Orchestration Server Developer's Guide

SCXML Language Reference

12/13/2025

Contents

SCXML Language Reference

Click here to view the organization and contents of the SCXML Language Reference.

SCXML stands for State Chart XML: State Machine Notation for Control Abstraction. Orchestration Server utilizes an internally developed SCXML engine which is based on, and supports the specifications outlined in the W3C Working Draft 7 May 2009 [1]. There are, however, certain changes and/or notable differences (see Extensions and Deviations) between the W3C specification and the implementation in Orchestration Server which are described on this page. Only the ECMAScript profile is supported (the minimal and XPath profiles are not supported).

Usage

SCXML documents are a means of defining control behaviour through the design of state machines. The SCXML engine supports a variety of control flow elements as well as methods to manipulate and send/receive data, thus enabling users to create complex mechanisms.

For example, Orchestration, which is a consumer of the SCXML engine, uses SCXML documents to execute strategies. A simple strategy may involve defining different call routing behaviours depending on the incoming caller, however, the SCXML engine facilitates a wide variety of applications beyond simple call routing.

Authoring SCXML documents can be done using any text editor or by leveraging the Genesys Composer tool.

Syntax and Semantics

The appearance of an SCXML document is very similar to that of any other markup language. The use of various SCXML-specific tags define the structure and operation of a state machine. The SCXML snippet below illustrates how an SCXML document may look like in structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Specifying the encoding is important! -->

<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
  name="SCXML DOCUMENT NAME" >
  <initial>
    <transition target="FIRST_STATE">
      <log expr="'Inside the initial transition'" />
    </transition>
  </initial>

  <state id="FIRST_STATE">
    <onentry>
      <log expr="'Do things here when state is first entered'" />
      <send event="LEAVE_STATE" />
    </onentry>
    <onexit>
```

```
      <log expr="'Do things here when state is being exited'" />
    </onexit>
    <transition event="LEAVE_STATE" target="exit">
      <script>
        var message = 'Do things here during a transition';
      </script>
      <log expr="message" />
    </transition>
  </state>

  <final id="exit" />
</scxml>
```

Basic Elements

SCXML elements for defining states, transitions, and behavior include (but are not limited to):

- <state>
- <transition>
- <parallel>
- <initial>
- <final>
- <onentry>
- <onexit>

In addition, it is also possible to define actions which are performed during state transitions (<onentry>, <onexit>, within <transition>) by using the following Executable Content elements (but are not limited to):

- <if>/<elseif>/<else>
- <foreach> ***(planned feature)***
- <raise>
- <log>

It is also possible to embed script code (ECMAScript) within the state machine by using the following element:

- <script>

One may refer to the W3C Working Draft [2] for more detailed explanations of the above elements and basic usage examples.

Managing Data

SCXML applications are commonly driven by data acquisition and manipulation, where the data gathered can be processed to determine application behaviour. Users may define data models within an SCXML document by using the <datamodel> element. A <datamodel> element may have any number of <data> elements as seen below:

```
<datamodel>
    <data ID="target" expr="'Hello World!'" />
    <data ID="this_is_one" expr="1" />
    <data ID="m_array" expr="['a', 'b', 'c']" />
</datamodel>
```

Any data objects defined in this manner become accessible as a child of the `_data` global object. To access a data object defined within a <datamodel>, the following syntax is used:

```
_data.data_ID    // Where data_ID is replaced by the ID of the data element
```

Alternatively, one may opt to declare data objects/variables within a <script> block instead if more complex initialization routines are required. Variables defined within a <script> block, however, become children of the <script> element's parent's local scope. That is, if it was defined in the global scope (<scxml>), the variables will be globally accessible; if it was defined within a state, the variables will become children of the state's local scope.

```
<script>
    var target='Hello World!';
    var this_is_one=1;
    var m_array = ['a', 'b', 'c'];
</script>
<log expr="'The value of target is: ' + target" />
```

Data sharing between SCXML sessions

It may be desirable in many situations to be able to share data between multiple SCXML sessions. Data may be shared between sessions using the following methods:

Session Initiated

> When one SCXML session initiates another SCXML session via the <invoke> action (or <session:start>, <session:fetch>, which are specific to **Orchestration only!**) the initiating session can share data via the model defined in the SCXML specification. For details, see the <invoke> implementation section on the W3C website.

Session runtime

> During the execution of a session, a session can shared data with another session via events and the <send> action.

Events

Event handling (both internal and external) is fully supported by the SCXML engine. The event model allows users to control SCXML sessions by sending events from external entities or by *raising* events internally during the execution of an SCXML document. These events can be used to drive transitions or *send* data to external systems.

Internal Events

These events are published and consumed by the same SCXML session. The following are the methods of managing them:

Publish

To generate an event, either the <event> or <send> element can be used. The SCXML engine puts the event into the session's event queue. When using <send>, it is possible to place the event on either the external event queue or internal event queue, based on the value of the target attribute. (If the special value '`_internal`' is specified, the event is added to the internal event queue of the current session. If no value is specified, the event is added to the external event queue of the current session.). When using <send> to generate events, if there is an intention to cancel the event sent, it is recommended to use the attribute `idlocation` instead of `id`.

Subscribe

Receiving events is achieved by using the <transition> element. If the event contains properties, one may access the event's properties via the `_event` system variable:

```
_event.data
```

The Orchestration platform supports the use of wildcards ("*") when evaluating event names.

External Events

These events are published and consumed by the given SCXML session and the corresponding external entity. The following is a list of external entities that are supported:

- Other SCXML sessions
- External systems via Functional Modules
- External applications

The following are the methods of managing events from an SCXML-session standpoint:

Publish

The <send> element with the appropriate `targettype` attribute value:

- scxml - for other SCXML sessions. Events may be delivered to the appropriate session within the same platform server or across platforms, and is facilitated by the message functionality of the platform. The `target` attribute has the following format: `url#sessionid`
- basichttp - for external applications. These events are delivered to the appropriate external application, based on the defined target URL and an HTTP POST message.
- fm - for any Functional Module-related systems. The `target` attribute is the functional module's namespace name.

In addition to the <send> element, a given Functional Module may have an action element to send events, as well.

Subscribe

The <transition> element. If the event contains properties, one may access the event's properties via the `_event` system variable:

```
_event.data
```

In general, overall external event subscription is implicit:

- Functional Modules
- External applications and other SCXML sessions - When these events are sent, they are sent explicitly to the given session (by session ID), so no explicit subscription is needed.

The following are the methods of managing events from an external system standpoint:

Publish

The method depends on the source of the event:

- Other SCXML sessions - The <send> element is used.
- External applications - The platform external interface is used (SendTransitionEvent). The platform has the appropriate functionality to receive events from external sources and deliver them to the appropriate sessions.
- Functional Modules - The Functional Module sends the events to the platform based on the defined Functional Module framework interfaces and the platform then delivers the events to the appropriate session event queue.

Subscribe

For any of the potential subscribers, there is no explicit subscription method, because the SCXML session is targeting a specific destination when publishing the event, so the destination must have the appropriate interface to receive the event.

- Functional Modules - The Functional Module supports the appropriate functional module framework interface to receive the events from the session.
- External applications have the appropriate web application to process the HTTP post.
- Other SCXML sessions receive the event on their event queues via the platform.

Common Properties for Internal and External Events

The following common properties are present in all events, whether internal or external:

- `name` - This is a character string giving the name of the event. It is what is matched against the 'event' attribute of <transition>. Note that transitions can carry out additional tests by using the value of this field inside boolean expressions in the 'cond' attribute.
- `type` - This field describes the event type. It MUST contain one of an enumerated set of string values consisting of: "platform" (for events raised by the platform itself, such as error events), "internal" (for events raised by <event>), and "external" (for all other events, including those that the state machine sends to itself via <send>).

- `sendid` - In the case of error events triggered by a failed attempt to send an event, this field contains the sendid or id of the triggering <send> element. Otherwise it is blank.
- `invokeid` - If this event is generated from an invoked child process, this field contains the invokeid of the invocation (<invoke invokeid="..." or id="...">) that triggered the child process or in the case of error events triggered by a failed attempt to invoke another process, this field contains the invokeid or id of the invoking <invoke> element. Otherwise it is blank.

The following fields are logically present in all events, but are filled in only in external events:

- `origin` - This a URL, equivalent to the 'target' attribute on the <send> element. The combination of this field with the 'origintype' field SHOULD allow the receiver of the event to <send> a response back to the entity that originated this event. Not currently supported.
- `origintype` - This is a character string, similar to the 'targettype' or 'type" attribute in <send>. The combination of this field with the 'origin' field SHOULD allow the receiver of the event to <send> a response back to the entity that originated this event. Not currently supported.
- `data` - This field contains whatever data the sending entity chose to include in this event. The receiving platform SHOULD reformat this data to match its data model, but MUST not otherwise modify it.

Extensions and Deviations

The Genesys SCXML implementation introduces various additions and differences from the W3C SCXML specifications. The following extensions and deviations were introduced to accommodate the needs of the SCXML engine.

ECMAScript

SpiderMonkey 1.8.5 is used as the ECMAScript engine. It implements a superset of ECMA-262 Edition 5. This allows for the inclusion of scripts within SCXML documents when more advanced computational logic is required beyond the standard SCXML elements.

System Variables

The SCXML specification defines the following system variables which may provide useful information to applications (all of which are available in the *global* scope):

`_sessionid`

> This represents the unique ID associated with this SCXML session. It is set by the platform.

`_name`

> This represents the name that the developer gives this particular SCXML document (for example, "Mortgage Process Logic". It is set by the developer when creating the document.

`_event`

This represents the event being presented to the application. It is set by the platform when the event is available to the application.

`_type`

This represents the type of application that the developer gives this particular SCXML document (that is, <SCXML> element `_type` attribute). It is set by the developer when creating the document.

In addition to the above variables, the SCXML engine also provides the following extension system variables:

`_parentSessionid`

This represents the unique ID associated with the parent of this SCXML session. If the session has no parent, the value returned is an empty string. It is set by the platform.

`_genesys`

This is the root object for accessing all Genesys-specific ECMAScript objects and functions. Note that the user is not allowed to set properties in _genesys as it is a protected system object. See Orchestration Extensions for more information.

`_data` (<datamodel>)

These are the objects that are created based on the datamodels defined within the SCXML document. For example, data to be used during the processing of the logic and expected initiation and return parameters for the session. See the _data (ORS Extensions) section below for Orchestration-specific properties of the datamodel.

Protected Variables

Variables may be defined and set on any scope except within _genesys. In addition, top-level variables starting with '_' are considered system variables and should not be modified. Defining top-level variables with names starting with an underscore '_' is prohibited. However, the same restriction does not apply if the variable is defined under a top level property such as the datamodel.

_data (ORS extensions)

In addition to storing session start parameters and user-defined datamodel items, the _data object may sometimes be used by Orchestration to provide extra information about the session.

Property Name	Description
_data.provision_object_name (8.1.200.48)	Name of Enhanced Routing Script object associated with session. Provided in sessions that are started

Property Name	Description
	from a Script object (of type Enhanced Routing).

Variable Scoping

An SCXML session has one Global Scope, and many Local Scopes. Note that this implementation differs from the W3C specification (as of February 16, 2012 [3]) as the specification dictates that all variables must be placed into a single global ECMAScript scope. Nevertheless, it is still in compliance with the W3C Working Draft 7 May 2009 [4].

The SCXML engine creates a scope for each state in the document. The parent scope for each local scope is the parent state's local scope (or for <scxml>, the global scope). Each local scope shares it's name with the state name. This allows the SCXML logic to access ECMAScript objects and variables in its active ancestor's local scopes. For example, to access object x in the local scope of the grandparent of the current local scope state, one may use the following syntax:

```
__grand-parent-name__.x    // Returns value of x from grandparent's local scope
```

See the following example on variable scoping:

```
<?xml version="1.0" encoding="UTF-8"?>
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml" initial="outer">
  <datamodel>
    <data ID="dataval" expr="'Accessible from anywhere!'" />
  </datamodel>

  <script>
    var globalval = 'Also accessible from anywhere!';
  </script>

  <state id="outer">
    <onentry>
      <script>
        var parentval = 'Accessible from outer';
      </script>
      <log expr="_data.dataval" />
      <log expr="globalval" />
    </onentry>

    <initial>
      <transition target="inner" />
    </initial>

    <state id="inner">
      <onentry>
        <script>
          var childval = 'Accessible from inner';
        </script>
        <log expr="childval" />
        <log expr="__outer__.parentval" />
      </onentry>
      <transition target="done_inner" />
    </state>

    <final id="done_inner" />

    <transition event="done.state.outer" target="exit" />
  </state>
```

```
  <final id="exit" />
</scxml>
```

Functions and <script> Elements

ORS functions defined within SCXML documents (e.g., via <script> elements) may not behave as expected following an Orchestration failover and session restoration. Specifically, the scope in which a function executes may change following a session restore. Consider the following example:

```
<scxml initial=”my_state”>
...
<script>
  // This is a top-level script block
  var scope = “global”;
</script>

<state id=”my_state”>
  <onentry>
    <script>
      var scope = “my_state”;
      var hello = “hello world!”;
      var fun = function(){
        __Log(scope + “: “ + hello);
      }
    <script>
  </onentry>
  <transition event=”show_message”>
    <script>
      fun();
    </script>
  </transition>
</state>
...
```

The purpose of the function `fun` is simple: it will print the message "my_state: hello world!". Given the example SCXML above, every time the event `show_message` is processed, the function fun will be called.

Now assume that an ORS failover has occurred. On session recovery, the user may find that he/she can no longer call the function `fun` without receiving an error like:

```
14:57:28.247 METRIC <exec_error sid='T5SL1E5D9923J7OG8TM4LCLQMG000001'
result='ReferenceError:
hello is not defined. Line 1 - in <script> at line: 117' thread='8596' />
```

An uneval of the session datamodel post-recovery might indicate the variables declared within the state have all been persisted:

```
__my_state__:{
       scope:”my_state”,
       hello:”hello world!”,
       fun:(function () {__Log(scope + “: “ + hello);})
       }
```

The key difference between pre-recovery and post-recovery is that now, the function `fun` will execute in global scope instead of local scope. This means that any variables referred to within the function `fun` will only be resolved in the global scope. A slight modification can illustrate the difference:

```
<scxml initial="my_state">
...
<script>
  // This is a top-level script block
  var scope = "global";
</script>

<state id="my_state">
  <onentry>
    <script>
      var scope = "my_state";
      var fun = function(){
        __Log(scope);
      }
    <script>
  </onentry>
  <transition event="show_message">
    <script>
      fun();
    </script>
  </transition>
</state>
...
```

With the above code, `fun()` will now instead print: "global". The scope will be resolved to the variable declared in the top-level script block.

One possible solution would be to design the function as follows:

```
<scxml initial="my_state">
...
<script>
  // This is a top-level script block
  var scope = "global";
</script>

<state id="my_state">
  <onentry>
    <script>
    var scope = "my_state";
    var hello = "hello world!";
    var fun = function(self){
      __Log(self.scope + ": " + self.hello);
    }
    </script>
  </onentry>
  <transition event="show_message">
    <script>
      fun(this);
    </script>
  </transition>
</state>
...
```

By passing in a reference to `this`, the scope of the variables are now strictly-defined and should survive persistence with no change in behaviour.

Object Ownership

Objects and its associated properties are not implicitly shared with other sessions or external applications, but there are methods to explicitly share these objects and properties with other

sessions and applications. A session can only share snapshots of the current properties and objects - they are not updated dynamically when the owning session changes them. The following are the methods of how content can be shared:

- When the current session is starting another session, the current session can send these properties and objects to the new session via the <invoke> or <session:start> with the <param> elements.
- One can use the <send> action element or the Web 2.0 API equivalent of the <send> element. This allows any session or external application to get any property or object on any session.

Functions

The session has access to system functions (time, date, and so on) through the standard ECMAScript objects and functions. In addition to the core ECMAScript script functions, the SCXML engine exposes some other useful functions.

E4X (ECMAScript for XML)

SpiderMonkey supports E4X, which adds native XML support to ECMAScript. This allows the user to access XML data as primitives, rather than as objects.

See the following example for usage:

```
var sales = <sales vendor="John">
    <item type="peas" price="4" quantity="6"/>
    <item type="carrot" price="3" quantity="10"/>
    <item type="chips" price="5" quantity="3"/>
  </sales>;

alert( sales.item.(@type == "carrot").@quantity );
alert( sales.@vendor );
for each( var price in sales..@price ) {
  alert( price );
}
delete sales.item[0];
sales.item += <item type="oranges" price="4"/>;
sales.item.(@type == "oranges").@quantity = 4;
```

JSON

The following functions provide a convenient method of serializing and deserializing data to and from the JavaScript Object Notation (JSON) format:

- JSON Function Set 1 - These functions are fast and should not be used on JSON-related data that is untrusted:

  `uneval( object )`
  : Converts object to JSON string form.

  `eval( string )`
  : Converts JSON string to object form.

- JSON Function Set 2 - These functions are more secure (for example, will not run script logic) and are defined in the ECMAScript 5th Edition standard. This function set is based on the open source version

found at http://www.json.org/js. Note also that it is currently unable to handle cycles:

`JSON.stringify( object, replacer function )`
Converts object to string form.

`JSON.parse( string, replacer function )`
Converts JSON string to object form

__GetDocumentURL

Returns the URL of the currently running scxml strategy.

Usage:

```
__GetDocumentURL()
```

Parameters:

- None

Returns:

- `url`: STRING - e.g. "www.example.com/scxml/strategy.scxml"

__GetDocumentBaseURL

Returns the base URL of the currently running scxml strategy.

Usage:

```
__GetDocumentBaseURL()
```

Parameters:

- None

Returns:

- `base_url`: STRING - e.g. www.example.com/scxml/

__Log

This function is the ECMAScript equivalent to the <log> element. It allows an application to generate a logging or debug message which a developer can use to help in application development or post-execution analysis of application performance.

Usage:

```
__Log (expr) or __Log(expr, label, level)
```

Parameters:

- `expr`: STRING which can be a variable or a constant - This parameter returns the value to be logged.
- `label`: STRING which can be a variable or a constant - This parameter may be used to indicate the purpose of the log.
- `level`: STRING which can be a variable or a constant - This parameter specifies the log level.

Returns:

- None

__Raise

This function is the ECMAScript equivalent to the <raise> element. It allows an application to raise an event which can be used to direct the execution flow of an SCXML strategy.

Usage:

1. __Raise(expr)
2. __Raise(expr, data)
3. __Raise(expr, delay)
4. __Raise(expr, data, delay)

Parameters:

- `expr`: STRING which can be a variable or a constant - This parameter specifies the event name.
- `data`: OBJECT which can be a variable or a valid expression - This parameter specifies a data model. The data from which is included in the event (like <param> children of a <raise> element).
- `delay`: STRING which can be a variable or a constant - This parameter must evaluate to a valid CSS2 time designation. It specifies the time delay prior to firing the event.

Returns:

- None

__GetDocumentType

Returns the document type of the currently running scxml strategy. The value returned is equivalent to the value of the _type attribute of the <SCXML> element, as specified by the developer.

Usage:

```
__GetDocumentType()
```

Parameters:

- None

Returns:

- `type`: STRING

__GetCurrentStates

Returns an array of strings containing names of the currently active states.

Usage:

```
__GetCurrentStates()
```

Parameters:

- None

Returns:

- `states`: ARRAY[STRING1, STRING2, ..]

__GetQueuedEvents

Returns an array of strings containing events currently placed into the event queue.

Usage:

```
__GetQueuedEvents()
```

Parameters:

- None

Returns:

- `events`: ARRAY[STRING1, STRING2, ..]

In

Determines if the state specified is currently active. If so, returns true, otherwise returns false.

Usage:

```
In( "example_state_name" );
```

Parameters:

- `state`: STRING which can be a variable or a constant - This parameter specifies the name of the state to be compared against.

Returns:

- `is_in_state`: BOOLEAN

Function Scoping

A caveat resulting from the Scoping Variable Scoping deviation is that developers **must** now be

aware of the scope in which his/her functions execute. User-defined functions will generally execute in the *local scope* where they were defined. This means that any variables referenced within a user-defined function **must** also exist within that very same scope.

It is possible to invoke a function outside of its native scope (e.g. by referencing another state directly through the object model, or by referencing a function that was defined in a parent state), but note that its variable scoping will remain in that native scope (where the function was defined). See example below:

```
<state id="outer" initial="inner">
  <onentry>
    <script>
      var scope="outer";
      var foo=function(){__Log("foo finds scope: " + scope);}
    </script>
  </onentry>
  <state id="inner">
    <onentry>
      <script>
        var scope="inner";
        var bar=function(){__Log("bar finds scope: " + scope);}
      </script>
      <script>
        foo(); // Outputs --> foo finds scope: outer
        bar(); // Outputs --> bar finds scope: inner
      </script>
    </onentry>
  </state>
</state>
```

Note for Orchestration users only:

Function scope will not be restored after session recovery! This is a critical difference that **must** be accounted for when designing an SCXML application to survive fail-over and recovery. Tests have shown that after a session has been restored from persistence, all user-defined functions (particularly those defined within states will execute in the *global* scope as opposed to their original native scope.

To address this issue, it is highly recommended that developers explicitly specify the scope of their variables rather than use implicit scoping. See example below:

```
<state id="outer" initial="inner">
  <onentry>
    <script>
      var scope="outer";
      var implicit=function(){__Log("implicit finds scope: " + scope);}
      var explicit=function(self){__Log("explicit finds scope: " + self.scope);}
    </script>
  </onentry>
  <state id="inner">
    <onentry>
      <script>
        var scope="inner";
      </script>
      <script>
        implicit();          // Outputs --> implicit finds scope: outer
        explicit(this);      // Outputs --> explicit finds scope: inner
        explicit(__outer__); // Outputs --> explicit finds scope: outer
      </script>
    </onentry>
  </state>
```

```
</state>
```

SCXML Elements

<anchor>

The <anchor> module is *not supported* by the SCXML engine. This element would otherwise be used for providing 'go back' or 'redo'-like functionality for applications.

<cancel >

For <cancel>, either one of the attributes `id` and `sendid` may be used. However, both cannot be defined at the same time.

When using <send> to generate events, if there is an intention to cancel the event sent, it is recommended to use the attribute `idlocation` instead of `id`. The sendid stored at the location specified by `idlocation` may then be used in <cancel>.

When the <cancel> request has been processed, the SCXML engine will send back the "cancel.successful" event if the event was successfully removed, or "error.notallowed" if there was a problem, along with the attribute `sendid` in the event.

<data>

The following are the additional Genesys attributes for <data> element. They are strictly used to help define and administer the provisioning of this data from the appropriate source.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
_type	false	NMTOKEN	data	The following is the set of valid values: • data • parameter	This allows the developer to identify the data elements that are to be parameters that the platform must obtain values for when the session is initiated. Note that this does not impact the way in which the SCXML document is executed.
_desc	false	string	none	Any valid string	This allows the

Name	Required	Type	Default Value	Valid Values	Description
					developer to provide a description of the parameter that is to be supplied at session initiation. Note that this does not impact the way in which the SCXML document is executed.

src Attribute

The currently supported URI schema types for the src attribute are:

- HTTPS
- HTTP
- FILE

id Attribute

The value of this attribute must be a valid ECMAScript variable name. This means that variable semantics that include elements like "." (for example, foo.foo) and "-" (for example, foo-foo) are not allowed. The rule is that the variable name must be able to be processed on its own in an ECMAScript snippet. If not, then a TypeError event is generated.

For example,

Valid element

```
<data id="foo" expr="'value1'"/>
```

Invalid element

```
<data id="foo.foo" expr="'value2'"/> <!--TypeError event generated ->
```

If you need to create complex objects you can always create them with the <script> element as a child of the <scxml> element with the src attribute where the src attribute value points to a valid JSON object with a mime type of application/json.

<foreach>

This element is an extension to the W3C Working Draft 7 May 2009. However, it has been formally added to the W3C SCXML specification since the W3C Working Draft 26 April 2011. <foreach> is an Executable Content element (like <if>, or <log>) and can be used to create iterators. The behaviour of <foreach> is similar to that of the C# and Perl 'foreach' construct, which traverses items in a

collection. This implementation differs from the W3C specification in that the SCXML engine behaves as though a deep copy of each item in the collection is created during iteration as opposed to a shallow copy. Nevertheless, iteration behaviour will remain unaffected by changes to the collection.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
array	true	Value expression	none	A value expression that evaluates to an iterable collection.	The <foreach> element will iterate over a deep copy of this collection.
item	true	string	none	Any variable name that is valid in the specified data model.	A variable that stores a different item of the collection in each iteration of the loop.
index	false	string	none	Any variable name that is valid in the specified data model.	A variable that stores the current iteration index upon each iteration of the foreach loop.

<history>

The <history> element is *not supported* by the SCXML engine. This element would otherwise be used for allowing 'pause and resume' control flows.

<invoke>

The `<invoke>` element is used to create an instance of an external service. This implementation differs from the W3C specification in that the SCXML engine does not support the `typeexpr`, `srcexpr`, and `namelist` attirbutes, and the `<content>` child element.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
type	false	URI	"scxml"	"scxml" (equivalent to "http://www.w3.org/TR/scxml/")	Type of the external service.
typeexpr	Not supported				

Name	Required	Type	Default Value	Valid Values	Description
src	false	URI	none	Any URI	A URI to be passed to the external service.
srcexpr	Not supported				
id	false	ID	none	Any valid token	A string literal to be used as the identifier for this instance of <invoke>.
idlocation	false	Location Expression	none	Any valid location expression	Any data model expression evaluating to a data model location.
namelist	Not supported				
autoforward	false	Boolean	false	true or false	A flag indicating whether to forward events to the invoked process.

For more details, see the <invoke> implementation section on the W3C website.

The currently supported URI schema types for the `src` attribute are:

- HTTPS
- HTTP
- FILE

The value of the `id` attribute must be a valid ECMAScript variable name.

Child Elements

- `<param>`: Element containing data to be passed to the external service. Occurs 0 or more times.
- `<finalize>`: Element containing executable content to massage the data returned from the invoked component. Occurs 0 or 1 times.
- `<content>`: Not supported.

<scxml>

The following are the additional Genesys attributes for the <scxml> element:

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
_type	false	string	combination	Any valid string	This is set by the developer at the beginning of the SCXML document to define what type of SCXML logic has been defined. Composer sets this property based on the type of logic you are building. It is used for reporting purposes.
_persist	false	boolean	false true (prior to 8.1.2)	The following is the set of valid values: • true • false	This allows the developer to suppress all persistence capabilities. Persistence is not always desired, due to the associated performance overhead. For instance, in Orchestration, current voice-related routing strategies normally run to completion in a reasonable amount of time, and in the event of a failure, restarting the routing strategy may not be problematic. Therefore, this attribute allows sessions to suppress all use of persistence, which prevents the orchestration platform from ever persisting the session. (Note that this does *not* preclude the orchestration platform from employing other techniques, such

Name	Required	Type	Default Value	Valid Values	Description
					as hot standby servers, to achieve fault tolerance for these types of session.
_statePersistDefault	false	string	"may"	The following is the set of valid values: • must • may • no	To ensure proper session persistence during High Availability recovery, the _statePersistDefault may be used as an attribute to the top-level <scxml> element. Orchestration Server uses the value of _statePersistDefault as the default for the <state> _persist attribute, if it is not specified at the <state> level. • may—Default value. ORS will persist the SCXML session in the entered state once the event queue becomes empty. • must—ORS will immediately persist the SCXML session in the entered state. • no—ORS will not persist the SCXML session in the entered state.

Name	Required	Type	Default Value	Valid Values	Description
_maxtime (Since ORS 8.1.300.03, SCXML 8.1.300.00)	false	integer	"604800"	Any valid positive integer, inside double quotes.	Specifies the maximum age in seconds that an ORS session should exist. If this age is reached, ORS shall attempt to exit the session. If specified, this overrides the value specified in configuration for ORS under scxml/max-session-age. To disable this feature, set the _maxtime to "0". As of ORS 8.1.300.13, SCXML 8.1.300.13, an available Cassandra data store will be required for this functionality.
_microStepLimit (Since ORS 8.1.300.11, SCXML 8.1.300.10)	false	integer	1000	Any valid positive integer, inside double quotes.	Specifies the maximum number of microsteps allowed to be taken following the processing of one event. Subsequent transitions may arise from the processing of one event if the following transitions are eventless. If this number is reached, ORS shall attempt to exit the session. To use ORS configured default, leave _microStepLimit undefined. To disable this feature, set _microStepLimit="0".

Name	Required	Type	Default Value	Valid Values	Description
_stateEntryLimit (Since ORS 8.1.300.11, SCXML 8.1.300.10)	false	integer	100	Any valid positive integer, inside double quotes.	Specifies the maximum number of times that a state may be entered as the target of a transition. States entered indirectly as the result of a transition element or initial attribute are not considered for this limit (e.g. ancestors of the target state that must be entered before entering the target state). If this number is reached, ORS shall attempt to exit the session. To use ORS configured default, leave _stateEntryLimit undefined. To disable this feature, set _stateEntryLimit="0".
_maxPendingEvents (Since ORS 8.1.300.11, SCXML 8.1.300.10)	false	integer	100	Positive integer between 30 to 100000, inclusive, inside double quotes.	Specifies the maximum number of events allowed to be queued to a session (inclusive of internal, external, delayed and undelivered events). If this number is reached, ORS shall attempt to exit the session. This feature cannot be disabled.

Name	Required	Type	Default Value	Valid Values	Description
_processEventTimeout (Since ORS 8.1.300.11, SCXML 8.1.300.10)	false	integer	10000	Any valid positive integer, inside double quotes.	Specifies the maximum time allotted for the processing of the event queue. The processing of one event may lead to additional events being queued. Processing of the event queue does not complete until the event queue is empty. This feature sets an upper bound to the amount of time dedicated to processing these events. If the timeout is reached, ORS shall attempt to exit the session. To use ORS configured default, leave _processEventTimeout undefined. To disable this feature, set _processEventTimeout="0".
_sendSessionRecovered (Since ORS 8.1.300.13, SCXML 8.1.300.13) _recoveryEnabled (Since ORS 8.1.300.12, SCXML 8.1.300.12)	false	boolean	false	The following is the set of valid values: • true • false	Specifies whether or not this strategy is eligible for proactive recovery. If set to true, the session will be explicitly restored by ORS when an ORS node performs switch-over to Primary. Proactive recovery shall never be used

Name	Required	Type	Default Value	Valid Values	Description
					for sessions what process multimedia interactions.
_debug (Since ORS 8.1.300.11, SCXML 8.1.300.10)	false	boolean	false	The following is the set of valid values: • true • false	Specifies whether or not debugging of SCXML strategy is required. When set to true, the session will save a copy of the fully assembled SCXML strategy to disk (working directory).
_transitionStyle (Since ORS 8.1.300.28, SCXML 8.1.300.38)	false	string	legacy	The following is the set of valid values: • legacy • genesys • w3c	Specifies the order in which the <transition> executable content is to be executed in the scenario where there are two or more selected transitions (only in <parallel> regions). • *legacy* setting dictates that transitions are executed by line order (lowest line number first) • *genesys* setting orders transitions by reverse scope

Name	Required	Type	Default Value	Valid Values	Description
					order. Transitions of deepest scope (most nested) are executed first. Ties for scope are broken by lowest line number first. • *w3c* setting adheres to the ordering prescribed by the W3C Working Draft for SCXML. Transitions are executed in the scope order of the states which selected them. Ties for scope are broken by lowest line number first.

<log>

<log> has three attributes (expr, label, level). For attribute details, please refer to State Chart XML (SCXML): State Machine Notation for Control Abstraction W3C Working Draft 7 May 2009 (www.w3.org). As of version 8.1.200.46, specifying a level of 5 with a label of 22000 to 22020 will result in behavior equivalent to that for URS/IRD.

<state>

The following are the additional Genesys attributes, children, and behavior for the <state> element:

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
_es_store (as of 8.1.400.40)	false	boolean	false	true, false	Introduced as part of the Elastic Connector feature described in the Orchestration Server 8.1.4 Deployment Guide. When set to true in the state description in a strategy, ORS will save information about session states into Elasticsearch, which can be used for operational/ performance monitoring and analytics. Example: <state id="routing" _es_store=”true”>.
_type	false	NMTOKEN	normal	The following is the set of valid values: • normal	This allows the developer to control how the platform is to handle this state and is a place holder for future support.
_persist	false	NMTOKEN	may	The following is the set of valid values: • no • may • must	Long-running sessions typically experience concentrated time windows in which active processing is performed, followed by a relatively long time window during which the system

Name	Required	Type	Default Value	Valid Values	Description
					awaits follow-up by a customer or potentially by the agent. This attribute is used to indicate to the platform whether a session can or must be persisted: • no - Used to indicate a state is transitional, or is not meaningful for recovery purposes over the last persisted state. • may - The platform may persist the session in this state at its discretion. This is the default value. • must - The platform must persist the session as part of entry processing of this state (before the <onentry> elements are executed). This is used to guarantee

Name	Required	Type	Default Value	Valid Values	Description
					that the session can be recovered from this point in the event of failure (that is, the ability to reenter the session at this state). **Note:** Orchestration Server uses the value of <scxml> _statePersistDefault as the default for the <state> _persist attribute, if it is not specified at the <state> level.
_deactivate	false	string	no	The following is the set of current valid values: • now • no	This attribute defines whether the session that enters this state and is waiting for a transition should be immediately persisted, removed from platform memory, and marked as inactive. This attribute is valid only if the "_persist" attribute is set to "may" or "must", since only persistable sessions can be de-activated. This attribute is treated purely as a hint to the platform about

Name	Required	Type	Default Value	Valid Values	Description
					how meaningful it is to persist the session.
src	*Not Supported*				

Persist/Deactivate option matrix		_persist		
no	***may***	**must**		
_deactivate	***no***	Persistence disabled. _deactivate attribute is ignored.	The session may be persisted in this state at the platform's discretion.	The platform guarantees that this state will be persisted upon entry.
	now	Persistence disabled. _deactivate attribute is ignored.	The session may be persisted in this state at the platform's discretion and will be marked as inactive when waiting for transition.	The session that enters this state and is waiting for transition will be immediately persisted and marked as inactive.

<param>

The <param> element has the following restriction:

- When using the <param> element with any action element, you must specify both the `name` and `expr` attributes. Because of this, the platform *does not support* the `name` attribute value as a data model location expression if the expr attribute is missing.

<transition>

The attribute `anchor` of the <transition> is *not supported*.

ORS Version 8.1.200.60/8.1.300.01

The behaviour of transitions in the SCXML engine is different from that which is described in the W3C Working Draft 7 May 2009 [5]. This draft spec explains the following:

The **LCA** is the innermost <state>, <parallel>, or <scxml> element that is a proper ancestor of the transition's source state and its target state(s).

During a transition, all active states that are proper descendants of the LCA are exited.

The new transition behaviour of the SCXML engine shares greater similarities with that of the W3C Working Draft 16 December 2010 [6], in that in the case of a transition whose source state is a compound state and whose target(s) is a descendant of the source, the transition will not exit and re-enter its source state. In addition, the notion of the LCA is replaced by the LCCA:

The **LCCA** is the innermost compound <state> or <scxml> element that is a proper ancestor of the transition's source state and its target state(s).

During a transition, all active states that are proper descendants of the LCCA are exited.

<validate>

The <validate> element is *not supported* by the SCXML engine. This would otherwise be used to invoke a validation of the datamodel.

<xi:include>

The xinclude recommendation (http://www.w3.org/TR/xinclude/) is used for inlining of ECMAScripts (<script>) and states (<state>). An application developer may specify scripts, states, and other content separately from the main SCXML document. The included document or fragment can be text or xml. See section below on using <xi:include>.

When using xinclude, the following are important considerations to keep in mind:

- Developers should ensure that the 'resolveid' attribute value is unique within a document. This is necessary when the same included document is used multiple times within the including document since, the IDs of <state>, <parallel>, and so on, must be unique across the entire document.
- Included documents must NOT transition back to states that are defined in an including document. This does not work.

Xinclude can also be used to provide a subroutine-like capability within an SCXML application by using it like a macro facility. This replaces all <xi:include> elements with the referenced state content during the initial document fetch and load. Once the SCXML application is fully assembled, it is compiled and validated before sessions can be created based on this application.

In addition to the considerations above, the following guidelines must be followed when using xinclude as a macro style "subroutine":

- For the included document:
  - The document must be a valid <state> fragment that specifies the complete behavior of the subroutine. The document can contain an <scxml> document, but if it does, the xinclude declaration must use xpointer to reference the <state> that is to be used as the subroutine.
  - The referenced <state> can be a simple or a compound state. If it is a compound state, it must define <initial> as well as <final> states.
  - An atomic state must use <raise>/<event> to return the appropriate output parameters. A compound state, on the other hand, can use either <raise>, <event>, or the <donedata> element of the <final> states to perform this function.
  - The included <state> must be self-contained: it should not have transitions to states in the including document or outside of itself.
  - The included <state> must not use datamodel elements from the included document, unless the <data> elements are defined within the <state> or one of its children. Using <data> elements defined elsewhere in the included document will likely result in an error since they are not defined in the including document.

    - The included <state> should not use datamodel elements from the including document. Doing so makes subroutine information global to the application. It is recommended that data should be passed to a subroutine via an event, or through variables defined via <script>.
    - The included <state> must not rely on events from the including document other than the transition to the included state.
- For the including document:
    - The document must have a <transition> for the event generated by the included state or subroutine. This event will contain the results from the subroutine.
    - The document must have a <transition> to the included state. The event can contain the input parameters for the subroutine. Alternately, the including state can use a <script> element in its <onentry> element to define and initialize a set of parameters that are passed to the included <state>. The included state can access these parameters through the variable scoping that ECMAScript provides.
    - When using <xi:include> elements, the namespaces used in the included document need to be declared in the including document.

The following are the additional Genesys attributes for the <xi:include> element as well as existing attribute limitations.

Attribute Details

Name	Required	Type	Default Value	Valid Values	Description
accept	false	string			*Not supported*
accept-language	false	string			*Not supported*
encoding	false	string			*Not supported*
href	true	URL	none		The URI of the resource to include. As of **ORS 8.1.300.27**, this attribute also supports substitution by session start parameters. These parameters may come from the ApplicationParms section of an Enhanced Routing Script, URL-encoded parameters of a web-started session, or the <param> elements nested within a <session:start> For example: `<xi:include`

Name	Required	Type	Default Value	Valid Values	Description
					href="http://appsrv:80/scxml/subroutine_routing.scxml" /> It is possible to parametrize the URI as follows: <xi:include href="http://appsrv:80/scxml/$$MY_SUBROUTINE$$" /> When a special token of the form $$*parameter_name*$$ is provided, it will be automatically substituted with the value of the matching session start parameter (case-sensitive). If the session start parameters are as follows: MY_SUBROUTINE = subroutine_chat.scxml Resulting URI: <xi:include href="http://appsrv:80/scxml/subroutine_chat.scxml" />
parse	false	string	"xml"	"xml", "text"	See the following for details: http://www.w3.org/TR/xinclude/#include_element
resolveid	false	string	none	Any value string	In order to support subroutines and avoid issues with duplicate SCXML element IDs (for example, <state id=x>), this Genesys

Name	Required	Type	Default Value	Valid Values	Description
					extension attribute must be used. If this attribute is specified, then ID modifications will occur. All the SCXML elements with an ID attribute (<state>, <parallel>, <final>, <history>, <send>, <invoke>, <cancel>, and <data>) in the included document are prefixed by the value of this attribute, and a separating dot. In addition, the IDREF attributes in the included document (the initial attribute in the <state> element and the target and event attributes in the <transition> element) can also be modified as long as the following wildcard substitution key is specified in the value. Otherwise they will not be changed when included. The substitution key is the string "$$_MY_PREFIX_$$". If specified, it

Name	Required	Type	Default Value	Valid Values	Description
					is replaced by the value of this attribute for the included document. **IMPORTANT NOTE:** Developers must ensure that each use of the 'resolveid' attribute value is unique across the chain of included documents.
xmlns	false	string	none	Any value string	Used to provide namespaces for the included document. This is necessary for fragments. If subroutines include subroutines, this attribute must be set to the appropriate namespace for the including element. For example, `xmlns:xi="http://www.w3.o 2001/ XInclude"`
xpointer	false	string	none		When `parse="xml"`, xpointer may be used to specify a particular element and its children to include. The value of xpointer must be a literal ID. The first node in the included document that matches that ID is included. When xpointer is omitted, the

Name	Required	Type	Default Value	Valid Values	Description
					entire resource is included. **Note:** XPath is not supported.

When resolveid is used, two additional items can be used to handle the prefix provided by this attribute:

Name	Valid locations	Description
$$_MY_PREFIX_$$	*initial attribute of <state> • event and target attribute of <transition>	During document assembly, $$_MY_PREFIX_$$ is replaced with the value of resolveid only in the defined locations. The engine does not perform global search and replace with this token.
_my_prefix	Any ECMAScript expression	During document assembly, <state>, <parallel>, and <final> is given a `_my_prefix` attribute extension that contains the value of resolveid. This allows the prefix value to be used in ECMAScript expressions within these states.

Children

The child element <fallback> is *not supported*.

Event Extensions

In addition to the standard properties in all events (see the Events section), the following are the additional attributes that are added to the SCXML events. They are accessible via the `_event.data` variable.

Event Name	Property	Description
error.illegalcond.errortype error.illegalloc.errortype error.illegalvalue.errortype error.illegaldata.errortype error.script.errortype error.unsupported.element error.receive.datamismatch error.send.nosuchsession (ORS only, since 8.1.2)	document	This is the URL of the document in which the error occurred. **Note:** this is important if the application uses `<xi:include>`.
	element	This is the name of the element that was being executed when the error occurred.
	linenumber	This is the line number of the document where the error occurred.

Event Name	Property	Description
error.send.datamismatch error.send.ioprocessorerror error.send.targettypeinvalid.stateid error.send.failed error.cancel.notallowed error.illegalassign error.send.noeventspecified		
error.badfetch.protocol.response_code error.send.targetunavailable.stateid	target	This is the target URL which was being used in the element when the error occurred.
	document	This is the URL of the document in which the error occurred. **Note:** this is important if the application uses `<xi:include>`.
	element	This is the name of the element that was being executed when the error occurred.
	linenumber	This is the line number of the document where the error occurred.
done.state.stateid done.scxml.sessionid done.invoke.invokeid	sessionid	This is the session ID of the session that has ended.
	reason	This is the reason the started session has ended.
	params	This is the list of parameters that is passed back based on the `<param>` elements defined in the `<donedata>` element in the `<final>` element which is executed while the session is finishing or terminating.

Logging and Metrics

The recorded format of a metric is typically of the form:

```
<metric_name data_name="value" data_name1="value" ... />
```

The following table describes the standard metrics:

Metric Name	Description	Associated Data
appl_begin	Logged when the session is started	Name Url (main Scxml document url)
appl_end	Logged when the session ends	
cancel	Recorded when a request to cancel a delayed <send> event is processed.	Sendid
doc_request	A document has been requested from the fetching service	Requested URL
doc_retrieved	A requested document has been retrieved	Requested URL Result (Success, Failure) Error Message Cache Hit (true/false)
eval_cond	A condition attribute is evaluated	Condition Result (true/false) Line Number
eval_expr	An expression is evaluated	Expression Result (true/false) Line Number
event_processed	A queued event has been processed	Event name Disposition (normal, terminated, no target)
event_queued	An event has been queued to the session. Note that events may appear more than once, as a result of deserialization.	Event name Queue Line Number (of the element that generated the event) Type (internal/external/platform)

Metric Name	Description	Associated Data
exec_error	An error was encountered while executing the document	Message
fm_exec_error	An FM encountered an error.	Function Scope Message
extension	A Custom Action Element has been selected for execution (but has not executed, yet).	Name Namespace
initial	An <initial> tag was entered.	
invoke	The <invoke> tag is about to be executed.	Target Type Target
js_diag	JS Diag data (occurs whenever max ops is reached; only available on Win32)	JSRuntime GC Bytes JSRuntime GC MaxBytes JSRuntime GC Max Malloc Bytes JSRuntime GC Last Bytes JSRuntime ID
log	Summarizes the result of a <log> tag.	Label Expression Level
onentry	The executable content of an <onentry > element is about to be run	Line Number State (name of containing state)
onexit	The executable content of an <onexit > element is about to be run	Line Number State (name of containing state)
param	The value of a param element passed to <invoke> and triggered sessions.	Name Value
parse_warning	A warning generated while parsing the document that does NOT result in a failure to parse or process the document	Message
send	The <send> tag has been selected for execution, but has not executed yet.	Target Target Type

Metric Name	Description	Associated Data
		Event Send ID
state_enter	The specified state has been entered	Name Type (standard, parallel, final, history, initial)
state_exit	The specified state has exited	Name Duration (the time in ms since the related state_enter metric was logged)
transition	The executable content of a <transition> is about to be run.	State (name of containing state) Condition (the string) Line Number Event Target
persist_store	A request to persist the session has been made.	Type (document, session) Key (for document type) Request ID (for session type) Message (for session type)
persist_result	A request to persist has completed.	Message
persist_restore	A request to restore the session has been made.	
persist_destroy	A request to delete the session from persistence has been made.	
persist_error	An error was encountered during persistence.	Type (document, session) Request Key Error Code Error Message
deactivate	Deactivation was requested for the session.	Status (success, failed)
restored	The session was restored.	

Supported URI Schemes

The following are the set of URI schemes that are supported:

- *HTTP* as defined by the RFC 2616
- *HTTPS* as defined by the RFC 2817
- *File* as defined by the RFC 1738
- *gesp* is a Genesys specific schema which is used to invoke an Interaction Server ESP function. **(For Orchestration only)**
  - FORMAT - `gesp: <applname>\ [<type>\]<service>\[method>]` For example, `gesp:CFGInteractionServer/Interaction/SubmitNew`
  - The following are the meanings of the different elements of the format:
    - `applname` is the 3rd party application that is to be used to process this request.
    - `service` is the name of the service with which this request is associated.
    - `method` is the specific function to be performed by the 3rd party application.
- *gdata* is a Genesys specific schema which is used to address configuration layer objects and options in Genesys Configuration Server, as well as data in an interaction's user data. Currently only supported in <submit> and <createmessage> actions. **(For Orchestration only)**
  - FORMAT - `gdata:[<host>:<port>/]<source> [/<objtype>.<objname>][/<p-name>]` For example, `gdata:config/AG.SalesGroup/supervisor`, `gdata:udata`
  - The following are the meanings of the different elements of the format:
    - `host` is the IP host address for the targeted Configuration Server. This is optional. This element is ignored if the source element is "udata"
    - <ocde>port is the port number for the targeted Configuration Server. This is optional. This element is ignored if the source element is "udata".
    - `source` is the source of the data. The following are the possible values:
      - *config* - data from Configuration Server.
      - *udata* - data from the associated interaction.
    - `objtype` is the type of configuration layer object. ***This element is only valid when the source element is "config".*** The values are the following:
      - *DN* - DN object
      - *SS* - Script object
      - *AG* - Agent Group object
      - *PG* - Place Group
      - *CA* - BA Category or Standard Response object
      - *CM* - Campaign object
      - *CL* - Calling ListTR-Transaction object
      - *AP* - Application object

    - *PE* - Person object.
    - *SK* - Skill object
    - *PL* - Place object
    - *ST* - Statistics Table
    - *SC* - BA Screening Rule object
- `objname` is the unique name (ID) for the configuration layer object that is being referenced. There is one exception where the name is not unique and this is with the CfgDN object. ***This element is only valid when the source element is "config".*** So the unique name will be the following combination of names:
    - *DN name* - This is the name of the CfgDN object.
    - *Switch name* - This is the name of the CfgSwitch object.
    - The format of the objname string in this case will be *dname@switchname.*
- "p-name" is the name of a specific property of the object. For properties like annex data. The p-name has the following format: "annex"/section-name/option-name. In the case where the source element value is "udata", this will be the key name of the user data that you want to use. For example, `gdata:udata/CategoryID`

Supported Profiles

The SCXML engine supports only the ECMAScript profile. Other profiles (such as minimal or XPath) will not be supported.

Examples

Using <xi:include>

The following example illustrates how xinclude can be used to compose several subroutine documents into the main application. The main application (complex_main.scxml) includes a subroutine (complex_sub) that is composed of two additional subroutines (compound_sub and simple_sub).

Main Document/SCXML Application (complex_main.scxml)

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml"
  xmlns:queue="http://www.genesyslab.com/modules/queue"
  xmlns:ixn="http://www.genesyslab.com/modules/interaction"
  xmlns:dialog="http://www.genesyslab.com/modules/dialog"
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <initial>
    <transition target="start"/>
  </initial>

  <state id="start" initial="complex.complex_sub">
    <onentry>
```

```
      <script>
        var args = new Object( );
        args.var1 = 3;
        args.var2 = 'data3';
      </script>
    </onentry>

    <transition event="done.state.complex.complex_sub" target="exit">
      <log expr="_event.data.val1"/>
      <log expr="_event.data.val2"/>
    </transition>
    <xi:include parse="xml" href="complex_sub.scxml" resolveid="complex" />
  </state>

  <final id="exit">
    <onentry>
      <log expr="'success!'"/>
    </onentry>
  </final>

  <final id="error">
    <onentry>
      <log expr="'failed!'"/>
    </onentry>
  </final>
</scxml>
```

Primary Subroutine (complex_sub.scxml)

```
  <state id="complex_sub" initial="$$_MY_PREFIX_$$.first_step" >
    <onentry>
      <script>
        var usefulName1 = args.var1;
        var usefulName2 = args.var2;
      </script>
    </onentry>
    <state id="first_step" initial="$$_MY_PREFIX_$$.compound.compound_sub">
      <onentry>
        <log expr="'Performing complex_sub.results calculation for ' + _my_prefix" />
        <script>
          var args = new Object( );
          args.var1 = 1;
          args.var2 = 'data1';
        </script>
      </onentry>
      <xi:include parse="xml" href="compound_sub.scxml" resolveid="compound"
xmlns:xi="http://www.w3.org/2001/XInclude" />
      <transition event="done.state.$$_MY_PREFIX_$$.compound.compound_sub"
target="$$_MY_PREFIX_$$.second_step">
        <log expr="_event.data.val1"/>
        <log expr="_event.data.val2"/>
      </transition>
    </state>
    <state id="second_step" initial="$$_MY_PREFIX_$$.simple.simple_sub">
      <onentry>
        <script>
          var args = new Object( );
          args.var1 = 2;
          args.var2 = 'data2';
        </script>
      </onentry>
      <xi:include parse="xml" href="simple_sub.scxml" resolveid="simple"
xmlns:xi="http://www.w3.org/2001/XInclude" />
```

```
      <transition event="simple_sub.results.*" target="$$_MY_PREFIX_$$.complete">
        <log expr="_event.data.val1"/>
        <log expr="_event.data.val2"/>
      </transition>
    </state>
    <final id="complete">
      <donedata>
        <param name="val1" expr="usefulName1"/>
        <param name="val2" expr="usefulName2"/>
      </donedata>
    </final>
  </state>
```

Nested Compound Subroutine (compound_sub.scxml)

```
  <state id="compound_sub" initial="$$_MY_PREFIX_$$.calculate" >
    <onentry>
      <script>
        var usefulName1 = args.var1;
        var usefulName2 = args.var2;
      </script>
    </onentry>
    <state id="calculate">
      <onentry>
        <log expr="'Performing compound_sub.results calculation for ' + _my_prefix" />
      </onentry>
      <transition target="$$_MY_PREFIX_$$.complete"/>
    </state>
    <final id="complete">
      <donedata>
        <param name="val1" expr="usefulName1"/>
        <param name="val2" expr="usefulName2"/>
      </donedata>
    </final>
  </state>
```

Nested Simple Subroutine (simple_sub.scxml)

```
  <state id="simple_sub">
    <onentry>
      <script>
        var usefulName1 = args.var1;
        var usefulName2 = args.var2;
      </script>
      <log expr="'Performing simple_sub.results calculation for ' + _my_prefix" />
      <raise event="simple_sub.results.success">
        <param name="val1" expr="usefulName1"/>
        <param name="val2" expr="usefulName2"/>
      </raise>
    </onentry>
  </state>
```

Fully Assembled SCXML Application Document

```
<scxml xmlns="http://www.w3.org/2005/07/scxml" version="1.0"
    xmlns:dialog="http://www.genesyslab.com/modules/dialog"
    xmlns:ixn="http://www.genesyslab.com/modules/interaction"
    xmlns:queue="http://www.genesyslab.com/modules/queue"
    xmlns:xi="http://www.w3.org/2001/XInclude">
  <initial>
    <transition target="start"/>
```

```
  </initial>

  <state id="start" initial="complex.complex_sub">
    <onentry>
      <script>
        var args = new Object( );
        args.var1 = 3;
        args.var2 = 'data3';
      </script>
    </onentry>
    <transition event="done.state.complex.complex_sub" target="exit">
      <log expr="_event.data.val1"/>
      <log expr="_event.data.val2"/>
    </transition>
    <state _my_prefix="complex" id="complex.complex_sub" initial="complex.first_step" >
      <onentry>
        <script>
          var usefulName1 = args.var1;
          var usefulName2 = args.var2;
        </script>
      </onentry>
      <state _my_prefix="complex" id="complex.first_step"
initial="complex.compound.compound_sub">
        <onentry>
          <log expr="'Performing complex_sub.results calculation for ' + _my_prefix"/>
          <script>
            var args = new Object( );
            args.var1 = 1;
            args.var2 = 'data1';
          </script>
        </onentry>
        <state _my_prefix="complex.compound" id="complex.compound.compound_sub"
initial="complex.compound.calculate">
          <onentry>
            <script>
              var usefulName1 = args.var1;
              var usefulName2 = args.var2;
            </script>
          </onentry>
          <state _my_prefix="complex.compound" id="complex.compound.calculate">
            <onentry>
              <log expr="'Performing compound_sub.results calculation for ' + _my_prefix"/>
            </onentry>
            <transition target="complex.compound.complete"/>
          </state>
          <final _my_prefix="complex.compound" id="complex.compound.complete">
            <donedata>
              <param expr="usefulName1" name="val1"/>
              <param expr="usefulName2" name="val2"/>
            </donedata>
          </final>
        </state>

        <transition event="done.state.complex.compound.compound_sub"
target="complex.second_step">
          <log expr="_event.data.val1"/>
          <log expr="_event.data.val2"/>
        </transition>
      </state>
      <state _my_prefix="complex" id="complex.second_step"
initial="complex.simple.simple_sub">
        <onentry>
          <script>
```

```
            var args = new Object( );
            args.var1 = 2;
            args.var2 = 'data2';
          </script>
        </onentry>
        <state _my_prefix="complex.simple" id="complex.simple.simple_sub">
          <onentry>
            <script>
              var usefulName1 = args.var1;
              var usefulName2 = args.var2;
            </script>
            <log expr="'Performing simple_sub.results calculation for ' + _my_prefix"/>
            <raise event="simple_sub.results.success">
              <param expr="usefulName1" name="val1"/>
              <param expr="usefulName2" name="val2"/>
            </raise>
          </onentry>
        </state>
        <transition event="simple_sub.results.*" target="complex.complete">
          <log expr="_event.data.val1"/>
          <log expr="_event.data.val2"/>
        </transition>
      </state>
      <final _my_prefix="complex" id="complex.complete">
        <donedata>
          <param expr="usefulName1" name="val1"/>
          <param expr="usefulName2" name="val2"/>
        </donedata>
      </final>
    </state>
  </state>
  <final id="exit">
    <onentry>
      <log expr="'success!'"/>
    </onentry>
  </final>
  <final id="error">
    <onentry>
      <log expr="'failed!'"/>
    </onentry>
  </final>
</scxml>
```

Output from Application

```
"Performing complex_sub.results calculation for complex"
"Performing compound_sub.results calculation for complex.compound"
"1"
"data1"
"Performing simple_sub.results calculation for complex.simple"
"2"
"data2"
"3"
"data3"
"success!"
```