat`:

  ```
  installDir\SBM\Common\Misc\renew\renew_templates\clearCL.bat
  ```

  This option clears the common log tables using the database credentials that are specified in:

  ```
  installDir\SBM\Common\tomcat\server\default\conf\renew.properties
  ```

- Run `clearCL_dbCredentials.bat`:

*installDir*`\SBM\Common\Misc\renew\renew_templates\clearCL_dbCredentials.bat`

This option clears the common log tables using the database credentials that are specified in the batch file.

Example output:

```
java -Dfile.encoding=UTF-8 -jar renew.jar -clearCommonLog
-dbUsername "sa" -dbPassword "password"

WARNING: This will permanently delete all Common Logger data. Are you sure? [Y/N]:Y
Done.
```

## -clearEventLog

Deletes all records in the EL_EVENT, EL_EVENT_PROCSNG_DATA, EL_EVENT_SRVC_FLW_PROCSNG_DATA, EL_EVENT_SERVICE_FLOW and EL_EVENT_LOG_MESSAGE event log tables. If you have not purged the event log in some time and a large amount of records have accumulated, you might not be able to delete event log records using Application Repository. You can use this command to clear the event log tables in that scenario.

**Important:** Before you run the `renew -clearEventLog` command, it is highly recommended that you stop the SBM Tomcat service first.

There are two ways to execute this command using the provided batch template files:

- Run `clearEL.bat`:

  *installDir*`\SBM\Common\Misc\renew\renew_templates\clearEL.bat`

  This option clears the event log tables using the database credentials that are specified in:

  *installDir*`\SBM\Common\tomcat\server\default\conf\renew.properties`

- Run `clearEL_dbCredentials.bat`:

  *installDir*`\SBM\Common\Misc\renew\renew_templates\clearEL_dbCredentials.bat`

  This option clears the event log tables using the database credentials that are specified in the batch file.

Example output:

```
java -Dfile.encoding=UTF-8 -jar renew.jar -clearEventLog
-dbUsername "sa" -dbPassword "password"

WARNING: This will permanently delete all Common Event Log data. Are you sure? [Y/N]:Y
Done.
```

## -redeploy

Redeploys the orchestrations and eventmap from the process apps that are deployed to a particular environment. It does this by retrieving the environment's deployed process apps from the environment's Application Engine server, and then retrieving the target server and endpoint configuration from the environment. For each process app, it redeploys the orchestration workflows and eventmap to the target Orchestration Engine, fixing the various endpoints consumed by those workflows.

You can run this command multiple times against an environment, even if you did not run renew -clear earlier. It will simply redeploy the orchestration definitions deployed to the environment and cached by the Application Engine to the Orchestration Engine, creating a new internal version within the Orchestration Engine. Running renew -redeploy does not affect the deployment records that Application Repository keeps.

**Note:**

- Because renew retrieves the endpoints from Application Engine, it is possible that they were modified in Application Repository since the last deployment. After running renew, the deployed orchestrations will have the endpoints that were last deployed to Application Engine, unless you include the `-useEnvEndpoints` option as well.
- See Overlapping Environments [page 207] for important information about redeploying in configurations with overlapping environments.

Example output with overlapping environments, without the `-nowarning` option:

```
java -Dfile.encoding=UTF-8 -jar renew.jar -redeploy -environment "Default Environment"
-username "admin" -password "password" -arAddress "http://localhost:8085"

WARNING: Target servers shared across environments.
         Using renew with this configuration requires special care !!!
         DO NOT PROCEED WITHOUT READING THE DOCUMENTATION.

DS, [JBPM_BPEL->http://localhost:8085] used by : [Default Environment][clone1][clone 2]
DS, [ALF->http://localhost:8085] used by : [Default Environment][clone1][clone 2]
AE, [AE->http://localhost:80/gsoap/gsoap_ssl.dll?sbminternalservices72] used by :
[Default Environment][clone1][clone 2]

WARNING: Are you sure? [Y/N]:y
Redeploying orchestrations ...

AAS_TEST_1 :
    AAS_One : Deployed
    AAS_SYNC_1 : Deployed
AAS_TEST_1 : Deployed

Change Approval Requests :
    CAR : Deployed
Change Approval Requests : Deployed

Incident Management :
    IncidentOrchestration : Deployed
Incident Management : Deployed
```

```
Issue Defect Management :
    IDMOrchestrations : Deployed
Issue Defect Management : Deployed

OeOnly:
    OeOnlyOrch : Deloyed
OeOnly : Deployed

OrchestrationTest :
    SynchOrch : Deployed
    TestOrch : Deployed
OrchestrationTest : Deployed
Done.
```

Example output with the `-nowarning` option:

```
java -Dfile.encoding=UTF-8 -jar rewnew.jar -redeploy -environment "Default Environment"
-nowarning -username "admin" -password "password" -
arAddress "http://localhost:8085"
Redeploying orchestrations ...

AAS_TEST_1 :
    AAS_One : Deployed
    AAS_SYNC_1 : Deployed
AAS_TEST_1 : Deployed
Change Approval Requests :
    CAR : Deployed
Change Approval Requests: Deployed

Incident Management :
    IncidentOrchestration : Deployed
Incident Management : Deployed

Issue Defect Management :
    IDMOrchestrations : Deployed
Issue Defect Management: Deployed

OrchestrationTest :
    SynchOrch : Deployed
    TestOrch : Deployed
OrchestrationTest : Deployed
Done.
```

You can also run the redeploy command with the following options:

- `-redeployInFile`
- `-failOutputFile`
- `-eventmap`
- `-useEnvEndpoints`

The `-redeployInFile` option has an argument which is an input file that contains a list of apps, users, and environments that need to be redeployed.

File example (`redeploy.xml`):

```
<RedeployList>
        <User name="admin">
                <Environment name="Dev Environment" />
                <Environment name="QA Environment">
                        <ProcessApp>QuickTest</ProcessApp>
                </Environment>
        </User>
</RedeployList>
```

Example:

```
java -Dfile.encoding=UTF-8 -jar renew.jar -redeploy -username "admin"
-password "password" -redeployInFile "redeploy.xml"
```

In this example, the redeploy is performed for all apps in the `Dev Environment` and the `QuickTest` app in the `QA Environment`.

The `-failOutputFile` option has an argument that outputs a file for failed apps, users, and environments. This output file has the same structure as `-redeployInFile`.

Example:

```
java -Dfile.encoding=UTF-8 -jar renew.jar -redeploy -username "admin"
 -password "password" –failOutputFile "failOutput.xml"
```

You can include the `-eventmap` option to redeploy event maps for deployed applications.

Example:

```
java -Dfile.encoding=UTF-8 -jar renew.jar -redeploy -username "admin"
-password "password" -eventmap
```

This is usually not required, but it can be useful in the event that data becomes obsolete. For example, if the host name is changed for either server, the event map must be redeployed.

You can use the `-orchestrationsOnly` option to redeploy only orchestrations. However, it is not recommended to use it unless you want to skip eventmap deployment and you need only orchestrations to be redeployed.

You can use the `-useEnvEndpoints` option in the event that endpoints were manually changed in Application Repository after the last deploy.

Example:

```
java -Dfile.encoding=UTF-8 -jar renew.jar -redeploy -username "admin"
-password "password" -useEnvEndpoints
```

This is not required, but it can be useful in the event the endpoints have changed. For example, if the endpoints were changed in Application Repository, this option ensures that after the redeploy, any endpoints for Application Engine Web service calls that are made from an orchestration are current.

## -restartSSFIndexUpdate

Restarts the Work Center search index update process. This enables you to restart the indexer update process without having to wait and stop and start SBM Tomcat. This is useful if a problem occurs during the index update process and new items or updated items are not being indexed. Note that this command only applies to run-time indexing, not the initial indexing process.

You can execute this command using the provided batch template file:

- Run `restartSSFUpdateProcess.bat`:

  *installDir*`\SBM\Common\Misc\renew\renew_templates\restartSSFUpdateProcess.bat`

  Before you execute the batch file, edit the file and enter the user name and password for an SBM user with remote administration privilege.

Example output:

```
java -Dfile.encoding=UTF-8 -jar renew.jar -restartSSFIndexUpdate
-username "admin" -password ""

Progress report: Initializing SSO...
initSSO() completed
{"data":null,"status":{"value":"OK","message":null,"errorCode":null}}
Done.
```

# Part 2: Advanced Orchestration Topics

This section contains the following information:

# Chapter 9: Raising External Events

**Important:** These instructions are intended for experienced software developers/integrators who are familiar with the SOAP standard and Extensible Markup Language (XML) and who have expert knowledge of the Web Service Description Language (WSDL) and of creating Web services. A working knowledge of SBM is also required.

External events can be used to link a wide range of products to orchestration workflows. This allows you to enable Web service applications to work with Solutions Business Manager.

To raise and handle external events requires the creation of a custom event definition and mapping it to your orchestration workflow.

**Note:** Information about is SBM Web services is provided in the *SBM Web Services Developer's Guide*, which is included with the product.

The following topics describe how to create, import, and map the event definition to generate your external events:

## Events Terminology and Concepts

This topic terminology and concepts pertaining to events.

**Event**

**Note:** An event was referred to as a *mashup event* or an *ALF event* in previous releases of SBM.

An event is a Web services message signaling a meaningful change from an application or external product. For example, an issue defect management application in SBM might generate an event every time a user enters a new defect, or an external product might raise an external event named `BuildCompleted`. When an event is received by the Event

Manager, the SBM Orchestration Engine is called to execute the orchestration workflow linked to the event.

Events are SOAP messages. SBM supports both HTTP and e-mail transports for event SOAP messages. The Event Manager provides RPC/literal messaging and document/literal messaging services to accept event requests. It does not support RPC/encoded messaging.

**Important:** While events can be raised using either RPC/literal or document/literal messaging format, they are defined using RPC/literal messaging format (see Defining an Event Definition [page 223].)

Extensions that come with particular event types must be defined so an orchestration workflow can access the `extension` data in the event. However, the same event can also be handled by an orchestration workflow that only understands the base event.

To accomplish this, schema restriction-based derived types are defined that allow for tightening of the value range or other restrictions on certain elements in a complex type. In this way, a type that is an `ALFEventType` can be declared that only allows, for example, an `EventType` of `ThingCreated` and an `ObjectType` of either `Thing1` or `Thing2`.

For large events (for example, events with a large amount of BASE64 data) SBM can accept compressed events that are sent in Gzip format.

**Event Manager**

The *Event Manager* is a component in Solutions Business Manager that responds to events sent by applications and products. The Event Manager calls orchestration workflows implemented as BPEL processes.

**Event Map**

The Event Manager dispatches events according to a configured map, called the *event map*. The event map relates events to the orchestration workflows that should run when an event occurs. Products that can raise events can declare their events by means of a specific WSDL format called an event definition. (See Defining an Event Definition [page 223].) The event definition can be used by a product, such as Solutions Business Manager, to construct an event map and to deploy the event map to the Event Manager. At runtime, the Event Manager receives the event, and if the event appears in a deployed event map, it invokes the associated orchestration workflow or workflows, passing on the event data.

**Orchestration Workflow**

An orchestration workflow is a sequenced arrangement of Web service calls designed using SBM Composer. Orchestration workflows combine Web services using loops and decision branches and define the way data is mapped between the Web services. The final arrangement is saved as a BPEL process. Orchestration workflows can be linked to application workflows in a process app through actions on both transitions and states. Orchestration workflows can run asynchronously using an event, or they can be called synchronously, where the action waits for the orchestration workflow to return some data. When designed to be used with an event, orchestration workflows can also be invoked by external systems.

**SBM Orchestration Engine**

The SBM Orchestration Engine is the component in Solutions Business Manager that executes orchestration workflows. Using SBM Composer, Web services can be sequenced and then executed in response to an event or by transitions in applications.

**Object**

In Solutions Business Manager, an *object* is identified in the event structure by the values of `ObjectType` and `ObjectId`. In other words, the `ObjectType` and `ObjectId` are used to refer to the object that originated the event. An event typically originates due to a change in an object. The `ObjectType/ObjectId` identifies the source of the event within the product and the `EventType/EventId` identifies the type of change that caused the event.

## Accessing the Advanced Orchestration Package

The Advanced Orchestration Package contains files that you can use as a basis to create or update your event definition.

Before you create your application-specific event definition, download the Advanced Orchestration Package `.zip` file. You will be using these files throughout the instructions to create your custom event definitions and events.

The file is available from the Start Page in SBM Composer.

Refer to the `readme.txt` file in the Advanced Orchestration Package `.zip` file for a description of the files in the package.

## Defining an Event Definition

An event definition used in SBM defines events from an application workflow. You can create your own custom event definition for generating external events.

An event definition is a WSDL file that redeclares the ALFServiceFlow service, making it specific to your event source. The process involves creating a specialized event definition schema derived from the ALF event base schema. The event definition specifies custom, tool-specific types and events. It also defines the orchestration workflow (declared as a service flow service) that handles these events.

You can create a custom event directly in SBM Composer. This works for many cases. However, for some advanced cases, you might prefer to create the event definition as a separate WSDL file and import it into SBM Composer.

**Note:** This chapter describes how to create the event definition as a separate WSDL file. For information about how to create the event definition in SBM Composer see Raising Events from External Products [page 145], Creating a New Custom Event Definition [page 69], and Importing an Event Definition File for a New Custom Event Definition [page 69].

**Event Definition Messaging and Schema**

Event definitions must use RPC/literal messaging and must be created following the definitions in `ALFEventManager?wsdl`.

The ALF event base schema from which you create the derived schema for your event definition is defined in `ALFEventBase_1.xsd`. This schema is common to all event services and is included by reference in your event definition.

## Creating a Custom Event Definition

This section provides instructions for creating a custom event definition. Using the `ExampleEventDefinition.wsdl` file as a template, you provide values that let external products generate events.

**Important:** This topic provides instructions for manually creating a custom event definition. It is more convenient to create a custom event definition in SBM Composer. For more information, see Creating a New Custom Event Definition [page 69].

**To create a custom event definition:**

1. Create an empty folder on your local system.
2. Copy the `advanced_orchestration_package.zip` file and paste it in the folder.
3. Extract the `advanced_orchestration_package.zip` file.
   - The `ALFEventManager.wsdl` file is the ALF Event Manager WSDL. Note that the `ExampleEventDefinition.wsdl` imports the `ALFEventManager.wsdl`.
   - The `ALFEventBase_1.xsd` file is the ALF event base schema.
   - `ExampleEventDefinition.wsdl` is the template for your custom event definition. The other files are explained in other topics.

     **Note:** You can also obtain a copy of the ALF Event Manager WSDL and the `ALFEventBase_1.xsd` files from the `installDir\Composer\Conf` folder.
4. Create a copy of the `ExampleEventDefinition.wsdl` file in an XML editor and rename it as your external product WSDL file, for example, `SCMProduct.wsdl`. If you ever move this file to a different folder, be sure to place a copy of the `ALFEventManager.wsdl` and the `ALFEventBase_1.xsd` files in the folder as well.
5. Edit this new file to modify it for your installation:
   a. Look for following section for event type declaration. Change the value attribute of the enumeration to your event type. Each enumeration value corresponds to different event occurring in your application.

      ```
      <xs:simpleType name="MyEventDefinitionEventType">
      <xs:restriction base="EventTypeType">
      <xs:enumeration value="MyEventDefinitionEvent"/>
      </xs:restriction>
      </xs:simpleType>
      ```

      You can define as many enumeration values as you want, for example:

      ```
      <xs:simpleType name="MyEventDefinitionEventType">
      <xs:restriction base="EventTypeType">
      <xs:enumeration value="Issue Created"/>
      <xs:enumeration value="Issue Deleted"/>
      <xs:enumeration value="Waiting for Approval"/>
      ```

```
</xs:restriction>
</xs:simpleType>
```

b. Look for the following section for object type declaration. You can define as many enumeration values as you want. However, it is recommended that you define only one object type. Change the enumeration value to your external product specific object.

```
<xs:simpleType name="MyEventDefinitionObjectType">
<xs:restriction base="ObjectTypeType">
<xs:enumeration value="MyEventDefinitionObject"/>
</xs:restriction>
</xs:simpleType>
```

c. Look for product value, product instance and product version declarations. *Product value* indicates the external product name. *Product version* is the number you want to give this event declaration. You can use this value to distinguish the events between versions of a product if the event definitions are changed in an incompatible way. If the events are completely compatible, however, you should in general not change this value. *Product instance* is the "logical location," and is determined by the particular installation of the external product in your system. It can default to what is specified in the event definition, but you can also use SBM Composer to indicate the external product from which you are expecting the event.

```
<xs:simpleType name="MyEventDefinitionProductType">
    <xs:restriction base="ProductType">
      <xs:enumeration value="MyEventDefinition"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- Derived ProductVersionType -->
  <xs:simpleType name="MyEventDefinitionProductVersionType">
    <xs:restriction base="ProductVersionType">
      <xs:enumeration value="1.0"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- Derived ProductInstanceType -->
  <xs:simpleType name="MyEventDefinitionProductInstanceType">
    <xs:restriction base="ProductInstanceType">
      <xs:enumeration value="MyEventDefinitionInstance"/>
    </xs:restriction>
  </xs:simpleType>
```

Again in all three cases, replace the enumeration values to match the values for the external product.

**Note:** The event type, object type, product, product instance, and product version values make a set of events match, and are used to determine which orchestration workflows need to be invoked.

d. Look for the section where extension data is defined. Here you can define different fields you want to send as input to your orchestration workflows. Remember you can send only one set of data using one process app tool.

```
<xs:complexType name="MyEventDefinitionCustomExtension">
    <xs:annotation>
      <xs:documentation>
        Custom Extension
      </xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="MyEventDefinitionData" type="xs:string"/>
    </xs:sequence>
    <xs:anyAttribute/>
  </xs:complexType>
```

You can define any type of schema under sequence either simple type elements as shown above or complex type.

e. Replace all remaining occurrences of "MyEventDefinition" with your product value.

6. At this point, your event definition WSDL is ready to be imported it into SBM Composer.

## Testing Events from an External Source

After deploying your process app, you can test the external event by using `sample event xml` provided in `ExampleEventDefinitionSOAPMessage.xml` using tools like SoapUI. This procedure is optional.

If you do not want to perform this test, go directly to Creating Event Client using Apache Axis2 [page 227].

**To test the event from an external source, perform the following steps:**

1. Create new project in soapUI.
2. Import your event definition WSDL. For information on creating a custom event definition, see Creating a Custom Event Definition [page 224].
3. SoapUI will generate various request XMLs for you. Choose the one that matches your application's ALFServiceFlowSOAP request. This will be displayed as `<your product value>ALFServiceFlowSOAP`.
4. Open the corresponding request in the SoapUI editor and copy paste the `<Base>` element from `ExampleEventDefinitionSOAPMessage.xml` in it.
5. Replace the event match values—namely EventType, ObjectType, Product, ProductInstance, and ProductVersion—to match your application-specific values.
6. Fill in your extension data.
7. Make sure that you set the correct endpoint pointing to the Event Manager. The URL should like this:

```
http://<server>:<port>/eventmanager/services/ALFEventManager
```

8. Raise the event by running this request. You can read about the execution in the common logger view in SBM Composer.

## Creating Event Client using Apache Axis2

This topic gives an example of using Apache Axis2 to create your services. Apache Axis2 is an enterprise-ready Web service engine that is very user-friendly and provides Web service interactions with a dynamic and flexible execution framework. For more information, refer to `http://ws.apache.org/axis2/`.

**To create a client for raising events using Doc/Lit binding:**

1. Create a new application-specific WSDL file.

   a. Make a copy of the `ExampleEventDefinitionALFEventManagerDocLit.wsdl` file. This will become your application-specific `.wsdl` file.

   b. Copy both `<xsd:schema>` elements from your application-specific event definition `.wsdl` file and paste the schema into this new `.wsdl` file where the following comment is:

      ```
      <!-- Paste schema from your event definition wsdl here -->
      ```

      In other words, open the custom event definition file that you created in Creating a Custom Event Definition [page 224] and copy the schema elements into this file. When finished, the `.wsdl` file created from `ExampleEventDefinitionALFEventManagerDocLit.wsdl` will contain three schema elements.

   c. Replace all occurrences of `MyEventDefinition` with your product value.

   Your application-specific `.wsdl` file is ready for consumption by your application.

2. If you are using Axis2 to raise external events, modify the batch file `GenerateStubClassesFromAxis2.bat` and change the name of the `.wsdl` file. Remember to set the Axis2 home.

3. Once the classes are generated, you can use the stub class to raise events.

4. The URL for this endpoint should in the form `http://<server>:<port>/eventmanager/services/ALFEventManagerDocLit`.

## Raising an External Event through E-mail

This section contains general instructions for raising external events through e-mail using the `ExampleEventDefinitionSOAPMessage_forEmail.xml` template, which is included in the Advanced Orchestration Package. For details on creating a sample e-mail event message, see Creating a Sample E-mail Event SOAP Message [page 229].

**To raise an external event through e-mail, perform the following steps:**

1. Open the `ExampleEventDefinitionSOAPMessage_forEmail.xml` file in an XML editor. The template file is included in the Advanced Orchestration Package. (The contents of the template are shown in the next section.)
2. Provide values for the following `Body` elements. The values in italics are based on the definitions in your event definition:
   - `EventId`
   - `Timestamp`
   - *`EventType`*
   - *`ObjectType`*
   - `ObjectId`
   - *`Product`*
   - `ProductVersion`
   - *`ProductInstance`*
3. Define the data elements that you will send with the event in the `Extension` element.
4. Provide your authentication credentials in one of the following ways:
   - Use the `ns:ALFSecurity` element.

     This method is shown in the template.
   - Use a WS-Security header.
5. Specify the recipient's e-mail address in the `wsa:To` element.

6. To send the event XML in the body of the e-mail, copy and paste the contents of the XML file into the body of the e-mail, and then send it to the address in the `wsa:To` element.

   **Important:** SBM can retrieve the ALF event from either a plain text body (single part e-mail message) or the last plain text part of a multi-part message. SBM does not process single part messages that contain only HTML text or multi-part messages that do not contain plain text with the ALF event.

   Some POP3 mail servers are able to convert single part HTML messages into multi-part messages that also contain plain text equivalent HTML. If you are using this type of POP3 mail server, you can send HTML-only messages as well. For details, see solution S141688. If plain text cannot be used, then send the event XML as an e-mail attachment (see the next step).

   **Important:** Be sure that your e-mail client does not insert extra characters, such as line endings, when you send the e-mail.

7. To send the event XML as an e-mail attachment, attach the XML file to the e-mail, and then send it to the address in the `wsa:To` element. The text file attachment will be processed instead of the message body and treated as UTF-8 by default.

## Creating a Sample E-mail Event SOAP Message

This section contains instructions for creating a sample e-mail event SOAP message using the `ExampleEventDefinitionSOAPMessage_forEmail.xml` template.

The contents of the template are included here for your reference.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:ns="http://www.eclipse.org/alf/schema/EventBase/1"
xmlns:myev="http://www.example.org/MyEventDefinitionEventExtensions">
  <soapenv:Header>
    <wsa:To>mailto:eventemail@serverName?X-Service-Path=/eventmanager
    → /services/ALFEventManagerOneWayDocLit</wsa:To>
    <wsa:MessageID>urn:uuid:421A2EE66BA3A42A8C1203350641340</wsa:MessageID>
    <wsa:Action>urn:EventNotice</wsa:Action>
  </soapenv:Header>
  <soapenv:Body>
    <ns:ALFEventNoticeDoc version="1.0">
      <ns:Base>
        <ns:EventId>1</ns:EventId>
        <ns:Timestamp>2008-05-12T21:19:20.671Z</ns:Timestamp>
        <ns:EventType>MyEventDefinitionEvent</ns:EventType>
        <ns:ObjectType>MyEventDefinitionObject</ns:ObjectType>
        <ns:ObjectId>1</ns:ObjectId>
        <ns:Source>
          <ns:Product>MyEventDefinition</ns:Product>
          <ns:ProductVersion>1.0</ns:ProductVersion>
          <ns:ProductInstance>MyEventDefinitionInstance</ns:ProductInstance>
        </ns:Source>
        <ns:User>
```

```
           <ns:ALFSecurity>
             <ns:UsernameToken>
               <ns:Username>username</ns:Username>
               <ns:Password>password</ns:Password>
             </ns:UsernameToken>
           </ns:ALFSecurity>
         </ns:User>
        </ns:Base>
       <ns:Extension>
         <myev:MyEventDefinitionData>Test Data</myev:MyEventDefinitionData>
       </ns:Extension>
     <//ns:ALFEventNoticeDoc>
   </soapenv:Body>
 </soapenv:Envelope>
```

**Note:** The element prefixes (for example, `wsa:`) used in the rest of this section are simply shortcut references to the defining namespace. They are provided to help you identify the values in the template, but the prefixes are not part of the element name.

**To create a sample e-mail event SOAP message, you must update all of the bolded values in the template as follows:**

1. Set the following element values for the particular system you are using:

   - `<wsa:To>`**`mailto:event@yourdomain.com?X-Service-Path=/eventmanager/services/ALFEventManagerOneWayDocLit`**`</wsa:To>`

     The WS-Addressing `<To>` element value must contain a valid `xs:anyURI` in the format shown. The e-mail portion of the URI, in this case `eventemail@yourdomain.com`, should be set to the e-mail address that you have been provided for sending events to the system.

     **Note:** In the SBM On-Demand environment, the e-mail address you should use is usually `event@yourdomain.com`. In other words, you should replace `eventemail@serverName` in the template with `event@yourdomain.com`.

   - `<ns:Username>`**`username`**`</ns:Username>`

   - `<ns:Password>`**`password`**`</ns:Password>`

     The values of the preceding two elements should be set to the user name and password credentials for the system that you use for raising the event.

2. Set the following element values per event type, as defined in your event definition WSDL:

   - `<ns:EventType>`**`MyEventDefinitionEvent`**`</ns:EventType>`

     Set this element to the `EventType` value for this event.

   - `<ns:ObjectType>`**`MyEventDefinitionObject`**`</ns:ObjectType>`

     Set this element to the `ObjectType` value for this event.

   - `<ns:Product>`**`MyEventDefinition`**`</ns:Product>`

Set this element to the `Product` value for your event definition.

- `<ns:ProductVersion>`**`1.0`**`</ns:ProductVersion>`

  Set this element to the `ProductVersion` value for your event definition.

- `<ns:ProductInstance>`**`MyEventDefinitionInstance`**`</ns:ProductInstance>`

  Set this element to the `ProductInstance` value for your event definition.

3. Set the following element values for each event instance:

   - `<wsa:MessageID>`**`urn:uuid:421A2EE66BA3A42A8C1203350641340`**`</wsa:MessageID>`

     Set this element to a unique ID for each event sent. A UUID is suggested. If you use `ALFEventManagerOneWayDocLit` in the `<wsa:To>` element, this value is not critical since no reply is expected; however, it might be useful in the future.

   - `<ns:EventId>`*`1`*`</ns:EventId>`

     Set this element to a unique ID for each event sent. Although this is not critical, it allows you to identify the particular instance of the event, which might be useful to correlate data.

   - `<ns:ObjectId>`*`1`*`</ns:ObjectId>`

     This element identifies the instance of the object responsible for the event. Although this is not critical, a value recognized by your event definition might be useful for accessing data in your event definition to correlate data.

   - `<ns:Timestamp>`**`2008-05-12T21:19:20.671Z`**`</ns:Timestamp>`

     This element records the time the event was sent. It must be in `xsd:dateTime` format (a subset of ISO 8601). It can be set to `nil` as shown, where `xsi:` is the namespace `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`.

     ```
     <ns:Timestamp xsi:nil="true"/>
     ```

   - `<myev:MyEventDefinitionData>`**`Test Data`**`</myev:MyEventDefinitionData>`

     This element is the `Extension` data for this event, as defined by your event definition. The sample event definition only defines one element (`MyEventDefinitionData`) in its `Extension` data. Your event definition can send more complex data.

## Configuring Solutions Business Manager to Receive E-Mail Events

The Web service–based event mechanism exposed by SBM can accept events either through HTTP or via e-mail. Events through HTTP are enabled by default and require no additional configuration beyond the basic installation of SBM. Events through e-mail require that SBM be configured to access an e-mail mailbox provided by an external e-mail server. SBM Configurator provides options to configure this feature.

## Setting Up a POP3 E-mail Account

The SBM Event Manager can support receiving events only from a POP3-compliant e-mail server. To use the e-mail event feature, you must set up a dedicated e-mail account on the POP3 server, establishing the e-mail address that you want to use to accept the e-mail event requests. To configure SBM, you need to know the POP3 server host name, the port used by your POP3 server, the e-mail account name, and the e-mail account password.

**Important:** A single-part message that contains only plain text is treated as a SOAP message and accepted as an ALF e-mail event. If SBM receives a single-part message that only contains HTML, SBM cannot process it and the event is ignored. Multi-part messages are accepted, but they must contain the ALF event in plain text in order for the Event Manager to process the event.

For more information about POP3 mail service behavior, refer to solution S141688.

## Options Provided by SBM Configurator

*SBM On-Premise only*

On the **Mail Services** tab in SBM Configurator, you can enable mail settings for Application Repository and the Event Manager. Use the following fields to configure the Event Manager to receive e-mail events.

**Host**: The network name of the POP3 e-mail server you want to use to provide the e-mail inbox for the e-mailed events.

**Port**: The port used by the POP3 e-mail server. POP3 typically uses port 110. If you want to use a secure connection to the POP3 mailbox, select the **Use SSL** option. The default SSL port is 995.

**User name**: The e-mail account name for the inbox that your POP3 server provides.

**Password**: The password to the e-mail account.

## Adjusting the Configuration

If you want to adjust the configuration settings for e-mailed events after you have completed the initial mail server set up, you can change the settings in SBM Configurator at any time.

**Important:** If you apply changes while the SBM Configurator runs in **utility mode**, browser users may not be able to access the system immediately while the services are restarting. Therefore, consider applying configuration changes at a time when users are not actively using the system.

## E-mail Event Messages

To send an event by e-mail, you must construct an XML SOAP document in the correct format and send it to the event-enabled e-mail account.

Here is an example e-mail event message document:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns="http://www.eclipse.org/alf/schema/EventBase/1"
  xmlns:exam="http://www.eclipse.org/ALF/ExampleVocabulary"
  xmlns:myev="http://www.example.org/MyEventExtensions"
  xmlns:wsa="http://www.w3.org/2005/08/addressing"
  xmlns:mym="http://www.example.org/MyEventDefinitionEventExtensions">
  <soapenv:Header>
    <wsa:To>mailto yourEmailUser@yourEmailHost?X-Service-Path=/eventmanager/
→services/ALFEventManagerOneWayDocLit</wsa:To>
    <wsa:MessageID>urn:uuid:421A2EE66BA3A42A8C1203350641340</wsa:MessageID>
    <wsa:Action>urn:EventNotice</wsa:Action>
  </soapenv:Header>
  <soapenv:Body>
    <ns:ALFEventNoticeDoc version="1.0">
      <ns:Base>
        <ns:EventId>421A2EE66BA3A42A8C1203350641340</ns:EventId>
        <ns:Timestamp>2008-10-17T09:59:51.328-08:00</ns:Timestamp>
        <ns:EventType>MyEventDefinitionEvent</ns:EventType>
        <ns:ObjectType>MyEventDefinitionObject</ns:ObjectType>
        <ns:ObjectId>123</ns:ObjectId>
        <ns:Source>
          <ns:Product>MyEventDefinition</ns:Product>
          <ns:ProductVersion>1.0</ns:ProductVersion>
          <ns:ProductInstance>MyEventDefinitionInstance</ns:ProductInstance>
        </ns:Source>
        <ns:User>
          <ns:ALFSecurity>
            <ns:UsernameToken>
              <ns:Username>sbmUser</ns:Username>
              <ns:Password>sbmUserPassword</ns:Password>
            </ns:UsernameToken>
          </ns:ALFSecurity>
        </ns:User>
      </ns:Base>
      <ns:Extension>
        <mym:MyEventDefinitionData>Example Tool Data</mym:MyEventDefinitionData>
      </ns:Extension>
    </ns:ALFEventNoticeDoc>
  </soapenv:Body>
</soapenv:Envelope>
```

## Header Values

The most important different between an e-mail event and an event sent through HTTP is that a Web service addressing header must be used:

```
<soapenv:Header>
  <wsa:To>mailto yourEmailUser@yourEmailHost?X-Service-Path=/eventmanager/
→services/ALFEventManagerOneWayDocLit</wsa:To>
  <wsa:MessageID>urn:uuid:421A2EE66BA3A42A8C1203350641340</wsa:MessageID>
  <wsa:Action>urn:EventNotice</wsa:Action>
</soapenv:Header>
```

The value of element `wsa:To` must be in the following format:

**e-mailAddress**?X-Service-Path=/eventmanager/services/**eventmanagerService**

where **e-mailAddress** is the dedicated POP3 e-mail account set up to receive events, and **eventmanagerService** is the Event Manager service that you want to receive the event.

The Event Manager service should be either `ALFEventManagerOneWayDocLit` or `ALFEventManagerDocLit`. `ALFEventManagerOneWayDocLit` is generally preferred, because it does not send an e-mail response message.

### Event Instance Values

The value of `wsa:MessageID` should typically be a newly allocated UUID in the format shown. It may be useful to set the value of the message EventId to the same UUID value, because that will enable you to correlate the e-mail instance with the event.

You do not have to provide a value for `ObjectId`, although it can be useful to do so. Generally, it should contain a unique identifier for the object instance that is responsible for generating the event.

You must provide a value for `Timestamp` or set it to nil. The expected `Timestamp` value uses the subset of ISO 8601 used by XML Schema as specified for the XML Schema type `datetime`. To set `Timestamp` to nil, you must reference the following namespace:

```
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
```

and set the nil attribute instead of defining a value:

```
<ns:Timestamp xsi:nil="true"/>
```

### Event Match Values

As with any SBM external event, the values of `EventType`, `ObjectType`, `Product`, `ProductVersion`, and `ProductInstance` must be set correctly to match a deployed event mapping. E-mail events are essentially external events, so it is generally necessary to create a custom event definition WSDL and import it into a process app before you can usefully process an e-mail event.

### Authentication

As with any SBM external event, the event will not be accepted unless valid credentials are provided in the event request. Currently, the only practical way to do this is to provide a completed `ALFSecurity` element. You must fill in the values for `Username` and `Password`. At this time, only plain-text passwords are accepted. Because the credentials are passed in plain-text, it is recommended that the client send the message over HTTPS.

### Extension Data

If your event definition specifies extension data, you can provide the appropriate XML structure and values inside the `Extension` element in the message.

### Event Definitions and Message Style

Event definitions are used to define service flows that use the *RPC literal* message style, whereas the *document literal* message style is generally preferred for event communication. The example given above follows the *document literal* format and the document-literal event is contained within the root document element:

```
<ns:ALFEventNoticeDoc xmlns:ns="http://www.eclipse.org/alf/schema/EventBase/1">
  …Event Base and Extension elements go here…
</ns:ALFEventNoticeDoc>
```

Your message should validate against the WSDL/schema for the ALFEventManagerDocLit or ALFEventManagerOneWayDocLit services:

```
http://localhost:8085/eventmanager/services/ALFEventManagerDocLit?wsdl
```

```
http://localhost:8085/eventmanager/services/ALFEventManagerOneWayDocLit?wsdl
```

If you generate an XML document from an *RPC literal* service definition, the event content is contained within the parameter element `<EventNotice xmlns="">`, which in turn is contained within the operation element `<ns:EventNotice xmlns:ns="http://www.eclipse.org/alf/schema/EventBase/1">`:

```
<ns:EventNotice xmlns:ns="http://www.eclipse.org/alf/schema/EventBase/1">
  <EventNotice xmlns="">
    …Event Base and Extension elements go here…
  </EventNotice>
</ns:EventNotice>
```

To create the correct message for the e-mail event, you must replace the operation and parameter `EventNotice` elements with the document root element, `ALFEventNoticeDoc`.

## Upgrading Existing Event Definitions

The following topics describe how to upgrade your event definitions to work with the current version of Solutions Business Manager.

- Upgrading from SBM R3.X [page 235]
- Upgrading from SBM 2008 R2.X [page 235]

### Upgrading from SBM R3.X

The following points discuss upgrading your external event definition WSDL files from SBM 2008 R3.X to work with the current version.

- Replace the SBM 2008 R3.x Event Manager `.wsdl` and `.xsd` files with new ones from the Advanced Orchestration Package.
- Change the type of `<your product value>` EventBaseType's EventId to `SourceEventIdType`.

### Upgrading from SBM 2008 R2.X

Remember the following points when upgrading your existing event definition WSDL files from SBM R2.X to the current version.

- Replace the SBM 2008 R2.x Event Manager `.wsdl` and `.xsd` files with new ones from the Advanced Orchestration Package.
- Change the type of `<your product value>` EventBaseType's EventId to `SourceEventIdType`.
- You might need to update your clients to set ALFSecurityType in order to fulfill the authentication requirements if you choose to use Single Sign-On (SSO).

Starting in SBM 2008 R3, there is an authentication mechanism for external events.

- **Note:** The following information only applies to systems where external events were used with orchestration workflows in which SSO was not used:

With the use of security tokens for all communication with SBM components regardless of authentication method, it is now necessary to provide credentials in the `User` element of external events that are processed by the Event Manager. Credentials must be supplied in order to receive a security token.
Previous SBM releases allowed anonymous events if SSO was disabled. Security tokens are now used in all underlying communication. As part of the upgrade process, in order to still accept external events without credentials, the Event Manager is automatically configured to continue to accept external events without authentication credentials. If SSO was enabled prior to upgrade, then it is assumed that external events always included credentials and will continue to do so in your environment.

**Important:** If you are currently using external events without SSO, it is strongly recommended that you adjust the source of those external events to now include credentials. Once you adjust the external source to include a credential, you can then manually override the Event Manager settings by setting the `no_authentication` parameter to "false" in the `alf.properties` file. For configuration instructions, visit the Knowledgebase and search for `no_authentication`.

After upgrading, the `no_authentication` setting is independent of the SSO setting. If you are performing a new installation, you can override the default behavior for the Event Manager and enable it to accept external events without credentials. For configuration instructions, visit the Knowledgebase and search for `no_authentication`.

For SBM Application Engine Web services, the SBM Application Engine auth still overrides the security token auth. In some cases, this is useful in day-to-day operations and may be useful as you upgrade. For example, orchestration workflows that contain coded auth for the SBM Application Engine service calls will continue to work if the external event is changed to send a credential; the coded auth will override the security token and continue work as it did prior to upgrade.

# Chapter 10: Calling RESTful Web Services from an Orchestration Workflow

RESTCaller is a SOAP wrapper utility service provided by SBM that enables SBM orchestration workflows to call REST-based Web services. This chapter describes the utility and its applicable operations, arguments, and responses.

This chapter contains the following sections:

## Introduction

You can use the orchestration workflow service step to access and configure the RESTCaller service to call REST Web services.

XML data from the orchestration workflow can be mapped into the RESTCaller and sent to the REST Web service as text, XML, or as JSON data. Similarly, XML data returned by the REST Web service can be mapped into the orchestration workflow. If the REST Web service returns JSON data, it can be translated to XML and mapped into the orchestration workflow as XML.

**Note:** If you are modifying process apps created in a prior version of SBM, you must do one of the following to refresh the process app with the latest RESTCaller functionality:

- If you are using a process app that was created prior to 10.1.5 (or if you delete the service from your process app):
  - From the context menu, right-click the **Web Services** node, select **Add**, and then select **RESTCaller**.
  - From the context menu, right-click the **Web Services** node, select **Add New Web Service**, and then manually import the `RESTCaller.wsdl` into your orchestration. The `RESTCaller.wsdl` can be found here:

```
http://ServerName:Port/orchestrationutilities/services/RESTCaller?wsdl
```

- If you have an existing process app created in SBM version 10.1.5 or higher, you can get the latest features by viewing the property pane of the RESTCaller service in your Orchestration and clicking the WSDL Refresh button. Features added since 10.1.5 are upward compatible and refreshing the `.wsdl` is only necessary if you need the new features.

RESTCaller is a system service that uses Security Token authentication by default. If you import RESTCaller, you must ensure that it uses Security Token authentication by performing the following steps:

1. Log in to SBM Application Repository.
2. Open the **Environments** view and select your environment.
3. Edit the RESTCaller service endpoint and set the authentication to **Security Token**.
4. Redeploy the process app to the environment.

## RESTCaller Operations

This section describes the available RESTCaller operations.

### PUT

Sends an HTTP PUT request to the rest service URL. Generally used for **creating** the addressed resource.

**Arguments:**

- restUrl
- options
- params

- bodyXML

**Options that apply:**

- returnAllHttpCodes
- copyResultsAsText
- responseConversion
- sendAsJSON
- sendAsText
- sendParamsAsFormData
- sendThisContentType
- authorizationType
- httpAuthorization
- returnJSONTypeHints
- httpHeaders
- preserveJSONEscapes

**Note:** HTTP PUT commands are often blocked by outbound firewall policies; therefore, ensure you have access through the firewall before the orchestration workflow executes a PUT command.

## GET

Sends an HTTP GET request to the rest service URL. Generally used for **retrieving** the addressed resource, but some services may use GET for other purposes as well.

**Arguments:**

- restUrl
- options
- params

**Options that apply:**

- returnAllHttpCodes
- copyResultsAsText
- responseConversion
- authorizationType
- httpAuthorization
- returnJSONTypeHints
- httpHeaders

- preserveJSONEscapes

### POST

Sends an HTTP POST request to the rest service URL. Generally used for **updating** the addressed resource.

**Arguments:**

- restUrl
- options
- params
- bodyXML

**Options that apply:**

- returnAllHttpCodes
- copyResultsAsText
- responseConversion
- sendAsJSON
- sendAsText
- sendParamsAsFormData
- sendThisContentType
- authorizationType
- httpAuthorization
- returnJSONTypeHints
- httpHeaders
- preserveJSONEscapes

### DELETE

Sends an HTTP DELETE request to the rest service URL. Generally used for **deleting** the addressed resource.

**Arguments:**

- restUrl
- options
- params

**Options that apply:**

- returnAllHttpCodes

- authorizationType
- httpAuthorization
- httpHeaders

**Note:** HTTP DELETE, PUT, and PATCH commands are often blocked by outbound firewall policies; therefore, ensure you have access through the firewall before the orchestration worklow executes these commands.

Another workaround is to use an alternate HTTP header such as X-HTTP-Method-Override: PATCH, X-HTTP-Override, or X-Method-Override. Whether these headers work or not depends on the framework that implements the REST service.

### HEAD

Returns the headers that result from GET but does not return the resource.

For more information, visit https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html.

### OPTIONS

Returns information about methods the resource supports, typically with an Allow header that lists the operations. Request and response body data is optional.

For more information, visit https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html.

### PATCH

Allows partial updates of resources.

For more information, visit https://tools.ietf.org/html/rfc5789.

### TRACE

Returns the sent message as it is received.

For more information, visit https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html.

## Request Arguments

This section describes the available request arguments. The following arguments can be used by REST Web service operations.

### restUrl

This is the URL of the REST service or resource in the form:

```
http://server:port/service/resource...
```

If you are using a custom endpoint, you can select the custom endpoint and map `restUrl` to the `Url` parameter in the custom endpoint.

**Tip:** You can use the custom endpoint for `restUrl` and the RESTCaller `params` input to add any required HTTP query string parameters to manipulate or retrieve the resource you want. For an example use case, refer to Using Custom Endpoints with RESTCaller [page 166].

### options

The options control how the RESTCaller service creates the HTTP request that it sends to the REST service and how it translates the response from the REST service. Not all options apply to all RESTCaller operations as noted for the respective operation. Where the option does not apply, no action is taken.

- **returnJSONTypeHints**

  At times it is useful for the received XML element names to have type hint prefixes. The `returnJSONTypeHints` option makes it easier to copy received data to a send data structure since the elements defined in the sending and receiving XML structures can be declared using the same names. Set the **returnJSONTypeHints** option to `true` if this is desired.

- **returnAllHttpCodes**

  By default, for any HTTP return code from the HTTP service or resource that is not in the `2XX` success range (such as `200`, `201`), the RESTCaller service throws a RESTCallerFault. Setting the `returnAllHttpCodes` option to `true` suppresses this behavior; this means the RESTCaller will return whatever HTTP return code the HTTP service or resource used to respond. This allows the orchestration workflow to explicitly process these responses. Note that RESTCaller can throw a RESTCallerFault for reasons other than an HTTP code that is out of the `2XX` range.

- **copyResultsAsText**

  The RESTCaller will translate the response body to XML depending on the contentType of the HTTP service or resource response. To reduce the volume of extraneous data, if the data is translated to XML, the actual text of the response body not returned. You can set `copyResultsAsText` to `true` to instruct the RESTCaller service to return the body text, even if an XML translation was performed.

- **responseConversion**

  By default, the RESTCaller parses the response body, and translates it to XML depending on the contentType of the HTTP service or resource response. ContentTypes that are processed include:

  - `application/xml`
  - `text/xml`
  - `application/xhtml+xml`
  - `application/json`

  Occasionally, the response content or its contentType may not correspond with one of these recognized contentType values. This option instructs the RESTCaller how to attempt to translate the body response. Note that if the response is not actually in the form that is specified, RESTCaller may throw a RESTCallerFault. The following values can be specified:

  - `CONTENTTYPE` – This is the default. The response content type is used.

    - `XML` – An XML response is assumed, and an attempt is made to parse the data to XML.
    - `JSON` – A JSON response is assumed, and an attempt is made to parse the data to JSON.
    - `NONE` – No format is assumed, and the response is returned as text.

- **sendAsJSON**

  By default, RESTCaller sends the operation bodyXML data argument as XML with the contentType value set to `application/xml`. If the service or resource expects JSON data, set this option to `true` to instruct the RESTCaller service to translate the bodyXML data to JSON and send the translated JSON data with the content type `application/json`.

  **Note:** When you use the sendAsJSON option, you must declare a "JsonDoc" complexType element (with no namespace) under the bodyXML. For example, you can create a working data variable, "bodyXML", of complexType, add JsonDoc as a complexType child element with no namespace, and then structure your request under JsonDoc. You then map the working data "bodyXML" to the RESTCaller bodyXML parameter. For more details, refer to Constructing Working Data XML to Map to JSON [page 250].

- **sendAsText**

  Sends the operation bodyXML data arguments as text with the contentType value set to `text/plain`. By default, the content type is set to `text/plain` but you can override this by setting a specific `contentType` with the `sendThisContentType` option.

  **Note:** When you use the sendAsText option, you must declare a "TextDoc" string element (with no namespace) under the bodyXML. For example, you can create a working data variable, "bodyXML", of complex type, add TextDoc as a string child element with no namespace, and then assign the text string you want to send to TextDoc. You then map the working data "bodyXML" to the RESTCaller bodyXML parameter.

- **sendParamsAsFormData**

  Enables you to construct URL query parameters but have them sent as the body on a POST. URL parameters can be sent as part of the URL or via the **params** argument. Normally these are added together and sent as the URL query string, but with this option, any parameters on the URL remain part of the URL and any parameters passed using the **params** array argument are converted to a query string text that is sent as the body.

  With this option, the content type is set to `application/x-www-form-urlencoded` but you can override this by setting a specific contentType with the `sendThisContentType` option. The bodyXML argument is ignored if `sendParamsAsFormData` is specified.

- **sendThisContentType**

  By default, RESTCaller sends the operation bodyXML data argument as XML with the contentType value set to `application/xml`. If the sendAsJSON option is `true`, the contentType value `application/json` is sent. If the service or resource requires a particular contentType value other than these defaults, use this option to instruct the RESTCaller to send the specified value for the contentType instead.

The working data variable that is mapped to bodyXML must be constructed according to certain rules in order to use this option. For more information, refer to Constructing Working Data XML to Map to JSON [page 250].

- **authorizationType**

  By default, RESTCaller does not attempt any authorization with the service or resource that is called. Several methods of authorization are supported, and this option must be set to enable the desired method. In addition, the appropriate fields of the corresponding `httpAuthorization` option must be filled in as described.

  The possible values include:

  - `BASIC` – HTTP Basic authorization is attempted. You must provide the `httpAuthorization` argument.
  - `NONE` – The default. No authorization is attempted.
  - `NTLM` – Authorization using Windows credentials. You must provide the `httpAuthorization` argument, and then specify the username, password, and domain.
  - `SBMTOKEN` – Forwards the implicit SBM SAML token to the service or resource. This option primarily enables RESTCaller to invoke REST services that are provided by SBM.
  - `ENDPOINT` – Indicates that the authentication information is provided by a custom endpoint. The custom endpoint fields must be mapped to the `httpAuthorization` argument.

- **httpAuthorization**

  Enter the required information as follows:

  - If the `authorizationType` argument is set to BASIC, the `basic` element of `httpAuthorization` must have values for `username` and `password`. These values are used to complete the HTTP Basic authorization header, which takes the form:

```
Authorization: Basic <base64encoded representation of the string username:password>
```

  - If the `authorizationType` argument is set to NTLM, the `ntlm` element of `httpAuthorization` must have values for `username`, `password`, and `domain`.
  - If the `authorizationType` argument is set to ENDPOINT, the `endpoint` element of `httpAuthorization` must identify the desired custom endpoint. Map the `EndpointID` value from the corresponding custom endpoint.

- **httpHeaders**

  Enables you to send arbitrary HTTP headers with the REST service request in case the service requires some particular HTTP headers. Note that the HTTP header, "Content-Type" is set implicitly by RESTCaller or via the `sendThisContentType` option and should not be set via the `httpHeaders` option.

- **preserveJSONEscapes**

  This option preserves JSON escape characters in return values in their JSON-escaped format. By default, JSON escape characters returned by a JSON REST service are converted to the actual character; however, because some characters are invalid in XML, when RESTCaller attempts to convert the JSON result to XML, the call fails if the character is converted to an invalid character in XML. To avoid this issue, use preserveJSONEscapes to preserve the characters in their escaped format.

### params

This argument is a list of key-value pairs for query parameters to append to the service or resource URL. The keys and their respective values are appended to the URL after the question mark (`?`), and are separated by an ampersand (`&`). For example, given a key (`key1`) with a value (`value1`), and another key (`key2`) with a value (`value2`), the resulting URL is sent with the request:

```
http://server:port/service/resource?key1=value1&key2=value2
```

**Tip:** You can send empty keys or values if the value should be null or uninitialized. For example:

```
http://server:port/service/resource?=value1
```

Or:

```
http://server:port/service/resource?key1=
```

However, RESTCaller does not support null key names (values with no preceding `key=`) like:

```
http://server:port/service/resource?value1
```

### bodyXML

This argument sets the data for the body of the PUT and POST commands.

- If you are sending JSON data, you must instruct RESTCaller to translate the bodyXML to JSON and you must include JsonDoc under the bodyXML. For details, see sendAsJSON.

- If you are sending text data, you must instruct RESTCaller to translate the bodyXML to text and you must include TextDoc under the bodyXML. For details, see sendAsText.

## Request Responses

This section describes the request responses. The following responses are returned by REST Web service operations.

### code

This is the returned HTTP response code. If the HTTP services or resource request is successful, the HTTP response code that is returned is in the `2XX` range. By default, if this code is outside the success range, RESTCaller throws a RESTCallerFault. If the `returnAllHttpCodes` option is set to `true`, all HTTP response codes are returned.

### message

This is the returned HTTP response message. By default, if the HTTP response code is outside the success range, RESTCaller throws a RESTCallerFault. If the `returnAllHttpCodes` option is set to `true`, all HTTP response codes are returned along with the corresponding HTTP response message.

### resultString

Returns the HTTP body text, unless RESTCaller can return the response body formatted as XML. If the response body can be formatted as XML, this field is empty unless the `copyResultAsText` option is set to `true`.

### resultXML

Returns the HTTP body text if RESTCaller can return the response body formatted as XML.

### resultHeaders

Returns the HTTP headers from the REST service response in the form of an array of structures containing `name` and `value`.

## Sending and Receiving HTTP Body Data

The following sections describe how to construct working data to map XML and JSON body data to and from the RESTCaller service and the REST service that it is calling.

### Sending XML Data

**To send XML data:**

1. Create a working data variable with some convenient name, such as "BodyXMLVariableOut".

2. Set the type to Private Complex.

3. Add a child DataElement to represent the document root element of the HTTP body XML. It should be named and typed the same as the root XML element of the data that appears in the body of the REST HTTP call. Take care to set the correct XML namespace as expected by the REST services, because the child DataElement will take the namespace defined of the Orchestration inherited from the parent working data variable (in this example, BodyXMLVariableOut) by default. In most cases, the document root element should be a Private Complex type. Naming the child DataElement, "RESTServiceDocument", creates the working data structure:

   ```
   BodyXMLVariableOut
                     RESTServiceDocument
   ```

   which results in the document root element sent in the HTTP body as:

```
<RESTServiceDocument>…</RESTServiceDocument>
```

4. Add any further Child DataElements under the document root element. These should be added with the appropriate names, types, and namespaces to create the same structure as the expected XML. For example, naming the Child DataElement, "Name":

   ```
   BodyXMLVariableOut
                  RESTServiceDocument
                                 Name
   ```

   Results in the root XML element sent in the HTTP body as:

   ```
   <RESTServiceDocument>
          <Name>…</Name>…
   </RESTServiceDocument>
   ```

5. Initialize the created working data variable structure DataElements with appropriate default values or via Calculate steps in the normal way.

   **Note:** If a DataElement is optional and no value is mapped to it, it will not be sent in the resulting XML structure.

6. Create a RESTCaller service step, and set the operation to PUT or POST, and set the Option values as appropriate. Map the created working data variable to the "bodyXML" parameter. For example, the variable "BodyXMLVariableOut" should be mapped to the "bodyXML" parameter. At runtime, the content "RESTServiceDocument" is copied to set the content of the "bodyXML" parameter and sent as the HTTP body content of the resulting HTTP REST service call:

   ```
   <RESTServiceDocument>
          <Name>my name</Name>
          <OtherElement>some value</OtherElement >
   </RESTServiceDocument>
   ```

## Receiving XML Data

**To receive XML data:**

1. Create a working data variable with a convenient name such as "BodyXMLVariableIn".

2. Set the type to Private Complex.

3. Add a Child DataElement to represent the root element of the HTTP body XML. It should be named and typed the same as the root XML element of the data that will appear in the body of the REST HTTP call response. Take care to set the correct XML namespace as returned by the REST services because the Child DataElement will take the namespace defined of the Orchestration inherited from the parent working data variable (in this example, BodyXMLVariableIn) by default. In most cases, the root element should be a Private Complex type. For example, naming the Child DataElement "RESTServiceDocument" creates the following working data structure:

```
BodyXMLVariableIn
                RESTServiceDocument
```

which creates a container for the expected document root XML element as:

```
<RESTServiceDocument>…
</RESTServiceDocument>
```

4. Add any further Child DataElements under the HTTP body root element. These should be added with appropriate names, types, and namespaces to create the same structure as the expected XML. For example, naming the Child DataElement "Name" results in the expected root XML element to be received in the HTTP body as:

   ```
   <RESTServiceDocument>
          <Name>…</Name>…
   </RESTServiceDocument>
   ```

   **Important:** In addition to the name, it is essential to set the namespace property correctly for any element in this structure. If the declared namespace does not match the namespace of the element in the received XML, the orchestration interprets them as different elements and does not copy the value data.

   **Tip:** Due to the way Orchestrations work, it is only necessary to declare the specific elements from the received XML that you actually need to process in the Orchestration. The working data variable must be created with sufficient structure to place the element at the correct path in the XML data, but elements that are not part of a path and not accessed do not need to be declared.

5. Create a RESTCaller service step, and set the operation to GET, PUT or POST as appropriate. Set the Option values as appropriate. Map any inputs as required and described above.

6. Create a Calculate step that follows the RESTCaller service step, and map the RESTCaller service step response results.result.XML to the variable that you created to receive the returned data. For example, "BodyXMLVariableIn".

7. Create further Calculate steps to extract specific values from the variable that you created to receive the returned data. For example, use the "BodyXMLVariableIn" variable to retrieve the value of the returned RESTServiceDocument and its children. The value of the Name child element is retrieved using the calculate Expression `BodyXMLVariableIn.RESTServiceDocument.Name`.

**Note:** If the REST service provides an XSD schema for the XML document it expects, it may be possible to import that schema and use the named types defined within it to set the type of the root XML elements for either sent or received XML. For details on this advanced usage, search the knowledge base at https://www.microfocus.com/support-and-services/#SBM for solution S140078.

## Sending JSON Data

**To send JSON data:**

1. Create a working data variable with a convenient name such as "BodyJSONVariableOut".

2. Set the type to Private Complex.

3. Add a Child DataElement to hold the XML that will be converted to JSON data. It must be named "JsonDoc". Set the type to Private Complex. Set the namespace to empty by deleting any value in the Namespace property.

4. Add any further Child DataElements under the JsonDoc root element. These should be added with the appropriate names and types to create the same structure as the expected JSON. Set the namespace to empty by deleting any value in the Namespace property. The specific rules for correctly structuring and naming the elements is explained in Constructing Working Data XML to Map to JSON [page 250].

5. Initialize the created working data variable structure DataElements with appropriate default values and/or via Calculate steps in the normal way.

   **Note:** If a DataElement is optional and no value is mapped to it, it will not be sent in the resulting XML structure or the JSON structure that it converts to.

6. Create a RESTCaller service step, and set the operation to PUT or POST as appropriate. Set the sendAsJSON Option parameter to true. Set other Option values as appropriate. Map the created working data variable to the "bodyXML" parameter. For example, the variable "BodyJSONVariableOut" should be mapped to the "bodyXML" parameter. At runtime, the content "JsonDoc" will be copied to set the content of the "bodyXML" parameter and then converted to JSON as the HTTP body content of the resulting HTTP REST service call.

## Receiving JSON Data

**To receive JSON data:**

1. Create a working data variable with a convenient name such as "BodyJSONVariableIn".

2. Set the type to Private Complex.

3. Add a Child DataElement to hold the XML that will be converted to JSON data. It must be named "JsonDoc". Set the type to Private Complex. Set the namespace to empty by deleting any value in the Namespace property.

4. Add any further Child DataElements under the JsonDoc root element. These should be added with the appropriate names and types to create the same structure as the expected JSON. Set the namespace to empty by deleting any value in the Namespace property. The specific rules for correctly structuring and naming the elements is explained below in Constructing Working Data XML to Map to JSON [page 250].

   **Important:** In addition to the name, it is essential to set the namespace property to empty by deleting any value in the Namespace property for any element in this structure because the XML that is converted from JSON uses no namespace. If the declared namespaces for all the elements are not empty, they will not match the empty namespace of the elements in the received JSON XML. The orchestration interprets them as different elements and does not copy the value data.

   **Tip:** Due to the way Orchestrations work, it is only necessary to declare the specific elements from the received JSON XML that you actually need to process in the Orchestration. The working data variable must be created with sufficient structure to place the element at the correct path in the XML data, but elements that are not part of a path and not accessed do not need to be declared.

5. Create a RESTCaller service step, and set the operation to GET, PUT or POST as appropriate. Set the Option values as appropriate. Map any inputs as required and described above.

6. Create a Calculate step that follows the RESTCaller service step, and then map the RESTCaller service step response results.result.XML to the variable that you created to receive the returned data. For example, "BodyJSONVariableIn".

7. Create further Calculate steps to extract specific values from the variable that you created to receive the returned data. For example, use "BodyJSONVariableIn" to retrieve the value of the returned Name child element.

## Constructing Working Data XML to Map to JSON

JSON has Objects {} that contain an unordered list of key:value pairs and Arrays [ ] that contain an ordered list of values. Values are implicitly typed as either string, number, boolean, or null. The values in a JSON array need not be of the same type. Key names have no special restricted characters; this means they have the same allowed character set as string values. Some special characters in strings must be escaped with an escape character sequence using the backslash character \.

There are several considerations when constructing working data variables to map to or from JSON data:

- General Rules [page 251]
- Type [page 251] – Distinguishing between strings, numbers, and booleans
- Structure – Objects [page 252]
- Structure – Arrays [page 254]
- Sending Empty Arrays, Objects, and Strings [page 254]

- Mixed Arrays and returnJSONTypeHints [page 255]
- Character Escaping [page 256] – Handling characters that are restricted or not allowed in XML but are allowed in JSON
- XML Start Characters [page 257]

### General Rules

Note the following general rules:

- For requests sent as JSON, the root element contained by the working data variable must be named JsonDoc. The JsonDoc element must be a complexType and have no namespace.
- Elements declared under JsonDoc must have no namespace.
- For requests sent as text, the root element contained by the working data must be named TextDoc. The TextDoc element must be a string type and have no namespace.
- Elements declared under TextDoc must have no namespace.

### Type

XML value types are defined separately from the XML data itself, typically in an XSD schema. RESTCaller does not have access to the .XSD schema for the bodyXML, and instead relies on a default mapping scheme with element name type hint prefixes to distinguish non-default types. If the type hints are not used, by default an Element with child Elements is interpreted as a JSON Key with an Object value, and an Element with no child Elements is interpreted as a JSON key with a string value. For example:

```
<myObject>
       <myString>string value</myString>
       <myNumber>123456</myNumber>
       <myBooelan>true</ myBooelan >
<myObject>
```

Will be:

```
"myObject": {
                "myString": "string value",
                "myNumber": "123456",
                "myBoolean": "true"
        }
```

Note that the values for myNumber and myBoolean are quoted and therefore typed as strings in the JSON structure which is incorrect. The type hint prefixes override this default behavior allowing JSON Array, Number and Boolean values to be set correctly. The type hint prefixes are:

- ARRAY _va
- OBJECT _vo
- NUMBER _vn

- STRING _vs
- BOOLEAN _vb

To use a type hint prefix, prepend the appropriate type hint prefix to the name of the element. For example, if an element named myNumber is typed as some form of number (integer, long, float, double) such as 123456, then prefixing its name with _vn will create an element as follows:

```
<_vomyObject>
       <_vsmyString>string value</_vsmyString>
       <_vnmyNumber>123456</_vnmyNumber>
       <_vnmyBoolean>123456</_vnmyBoolean>
<_vomyObject>
```

When this element is sent as JSON, the RESTCaller service will remove the type hint prefix when converting the element name to a JSON key name. Using this example, the JSON key:value will be:

```
"myObject": {
               "myString": "string value",
               "myNumber": 123456,
               "myBoolean": true
       }
```

**Important:** Because the type hint prefix is removed before the name is converted to a JSON key name, the type hint prefix should not be considered as part of the name for the purposes of uniqueness. Note that the Working Data editor in SBM Composer is unaware of the RESTCaller type hint prefixes and cannot enforce uniqueness if the type hint hides the duplicate name.

For receiving JSON data, by default the JSON key names will become the XML element names and values will convert to the element text value in the case of strings, numbers, and booleans. For object and array values, a containing element named for the key will be created. Object properties will be sub-elements in any order. Array item values will be held in multiple repeating <Item> elements in the specified array order. Object and Array conversion is discussed in more detail in the structures below.

**Tip:** Use the `returnJSONTypeHints` option to receive data with elements prefixed with the JSON type hints _vo, _va, _vs, _vn, vb as appropriate.

## Structure – Objects

Elements with PrivateComplexType map to JSON objects by default. For example, this XML structure:

```
<JsonDoc>
       < myObject>…</myObject>
</JsonDoc>
```

maps to the JSON document:

```
{
"myObject": { …
```

```
        }
}
```

Elements that are children of the PrivateComplexType Element map to the key:values of a JSON object by default. This XML structure:

```
<JsonDoc>
        < myObject>
                <aMember>a member value</aMember>
                <bMember>b member value</bMember>
                <cMember>c member value</cMember>
        </myObject>
</JsonDoc>
```

maps to the JSON document:

```
{
"myObject": {
                "cMember": "c member value",
                "bMember": "b member value",
                "aMember": "a member value"
                }
}
```

**Note:** The object members may appear in a different order within the Object value because JSON object members are unordered. Each member should have a unique key name within the object.

The type hint name prefix, _vo, may be optionally be used. Typing hint name prefixes are removed to create the JSON name. Since the type hint gets removed from the element name when converting to JSON, it should not be considered as part of the key name for the purposes of uniqueness. This XML structure:

```
<JsonDoc>
        < _vomyObject>
                <aMember>a member value</aMember>
                <bMember>b member value</bMember>
                <cMember>c member value</cMember>
        </_vomyObject>
</JsonDoc>
```

maps to the JSON document:

```
{
"myObject": {
                "aMember": "a member value",
                "bMember": "b member value",
                "cMember": "c member value"
        }
}
```

### Structure – Arrays

Elements that are children of the Private ComplexType can be used to declare arrays but special a type hint name prefix, _va, must be used to distinguish it from an object and ensure the correct JSON structure is created.

If the JSON array contains values of the same type, a repeating element can be used to define the array values. The special name—Item—should be used to name this element in order to be consistent with data that is converted from JSON. The child Item element can have a type hint prefix to indicate its type. For example, refer to the following XML structure:

```
<JsonDoc>
       <_vamyArray>
               <_vnItem>123</_vnItem>
               <_vnItem>456</_vnItem>
       </_vamyArray >
</JsonDoc>

Maps to the JSON document
{ "myArray": [
               123,
               456
               ]
}
```

**Important:** Due to the way the XML to JSON conversion works, it detects repeating elements in the XML and attempts to convert them to JSON arrays. However, this works inconsistently and it is strongly recommended to use the pattern described here with an array type hint prefix on a containing element that names the array key with a contained repeating element to hold the array values.

**Important:** JSON arrays can contain mixed value types. That is, a JSON array can contain string, number, array, object or Boolean mixed together. For more information, see Mixed Arrays and returnJSONTypeHints [page 255].

### Sending Empty Arrays, Objects, and Strings

To send empty arrays, objects, and strings you must use the appropriate prefix (listed below), otherwise the default is a string.

JsonDoc defaults to a string value `""`, and if it is empty then `""` is sent.

- `_voJsonDoc` is an object `{}`, and `{}` is sent if the element value is set to `""` with a calculate step.
- `_vaJsonDoc` is an array `[]`, and `[]` is sent if the element value is set to `""` with a calculate step. By providing the appropriate child variables, you can also use this to send a JSON document that is an array.
- `_vsJsonDoc` is a string `""`, and if it is empty `""` is sent. To send a string, the value must be set.
- `_vnJsonDoc` is a number, and if it is empty null is sent. To send a number, the value must be set.

- `_vbJsonDoc` is a Boolean true or false, and if it is empty null is sent. To send a true or false, the value must be set.

For example, if you want a JsonDoc that is an empty array `[]`, declare the child element of complex type (the actual type does not matter here) and no namespace:

```
MyRequest
....
_vaJsonDoc
```

Then create a calculate step that assigns an empty string to `_vaJsonDoc`. This applies to `object _voJsonDoc {}`, `array _vaJsonDoc []`, `string _vsJsonDoc ""`, `number _vnJsonDoc 0`, and `boolean _vbJsonDoc true|false`.

Similarly, if you want an internal JSON property to have a value that is an empty array `[]`, `"MyProperty" : []`, declare the child element of complex type (the actual type does not matter here) and no namespace:

```
MyRequest
....
_vaMyProperty
```

Then create a calculate step that assigns an empty string to `_vaMyProperty`.

The same rules apply to object named values and array Item values.

Value defaults to an string value `""`, and if it is empty then `""` is sent.

- `_voValue` is an object `{}`, and if it is empty `{}` is sent
- `_vaValue` is an array `[]`, and if it is empty `[]` is sent
- `_vsValue` is a string `""`, and if it is empty `""` is sent
- `_vnValue` is a number, and if it is empty null is sent
- `_vbValue` is a Boolean true or false, and if it is empty null is sent

Item defaults to an string value `""`, and if it is empty then `""` is sent.

- `_voItem` is an object `{}`, and if it is empty `{}` is sent
- `_vaItem` is an array `[]`, and if it is empty `[]` is sent
- `_vsItem` is a string `""`, and if it is empty `""` is sent
- `_vnItem` is a number, and if it is empty null is sent
- `_vbItem` is a Boolean true or false, and if it is empty null is sent

## Mixed Arrays and returnJSONTypeHints

For arrays, the container element must have the `_va` hint and it is not repeating. The child element name is a placeholder and typically repeats if the array is a single type array (for example, all strings) and it should have the appropriate type hint for its type (for example, `_vs`).

Mixed type arrays are possible in JSON and the same applies to them, except there will be multiple children. Mixed arrays that are returned use the repeating element "Item". If you use `returnJSONTypeHints`, the item will be prefixed with the type (`_vsItem`, `voItem`, etc). You can access each set of these using a "for each" step. The XPath selects by name, so the order does not matter. However, the elements will be re-ordered from the original because they are separated by type.

Note that with `returnJSONTypeHints`, the returned root document element is always `JsonDoc` even it if is an array.

Also an empty object `{}` and empty array `[]` are returned as an empty `JsonDoc`. Currently a `JsonDoc` that is a non-complex type is not supported as a JSON `responseConversion` return value. Note that there is a workaround if you expect a plain string or number, which is to use `responseConversion = NONE` and look for the response in the `resultString` response element.

## Character Escaping

XML element names use a restricted set of UNICODE characters with some special rules. JSON key names can use any UNICODE character with the exception that a small set of control characters must be escaped rather than used directly. Because of these differences, JSON key names cannot necessarily be directly used as XML element names and it is not possible to model any JSON key name using an XML element name.

To overcome this, RESTCaller provides a character escaping mechanism for XML element names that allows XML element names to model almost any JSON key name. Character escaping works by substituting the desired character that cannot be used with a special "escape" character followed by a code character that indicates the desired character. RESTCaller uses the underscore, '_' , as the special "escape" character and defines the following codes:

| Name | XML Escape | JSON Character |
|---|---|---|
| Start | _ | The char that follows the _ |
| SPACE | _w | ' ' |
| UNDERSCORE | __ | '_' |
| HEXCHAR | _xhhhh | The Unicode char with the hex value hhhh |
| BACKSPACE | _b | \b |
| FORDFEED | _f | \f |
| LINEFEED | _n | \n |
| RETURN | _r | \r |
| TAB | _t | \t |
| QUOTE | _q | \q |

| Name | XML Escape | JSON Character |
|---|---|---|
| FORWARDSLASH | _s | \/ |
| BACKSLASH | _c | \\ |
| UTF16HEX | _uhhhh | \uhhhh |

Use the preserveJSONEscapes option to preserve JSON escape characters in return values in their JSON-escaped format.

## XML Start Characters

XML allows less characters to be used for the first character of the element name than it does for following characters. If the JSON key name starts with one of these illegal start characters, it can be modeled in XML with the _ escape and will be escaped by the REST caller when the JSON is mapped to XML. For example, XML element names cannot start with a number (0 through 9). If the desired JSON has a key name "10x10", this can be declared in XML as `<_10x10>`. This "start" escape can only be used on the first character of the XML name. Only invalid start characters that are otherwise valid name characters can be escaped this way and only if they are the first character of the element name.

**Note:** Type hint prefixes do not count as part of the element name. If type hint prefixes are used, the start character is the character immediately following the type hint prefix. For example, the start character of the element name `_vs1000` is 1, which is an invalid start character and must be escaped as `_vs_1000`.

**Note:** Namespace name qualifiers are ignored for this purpose, which is why XML structures that map to JSON data must be declared with no namespace.

**UNDERSCORE**

Because RESTCaller uses the _ character as an escape, it must be escaped so that the JSON key can contain an _ character. _ was chosen because it is a valid XML start character.

**HEXCHAR**

Illegal XML characters can be encoded with the HEXCHAR escape sequence, where the illegal character is expressed as a sequence of 4 hex digits that correspond to the UNICODE character hexadecimal code point value. For example, the char +, is `_h002B`. The purpose of this is to allow the character to pass un-escaped in the JSON data. It should only be used for characters that cannot be directly represented in an XML name, but can appear directly in a JSON string.

**BACKSPACE**, **FORMFEED**, **LINEFEED**, **RETURN**, **TAB**, **QUOTE**, **FORWARDSLASH**, **BACKSLASH**, **UTF16HEX**

These escapes correspond to the JSON escapes for various special control characters. Control characters (`U+0000` through `U+001F`) without a special escape must be represented using with `\uhhhh` where `hhhh` is the UNICODE character hexadecimal code point value. For example, the char Vertical Tab, is `_u000A`.

**Note:** JSON allows any character to be represented this way but it is unclear if such a representation is un-escaped by the consumer or by the JSON parser, so the behavior of this escape may vary.

**Note:** Unicode surrogate pairs are not currently supported.