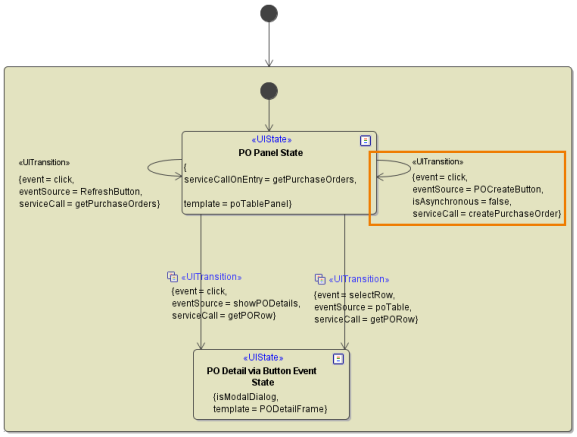
Service Calls

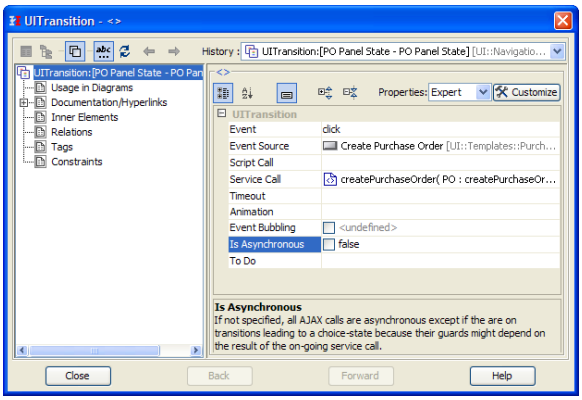
Synchronous Service Calls

AJAX calls are generally asynchronous as their nature dictates. It can though happen, that a process needs to wait for a AJAX call to finish because the call's response holds information which is needed to proceed to the process step, e.g. decisions, further calls.



In the above figure the <<UITransition>> defining the PO creation process is set to synchronous (isAsynchronous = false). The reason for that is, that the creation process could take some time to finish and will exit to the **PO Panel State** and refresh the table too early. In the above example the process now waits until the PO is created only then it will exit and refresh the PO table listing.

To set an AJAX call synchronous, set the **isAsynchronous** flag within the <<UITransition>> or <<UIGuardedTransition>> to false. The default of **isAsynchronous** is by default set to false, because the variables which the decision actually uses is not yet set.



On this Page:

Related Pages:

- Authentication and Authorization
- File Upload
- HTTPS
- History State
- Form and Form Validation
- Calling a UI from external Applications
- Usage of Choices
- Service Calls
- HTTP Proxy
- Controller States
- Back Button and Browser History
- Mock-Ups

Call Services By Script

Example Files (Builder project Advanced Modeling/UI):

	<your example path>\Advanced Modeling\UI\uml\uiCallServiceByScript.xml <your example path>\Advanced Modeling\UI\uml\uiCallServiceInTableContextByScript.xml

Typically, a JavaScript operation can be invoked *after* a service is being called. However, sometimes JavaScript code must be executed *before* calling a service in order to evaluate conditions, set mock-ups, do some calculations, etc.
For this purpose the following tagged values can be used:

UI Element	Tagged Value	Description	Values	
State	**serviceCallOnEntryByScript**	If true, the service is not invoked automatically, but must be called by the JavaScript operation specified in **serviceCallOnEntry**. In this case, the transition must have a JavaScript handler having the signature: `<handler name>(service)`.	true	services must be invoked manually in the JavaScript operation
			false	services are invoked automatically
State	**serviceCallOnExitByScript**	If true, the service is not invoked automatically, but must be called by the JavaScript operation specified in **serviceCallOnExit**. In this case, the transition must have a JavaScript handler having the signature: `<handler name>(service)`.	true	services must be invoked manually in the JavaScript operation
			false	services are invoked automatically

Transition	**serviceCallByScript**	If true, the service is not invoked automatically, but must be called by the JavaScript operation specified in **serviceCall**. In this case, the transition must have a JavaScript handler having the signature: `<handler name>(event, service[, row, data])`. The optional parameters are relevant only if the event is triggered within a table. Details see below.	true	services must be invoked manually in the JavaScript operation
			false	services are invoked automatically

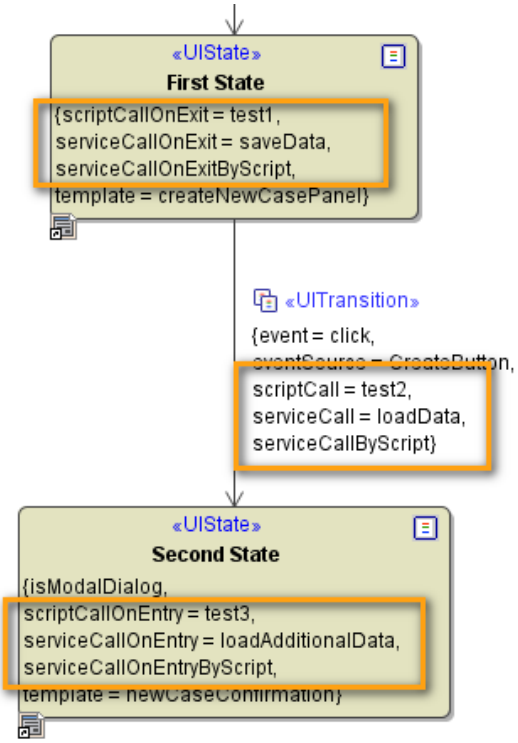
The service parameter is a function to be invoked in the script. The invocation depends on the **isAsynchronous** flag of the transition that enters this state:

- **isAsynchronous = true**: `service.<service name>();`
- **isAsynchronous = false**: `var result = service.<service name>();`
  result is false, if an error occurred.

For transitions, If the event is triggered within a table row, the row is supplied as a parameter:

- **isAsynchronous = true**: `service.<service name>(row);`
- **isAsynchronous = false**: `var result = service.<service name>(row);`
  result is false, if an error occurred.

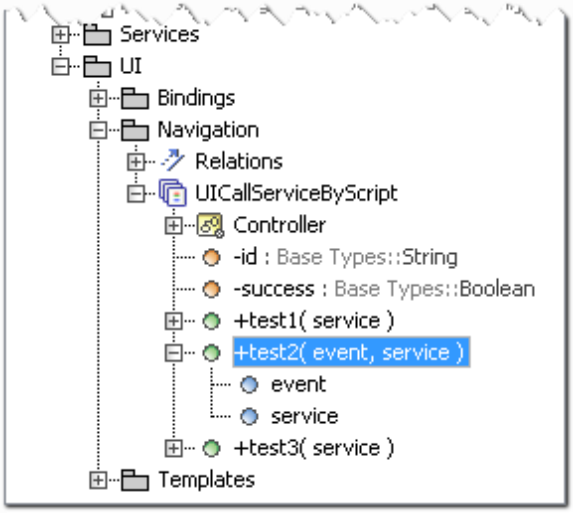
Examples of these tagged values are found in the figure below:



The signature of the called JavaScript operation depends on whether the operation is being called in a state or on a transition:

- **In states:** the JavaScript operation gets only one parameter: the **service** object.
- **On transitions**: The JavaScript operation takes two parameters: **event** and **service**. The event object is the W3C event triggering the transitions. The service object is the target object to call the service(s).

Note, that the following order of these parameters is mandatory: 1. event, 2. service

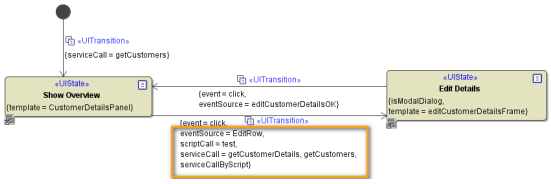


Additionally, the signature also depends on whether the event occurs within a table. This case is discussed below.

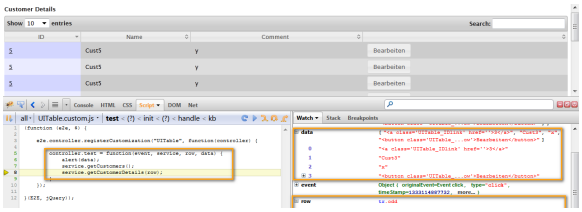
The **service** object enables the modeler to call the services being declared in one of the tagged values **serviceCallOnExit**, **serviceCallOnEntry**, or **serviceCall**. For example, the **test2** script used on the transition shown in the figure above might look like:

```
alert("Even target id: "+ event.currentTarget.id);
// do somenthing before calling the service, e.g:
// setting variables, mockups, calculating conditions, etc.
service.loadData();
```

If the event triggering the transition occurs in a table, the JavaScript operation gets two additional parameter: **row** and **data**. The row parameter points to the actual selected HTML row, the data object contains the data as JavaScript object. The former is more convenient for HTML manipulation and can also be given to the service call. The latter is more convenient if the data are to be processed within the JavaScript operation. The following figure shows an example of a transition triggered by a table event:



If the example is run, the input to the JavaScript **test** operation can be analyzed using Firebug. For example, the following figure shows the UI after the transition has been triggered by pressing the **Bearbeiten** button:



Callback when an Asynchronous Service Returns

The following tagged values can be assigned to JavaScript operations. These operations are called when the associated service calls return:

- **serviceCallOnExitResponseHandler**

- **serviceCallResponseHandler**
- **serviceCallOnEntryResponseHandler**

Examples of these tagged values are found in the figure below. Additionally, a simple dummy callback implementation is depicted.